Probabilistic Virtual Link Embedding under Demand Uncertainty

Fatemeh Hosseini, Alexander James, and Majid Ghaderi

Abstract—**This paper considers the problem of mapping virtual links to physical network paths, referred to as *Virtual Link Embedding (VLE)*, under the condition that bandwidth demands of virtual links are uncertain. To realize virtual links with predictable performance, the mapping is required to guarantee a bound on the congestion probability of the physical paths that embed the virtual links. To this end, we consider a general uncertainty model in which bandwidth demands of virtual links are expressed by random variables for which only the mean and variance (or a range) are known. We formulate the VLE problem as a nonlinear optimization program and design an algorithm called *Equal Partition VLE (epVLE)* to solve the problem by employing an approximate formulation that results in a second-order cone program (SOCP) that can be solved efficiently even for large networks. We then provide simulation results as well as model-driven and trace-driven experimental results from an SDN testbed to show the utility and efficiency of the epVLE algorithm in various network scenarios. We apply epVLE to commonly studied small networks as well as randomly generated large networks. Our results show that epVLE is able to satisfy the required link congestion constraint, and that it produces results that are very close to those obtained from the exact optimization model.**

Index Terms—**Network virtualization, Virtual link embedding, Uncertain demands, Congestion probability.**

I. Introduction

Background and Motivation. The problem of embedding virtual networks into a physical substrate network, known as *virtual network embedding (VNE)*, is an important problem in network virtualization. The VNE problem is computationally hard due to its combinatorial structure [1], and has been studied extensively (comprehensive surveys on the topic are presented in [2] and [3]). In its abstract form, a VN is represented by a set of virtual nodes and virtual links. Virtual nodes and links require specific amount of resources depending on the services provided by the corresponding VNs. The VNE problem is then to find a virtual to physical node-to-node and link-to-path mapping that does not exceed the node and link capacities of the physical network.

Most of the existing works on VNE assume that resource demands of VNs are known *deterministically* [4]–[11], *i.e.*, the resource demands are fixed and known a priori by the mapping algorithm. In a real deployment though, the node and link demands (*i.e.*, processing power and bandwidth) include significant *uncertainty* (*e.g.*, because of estimation errors or variability over time) [12]. As such, when employing deterministic embedding algorithms, one has to consider either the "worst-case" or "average" resource demands for each VN. Both approaches, however, lead to inefficient use of network resources depending on how much the actual demands deviate from their presumed *nominal* values. Recently, a few works have considered uncertainty in virtual network resource demands [13]–[18] with the objective of finding a virtual-to-physical mapping that is feasible even when resource demands deviate from their nominal values, without sacrificing the utilization of physical resources. In these works, the embedding algorithm computes a mapping that satisfies virtual resource demands *probabilistically*. In other words, the computed mapping may not be feasible in certain scenarios, *e.g.*, when all resource demands deviate significantly from their nominal values. Consequently, "congestion" occurs at a node or link where the allocated physical resources are insufficient to meet the virtual resource demands. Congestion is detrimental to the performance of VNs and results in unpredictable network performance (*e.g.*, unpredictable packet loss or latency).

In this paper, we focus on mapping virtual links to physical paths, referred to as the *Virtual Link Embedding (VLE)* problem, under demand uncertainty, where the objective is to minimize congestion in the substrate network. We show that under the constraint of guaranteed virtual link congestion, the mapping problem becomes considerably different and more challenging compared to existing formulations without link congestion guarantee.

Our Work. To guarantee a given congestion probability for a virtual link, the end-to-end congestion on the corresponding physical path must be bounded. Consider the simple example depicted in Fig. 1, in which virtual link ℓ between virtual nodes v_1 and v_2 is mapped to path $P = \langle e_1, e_2, e_3, e_4\rangle$ in the substrate network (*i.e.*, v_1 and v_2 are placed on n_1 and n_2). Suppose that the pre-specified bound on the congestion probability of virtual link ℓ is given by ϵ. Let ε_k denote the congestion probability on physical link e_k. In order to satisfy the virtual link congestion requirement ϵ, the following constraint should be satisfied,

$$1 - \textstyle\prod_{e_k \in P}(1 - \varepsilon_k) \le \epsilon\,. \qquad (1)$$

Notice that ε_k is a function of the total bandwidth demand on link e_k, and should be (optimally) determined by the embedding algorithm. The complexity of the problem arises from the fact that 1) the optimal path connecting the physical nodes n_1 and n_2 is not known in advance, and 2) the bandwidth demand on each link, and consequently the link congestion probabilities are functions of the unknown paths. In other words, the problem is a joint optimization of path selection and link congestion allocation, which as we show later, results in a non-convex nonlinear optimization problem that is computationally difficult to solve for large networks.

As mentioned earlier, most existing works assume deter-

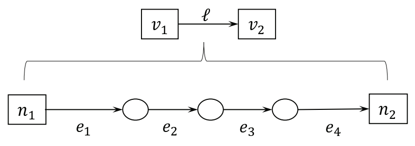


Fig. 1: Virtual link to physical path mapping.

ministic demands, and thus do not consider congestion at all. Those works that do consider congestion, assume that it is sufficient to guarantee a fixed congestion probability for every physical link (*e.g.*, [13]–[18]). As such, the actual end-to-end congestion probability achieved by these works depends on the length of the physical path, and hence cannot be guaranteed. For example, consider a link embedding algorithm that guarantees congestion probability ε on every physical link. When applied to the link mapping depicted in Fig. 1, the resulting congestion probability for virtual link ℓ is given by $1-(1-\varepsilon)^4$, which may, or may not satisfy the required congestion ϵ. If the same link ℓ was mapped to a different path with 2 links, then the resulting congestion probability would be $1-(1-\varepsilon)^2$. Our goal in this work is to design a mapping algorithm that guarantees a required end-to-end congestion probability regardless of the characteristics of the underlying paths through the substrate network.

This paper does not attempt to address all aspects of VNE under demand uncertainty. Instead, we focus on the problem of mapping virtual links to physical paths with guaranteed end-to-end congestion probabilities. However, we believe that the ideas and formulations presented in this paper can often be incorporated into existing VNE solutions. Specifically, if link embedding is performed in the VNE solution after node embedding (*e.g.*, as in [9], [19]), it is possible to directly use our algorithm for link embedding. We briefly touch on the node and link embedding integration issues in Section V.

Contributions. The contributions of this paper are:

- We consider demand uncertainty in a flexible and general model, where only limited information about resource demands, namely mean and variance, is needed.
- We formulate the virtual link embedding problem with constrained end-to-end congestion probability as a non-linear optimization program that can be solved using global optimization solvers for small network instances.
- To solve the problem efficiently, we propose an algorithm called Equal Partition VLE (epVLE), which is based on an approximate formulation of the problem as a second-order cone program (SOCP) that can be solved in polynomial time even for large network instances.
- We present simulation and real SDN testbed experimental results to show the efficiency and utility of epVLE in a variety of network scenarios.

Paper Organization. The remainder of this paper is organized as follows. Section II presents a survey of related works. We discuss our model and assumptions in Section III. Section IV is dedicated to the derivation of exact and approximate problem formulations. A discussion on how to incorporate our formulation in existing node embedding algorithms is presented in Section V. Performance evaluation results are presented in Section VI. Concluding remarks are discussed in Section VII.

II. Related Work

Most works on VNE have focused on link-by-link emedding constraints as opposed to end-to-end requirements. There are some works that consider end-to-end delay requirements, however to the best of our knowledge no existing work has considered guaranteed end-to-end congestion probability when embedding virtual links, which is the problem considered here.

Deterministic Approaches. The deterministic VNE problem is extensively studied in the literature. For example, the survey paper [2] alone lists 78 algorithms. The works in this category, generally formulate the VNE problem as a (mixed) integer program, and then try to solve it either exactly [4], [5] or by devising heuristic [6]–[9] or approximation [10], [11] algorithms. These works differ from each other in terms of the objectives considered (*e.g.*, minimizing the cost of resources, network energy consumption or minimizing the maximum link utilization) and the constraints imposed on node and link embedding (*e.g.*, delay, routing and location constraints).

Stochastic Approaches. This category includes works that assume the distribution of bandwidth demands is fully known. Their roots can be traced back to the literature on the effective bandwidth concept. For instance, the works [13]–[15] assume that virtual node and link demands follow a Normal distribution. In this case, the congestion probability at each physical node or link, *i.e.*, the probability that the aggregate demand exceeds available resources, can be computed using the tail probability of Normal distribution. A rather different approach based on demand prediction is proposed in [16], where a seasonal ARMA model is employed to estimate the unknown future demands. Then, to compute the link congestion probability, it is assumed that the prediction errors of the ARMA model follow a zero-mean Normal distribution for which the variance can be estimated from past observations.

Robust Approaches. This category includes works that assume while the distribution of demands is unknown, some uncertainty model can be used to describe their variability. In [20], the authors consider a model in which the demand uncertainty is described by a number of scenarios. Each scenario corresponds to a specific set of resource demands. The work [21] presents a hose-model formulation in which the computed embedding can support any traffic matrix consistent with the given aggregate demands between pairs of virtual nodes. A hybrid robust-reactive approach is presented in [22] which dynamically modifies resource reservations at runtime to satisfy minimum guarantees on link bandwidth and node processing demands. A popular approach in designing robust VNE solutions is to employ techniques from the chance-constrained robust optimization [23] and Γ-robust optimization [24] to formulate VNE as a robust optimization problem. For example, [17] formulates the problem as a robust optimization problem, where the objective is to provide link-by-link congestion probability guarantees. An example using Γ-robust formulation is [18]. In this approach, the level of uncertainty in demands can be controlled by parameter Γ, which determines how many demands can maximally deviate from their nominal values simultaneously.

TABLE I: Principal Notation Used in the Paper.

Symbol	Definition
Input parameters	
$\mathcal{N}$	Set of physical nodes in the substrate network
$\mathcal{E}$	Set of physical links in the substrate network
$\mathcal{L}$	Set of virtual links from all virtual networks
ℓ_i	Virtual link i ($\ell_i \in \mathcal{L}$)
e_k	Physical link k ($e_k \in \mathcal{E}$)
$O(\ell_i)$	Physical origin node of ℓ_i
$D(\ell_i)$	Physical destination node of ℓ_i
C_k	Capacity of physical link e_k
B_i	Bandwidth demand of virtual link ℓ_i
$\mathcal{P}_i$	Set of candidate physical paths for $\ell_i \in \mathcal{L}$
$\mathcal{P}$	Set of all candidate paths in substrate network
$\lvert P_j \rvert$	Length of path $P_j \in \mathcal{P}$
ϵ_i	Required congestion probability on virtual link ℓ_i
Decision variables	
$0 \leq x_{ij} \leq 1$	Fraction of demand B_i on path $P_j \in \mathcal{P}_i$
$0 \leq y_{ik} \leq 1$	Total fraction of demand B_i on link $e_k \in \mathcal{E}$
$0 \leq \varepsilon_k \leq 1$	Congestion probability on link $e_k \in \mathcal{E}$
$\alpha \geq 0$	Utilization of the most congested physical link

Online Approaches. In offline approaches, a set of virtual networks are given and the algorithm has to compute an embedding in a one shot solution. In real situations though, virtual network requests arrive sequentially over time and the embedding decisions have to made at runtime without knowing the sequence of future virtual network requests. Works that consider online VNE are scarce. To this end, [25] designs a competitive online algorithm, while [26], [27] use machine learning (*i.e.*, neural networks and reinforcement learning) to design online solutions.

VNF Placement. A special case of VNE is the *virtual network function (VNF)* placement problem [28]. The VNF placement problem deals with embedding of so-called service function chains (SFCs), which can be thought of as virtual networks with specialized topologies (*e.g.*, a line) and a specific order of virtual nodes that are visited as traffic traverses the network. There are several works on VNF placement that consider demand uncertainty. For example, the recent works [29], [30] consider VNF placement with end-to-end requirements. However, the former considers end-to-end chain requirements that are independent of the traffic load passing through the links (*e.g.*, fixed propagation delays on each link), while the latter considers end-to-end delay in a load dependent model, but does not consider congestion due to over-subscription of physical resources, as we consider in this paper. Other notable examples are [31], [32] in which the Γ-robust framework is used to model uncertainty in VNF compute demands while guaranteeing end-to-end delay when placing VNFs.

III. System Model and Assumptions

In this section, we present our system model and describe how demand uncertainty and link congestion are formulated. Table I lists the principal notation used throughout the paper.

A. Physical Network

The physical network is specified by an undirected graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ denotes the set of physical nodes and $\mathcal{E}$ denotes the set of physical links (or edges) between the nodes. Each physical link e_k has a fixed capacity denoted by $C_k > 0$.

B. Virtual Network Requests

Setting up virtual networks takes time. Therefore, we assume that virtual network (VN) requests are processed in batches. Time is divided into time intervals, and all embedding decisions for a time interval are made at the beginning of that interval. The batch of VN requests processed at the beginning of an interval includes requests that have arrived during the previous time interval. Each virtual network request is represented as a weighted undirected graph with a given set of nodes and links. The weight of each link indicates the (uncertain) bandwidth demand of that link. The objective is to find a mapping from virtual nodes to physical nodes and virtual links to physical paths so that the resource requirements of all VNs are met while the node and link capacity constraints of the physical network are not violated.

C. Virtual Link Embedding

Let $O(\ell_i) \in \mathcal{N}$ and $D(\ell_i) \in \mathcal{N}$, for virtual link ℓ_i, denote the physical nodes embedding the origin and destination of virtual link ℓ_i, respectively. The VLE problem is to map every virtual link $\ell_i \in \mathcal{L}$ to a *set* of physical paths connecting $O(\ell_i)$ to $D(\ell_i)$ in the substrate network so that the virtual link ℓ_i satisfies the target congestion probability ϵ_i. This model allows multi-path routing, however, we limit the number of candidate paths to a fixed number independent of the size of virtual networks. While multi multi-path routing requires explicit network support, we note that all modern networks rely on multi-path routing as the foundation of traffic engineering for implementing load balancing, increasing network robustness and reducing costs [19], [33], [34]. Specifically, our model is based on static flow splitting over a fixed number of paths at the source, which is straightforward to implement in current software-defined networks (SDNs) using commodity off-the-shelf OpenFlow switches (see our testbed description in subsection VI-E) and has been shown to scale to large production networks [34], [35]. One of the main drawbacks of multi-path routing is the potential packet reordering at the destination. Since different paths could have different round-trip times, multi-path routing may result in out-of-order packets at the destination, which could negatively affect applications relying on TCP. To deal with reordered packets, switch-level (*e.g.*, FLARE [33]) or host-level mechanisms (*e.g.*, reordering robust TCP [36]) can be used to ensure applications are not affected.

D. Bandwidth Demand Uncertainty Model

Let B_i denote the (uncertain) bandwidth demand of virtual link $\ell_i \in \mathcal{L}$. We assume that only limited statistical information about B_i is available. Specifically, we assume that the mean and variance of B_i, denoted by $\mathbb{E}\left[B_i\right] = \mu_i$ and $\mathrm{Var}\left[B_i\right] = \sigma_i^2$, are known. It is relatively straightforward to estimate the mean and variance based on historical traffic data [37]. In fact, as long as σ_i^2 provides an *upper bound* on the actual variance of B_i, our formulation holds. We emphasize that the considered uncertainty model is quite general. For example, in the literature on robust optimization, a common uncertainty model is the so-called box uncertainty model [23].

In this model, each uncertain variable B_i is allowed to deviate from its nominal value by a maximum deviation δ_i. That is, $B_i \in [\mu_i - \delta_i, \mu_i + \delta_i]$. Our uncertainty model can easily accommodate the box model by computing an upper bound on the variance of an *arbitrarily* distributed random variable that is confined to interval $[\mu_i - \delta_i, \mu_i + \delta_i]$. In this case, it can be shown that $\sigma_i^2 \leq \delta_i^2$, with the equality attained when all the probability mass is assigned to the extreme points of the interval. In general, for a random variable B_i that is confined to an interval $[a_i, b_i]$, we have that $\sigma_i^2 \leq (b_i - a_i)^2/4$. Alternatively, a less conservative approach is to assume B_i is *uniformly* distributed over the interval, which leads to $\sigma_i^2 = \delta_i^2/3$.

E. Congestion Probability

Let W_{ik} denote the bandwidth demand of virtual link $\ell_i \in \mathcal{L}$ on physical link $e_k \in \mathcal{E}$, where $\mathbb{E}[W_{ik}] = \mu_{ik}$ and $\mathrm{Var}[W_{ik}] = \sigma_{ik}^2$. Note that W_{ik} has to be determined by the embedding algorithm based on bandwidth demands of all virtual links (*i.e.*, B_i's) and capacities of all physical links (*i.e.*, C_k's). The congestion probability on physical link e_k is then given by,

$$\mathbb{P}\{\text{congestion on } e_k\} = \mathbb{P}\left\{\sum_{\ell_i \in \mathcal{L}} W_{ik} \geq C_k\right\}. \quad (2)$$

For simplicity of notation, in the following derivation, we abbreviate the summation index and use i in place of $\ell_i \in \mathcal{L}$. To avoid making any assumptions about the distribution of the uncertain bandwidth demands, we use a *concentration bound* to estimate (2). There are several forms of concentration bounds which can be used based on how much information about the distribution of the uncertain demands is available. In this work, we use the Chernoff bound, as follows:

$$\mathbb{P}\left\{\sum_i W_{ik} \geq C_k\right\} \leq \inf_{\theta \geq 0} \frac{\mathbb{E}\left[e^{\theta \sum_i W_{ik}}\right]}{e^{\theta C_k}}. \quad (3)$$

If $\mathrm{Var}[W_{ik}]$ is bounded, *i.e.*, $\mathrm{Var}[W_{ik}] \leq \sigma_{ik}^2$, then we have [38],

$$\mathbb{E}\left[e^{\theta W_{ik}}\right] \leq e^{\mu_{ik}\theta + \frac{1}{2}\sigma_{ik}^2\theta^2}. \quad (4)$$

Noting that W_{ik}'s are independent from each other and taking the derivatives of the terms inside the exponent with respect to θ, we see that $\theta = \frac{C_k - \sum_i \mu_{ik}}{\sum_i \sigma_{ik}^2} \geq 0$ minimizes the expression on the right-hand side of the inequality. Consequently, it follows that,

$$\mathbb{P}\left\{\sum_i W_{ik} \geq C_k\right\} \leq \exp\left(-\frac{(\sum_i C_k - \sum_i \mu_{ik})^2}{2\sum_i \sigma_{ik}^2}\right). \quad (5)$$

In the following sections, we use (5) to compute congestion probability on each physical link in the substrate network.

IV. Virtual Link Embedding

In this section, we present exact and approximate VLE formulations. Recall that each virtual link $\ell_i \in \mathcal{L}$ can be mapped to multiple paths from a set of candidate physical paths $\mathcal{P}_i$. In other words, $\mathcal{P}_i$ consists of multiple paths between origin and destination nodes $O(\ell_i)$ and $D(\ell_i)$. Denote the set of all candidate paths in the substrate network by $\mathcal{P}$, that is $\mathcal{P} = \cup_{\ell_i \in \mathcal{L}} \mathcal{P}_i$. Let x_{ij}, for $0 \leq x_{ij} \leq 1$, denote the fraction of bandwidth demand B_i that is allocated on path $P_j \in \mathcal{P}_i$ for virtual link ℓ_i. Our goal is to find the routing variables x_{ij} that minimize the utilization of the most congested link in the substrate network subject to a constraint on the end-to-end congestion probability of each path. To avoid splitting the demand of a virtual link across many paths in the substrate network, the number of paths for each virtual link can be limited to only a few paths, *e.g.*, first K shortest paths, as done in our evaluations. Indeed, in practical scenarios, often a few paths are sufficient to achieve a performance that is close to that of a solution that utilizes all available paths in the network [34], [39].

A. Congestion on Physical Links

Let y_{ik}, for $0 \leq y_{ik} \leq 1$, denote the total fraction of bandwidth demand B_i for virtual link $\ell_i \in \mathcal{L}$ that is allocated on physical link $e_k \in \mathcal{E}$. We have,

$$y_{ik} = \textstyle\sum_{P_j \in \mathcal{P}_i} x_{ij} \cdot \mathbf{I}_{e_k \in P_j}, \quad (6)$$

where $\mathbf{I}_{e_k \in P_j}$ denotes the indicator function, which is 1 if $e_k \in P_j$, and 0 otherwise. Notice that $W_{ik} = y_{ik}B_i$, and thus, $\mathbb{E}[W_{ik}] = y_{ik}\mu_i$ and $\mathrm{Var}[W_{ik}] = y_{ik}^2\sigma_i^2$. Next, applying (5), for a desired link utilization $\alpha \geq 0$, it is obtained that,

$$\mathbb{P}\left\{\sum_{\ell_i \in \mathcal{L}} y_{ik}B_i \geq \alpha C_k\right\} \leq \exp\left(-\frac{(\alpha C_k - \sum_{\ell_i \in \mathcal{L}} y_{ik}\mu_i)^2}{2\sum_{\ell_i \in \mathcal{L}} y_{ik}^2\sigma_i^2}\right).$$

Therefore, to restrict the congestion probability on physical link e_k by ε_k, the following inequality should be satisfied,

$$\exp\left(-\frac{(\alpha C_k - \sum_{\ell_i} y_{ik}\mu_i)^2}{2\sum_{\ell_i} y_{ik}^2\sigma_i^2}\right) \leq \varepsilon_k, \quad (7)$$

which leads to the following inequality,

$$\left(2\ln\frac{1}{\varepsilon_k}\right)\sum_{\ell_i \in \mathcal{L}} \sigma_i^2 y_{ik}^2 \leq \Big(\alpha C_k - \sum_{\ell_i \in \mathcal{L}} \mu_i y_{ik}\Big)^2. \quad (8)$$

Notice that ε_k is a decision variable whose optimal value needs to be specified by the embedding algorithm.

B. Congestion on Physical Paths

A path is a sequence of links, thus, we have,

$$\mathbb{P}\{\text{congestion on path } P_j\} = 1 - \textstyle\prod_{e_k \in P_j}(1 - \varepsilon_k). \quad (9)$$

In practice, we have $\varepsilon_k \ll 1$, and thus using the union bound, we obtain the following linear approximation for the path congestion probability,

$$\mathbb{P}\{\text{congestion on path } P_j\} \leq \textstyle\sum_{e_k \in P_j} \varepsilon_k. \quad (10)$$

Therefore, to restrict the congestion probability on path $P_j \in \mathcal{P}_i$ by a given ϵ_i, the following inequality should be satisfied,

$$\sum_{e_k \in P_j} \varepsilon_k \leq \epsilon_i, \quad (11)$$

which is a linear constraint.

C. Exact VLE Formulation

Problem Formulation. The VLE problem can be formulated as a non-linear optimization problem, as presented in Problem 1, where,

Problem 1: Exact VLE.

$$\begin{aligned}
&\text{minimize} \quad \alpha \\
&\text{subject to:} \\
&\sum_{P_j \in \mathcal{P}_i} x_{ij} = 1, && \forall \ell_i \in \mathcal{L} \quad (12a)\\
&y_{ik} = \sum_{P_j \in \mathcal{P}_i} x_{ij} \mathbf{I}_{e_k \in P_j}, && \forall \ell_i \in \mathcal{L}, \forall e_k \in \mathcal{E} \quad (12b)\\
&\sum_{e_k \in P_j} \varepsilon_k \le \epsilon_i, && \forall P_j \in \mathcal{P}_i \quad (12c)\\
&\big(2\ln\frac{1}{\varepsilon_k}\big) \sum_{\ell_i \in \mathcal{L}} \sigma_i^2 y_{ik}^2 \le \big(\alpha C_k - \sum_{\ell_i \in \mathcal{L}} \mu_i y_{ik}\big)^2, && \forall e_k \in \mathcal{E} \quad (12d)\\
&x_{ij}, y_{ij} \in [0,1], && (12e)\\
&\alpha \ge 0\,. && (12f)
\end{aligned}$$

- Constraint (12a) enforces bandwidth embedding over all possible candidate paths for embedding link ℓ_i.
- Constraint (12c) enforces end-to-end congestion probability ϵ_i on candidate paths for embedding link ℓ_i.
- Constraint (12d) enforces link congestion probability ε_k on link e_k, so as to satisfy Constraint (12c).

The objective of the optimization problem is to minimize α. It is then guaranteed that the utilization of every link in the substrate network will be at most equal to α. To see this, notice that if the inequality constraint (12d) is active for some link, then the utilization of that link is exactly equal to α. Otherwise, the utilization of the link is less than α. Thus, the variable α denotes the maximum utilization of any link in the substrate network (or equivalently, the utilization of the most congested link in the network).

The input to the VLE problem is a set of VN requests. All VNs in the set are embedded in one shot, and the problem always has a solution as variable α can be set large enough to satisfy the congestion requirement on every link. The resulting embedding, however, is infeasible if $\alpha > 1$.

Theorem 1. *Problem 1 (Exact VLE) is non-convex.*

Proof. All the constraints in Problem 1 are linear except (12d), which is nonlinear. We will show that this constraint is non-convex too. To this end, consider the equivalent form of this constraint in (7). A real function $f : \mathcal{X} \to \mathbb{R}$ is convex (over the convex set $\mathcal{X}$), iff $\forall x_1, x_2 \in \mathcal{X}, \forall t \in [0,1]$:

$$f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)\,. \tag{13}$$

To show non-convexity, it is sufficient to find x_1, x_2 and t that violate the above. For ease of exposition, assume that physical link $e_k \in \mathcal{E}$ is shared between two virtual links ℓ_1 and ℓ_2 and that $\mu_1 = \mu_2 = 1$ and $\sigma_1 = \sigma_2 = 1$. Set $\alpha = 1$ and $C_k = 1$. Let function $f(\cdot)$ denote the left-hand side of inequality (14). We have,

$$f(y_{1k}, y_{2k}) = \exp\left(-\frac{(1 - y_{1k} - y_{2k})^2}{2(y_{1k}^2 + y_{2k}^2)}\right)\,. \tag{14}$$

Function $f(\cdot)$ is plotted in Fig. 2. The non-convexity of $f(\cdot)$ can be seen by inspecting the graph, and can be verified by considering two points $z_1 = f(0.1, 0.1) = 1.12535 \times 10^{-7}$ and $z_2 = f(0.9, 0.9) = 0.820755$. The middle point of the line that connects $(0.1, 0.1, z_1)$ and $(0.9, 0.9, z_2)$, *i.e.*, set $t = 0.5$, is point $(0.5, 0.5, 0.410378)$ with $f(0.5, 0.5) = 1$. Since, $1 > 0.410378$, function $f(\cdot)$ is not convex. □

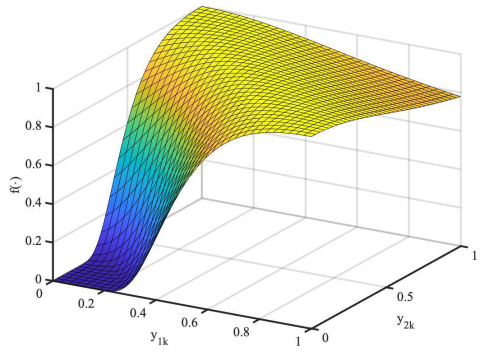


Fig. 2: Function $f(\cdot)$ in (14).

Discussion. To solve Problem 1, one approach is to use a global nonlinear solver such as Knitro [40]. We have implemented this approach and experimented with various network configurations. While it is possible to solve the problem for small network instances, it takes a prohibitively large amount of time to solve the problem for any realistic network size (*e.g.*, hours). Moreover, since the problem is not convex, the computed solutions may not even be globally optimal. Therefore, to solve the problem for large network instances, we design an approximate solution, as presented in the next sub-section. We show that the approximation results in a second-order cone program (SOCP) [41] that can be solved efficiently (in polynomial time) using conventional solvers such as Gurobi [42].

D. Approximate VLE Formulation

The complication in Constraint (12d) is due to the fact that the program tries to *optimally* assign congestion probabilities to each link on a given path. If we could pre-compute ε_k for each link e_k, then Problem 1 could be converted to a SOCP, as established in Lemma 1.

Lemma 1. *Problem 1 (Exact VLE) reduces to a second-order cone program if ε_k is fixed for all $\ell_k \in \mathcal{E}$.*

Proof. Let $\theta_k = 2\ln\frac{1}{\varepsilon_k}$. Then, constraint (12d) is equivalent to the following system of equations,

$$\textstyle\sum_{\ell_i} \sigma_i^2 y_{ik}^2 \le u_{ik}^2, \tag{15a}$$

$$u_{ik} = \frac{1}{\sqrt{\theta_k}}\Big(\alpha C_k - \textstyle\sum_{\ell_i} \mu_i y_{ik}\Big)\,. \tag{15b}$$

Notice that the above inequality is a second order cone, while the equality is linear for a fixed θ_k. Recall that the objective as well as all other constraints in Problem 1 are linear. Thus, the formulation with fixed ε_k reduces to a SOCP. □

Equal Partition Approximation. Our approximate formulation is based on the simplification that all links in a path achieve the same congestion probability (*i.e.*, *equal partition* allocation). Clearly, this results in a sub-optimal solution because the optimal solution may assign different congestion

Algorithm 1: epLCA – Link Congestion Assignment.

```
procedure epLCA($\mathcal{L}$, $\mathcal{P}$, $\{\epsilon_i\}$)
    foreach $\ell_i \in \mathcal{L}$ do
        foreach $P_j \in \mathcal{P}_i$ do
            $\pi[P_j] \leftarrow 1 - \sqrt[|P_j|]{1-\epsilon_i}$
            foreach $e_k \in P_j$ do
                $\varepsilon_k \leftarrow 0$
    $\vec{\mathcal{P}} \leftarrow$ sort $\mathcal{P}$ based on increasing order of $\pi$
    for $j \leftarrow 1$ to $|\vec{\mathcal{P}}|$ do
        $\lambda \leftarrow 1$
        $P_j \leftarrow \vec{\mathcal{P}}[j]$
        $\widehat{P}_j \leftarrow \{e_k \in P_j | \varepsilon_k = 0\}$
        foreach $e_k \in P_j \setminus \widehat{P}_j$ do /*links already allocated*/
            $\lambda \leftarrow \lambda \times (1 - \varepsilon_k)$
        foreach $e_k \in \widehat{P}_j$ do /*links that need to be allocated*/
            $\varepsilon_k = 1 - \sqrt[|\widehat{P}_j|]{\frac{1-\epsilon_i}{\lambda}}$ /*cong $\epsilon_i$ is for $P_j \in \mathcal{P}_i$*/
            $\theta_k = 2\ln(1/\varepsilon_k)$
    return $\{\theta_k\}$
```

probabilities to different links on the same path. However, it has been shown that when the end-to-end congestion bound ϵ_i (for each virtual ℓ_i) is very *small*, there is little difference in the performance of different congestion allocation policies [43][1]. From a practical point of view, the equal partition approximation is indeed quite accurate, as will be shown in Section VI. Recall that, our focus is on applications that are congestion sensitive and thus require very small end-to-end congestion probability. For instance, applications relying on TCP, generally require end-to-end packet loss rates of only a few percentage points to operate effectively (otherwise, TCP connections break down).

Link Congestion Assignment. Let ε_j denote the link congestion probability for each link in path P_j, that is $\varepsilon_k = \varepsilon_j$, for all $e_k \in P_j$. Let $|P_j|$ denote the length of path P_j. We have,

$$\mathbb{P}\{\text{congestion on path } P_j\} = 1 - (1-\varepsilon_j)^{|P_j|}. \tag{16}$$

Therefore, to satisfy the end-to-end path congestion probability ϵ_i, we obtain that,

$$1 - (1-\varepsilon_j)^{|P_j|} \le \epsilon_i \Leftrightarrow \varepsilon_j \le 1 - \sqrt[|P_j|]{1-\epsilon_i}. \tag{17}$$

One problem arising from the above congestion probability allocation policy is that a link may be common among multiple paths requiring different end-to-end congestions. In such cases, the path whose links have the most stringent congestion requirement determine the allocated congestion on the common links. As a result, the congestion probability assignment for the uncommon links must be updated (*i.e.*, increased) to satisfy the end-to-end path congestion constraint, as described next. Consider some path P_j. Let $\widehat{P}_j$ denote the set of links in this path whose congestion probabilities are not assigned yet. For the remaining links in the path, *i.e.*, $e_k \in P_j \setminus \widehat{P}_j$, their congestion probabilities have already been assigned as they are common with some other paths with lower link congestion requirement. Let $\widehat{\varepsilon}_j$ denote the congestion probability that should be assigned to every link in $\widehat{P}_j$. The following relation should be satisfied:

$$1 - (1-\widehat{\varepsilon}_j)^{|\widehat{P}_j|} \prod_{e_k \in P_j \setminus \widehat{P}_j} (1-\varepsilon_k) \le \epsilon_i, \tag{18}$$

which yields the following relation,

$$\widehat{\varepsilon}_j \le 1 - \sqrt[|\widehat{P}_j|]{\frac{1-\epsilon_i}{\prod_{e_k \in P_j \setminus \widehat{P}_j} (1-\varepsilon_k)}}. \tag{19}$$

The congestion assignment algorithm called epLCA is described in Algorithm 1. The first for loop is to compute the default link congestion requirements, which are then used to sort paths from the smallest to largest link congestion requirement. The second for loop iterates over the paths and for each path adjusts its link congestion requirements based on (19).

Problem 2: Approximate VLE.

minimize α

subject to:

$$\sum_{P_j \in \mathcal{P}_i} x_{ij} = 1, \qquad \forall \ell_i \in \mathcal{L} \tag{20a}$$

$$y_{ik} = \sum_{P_j \in \mathcal{P}_i} x_{ij} \mathbf{I}_{e_k \in P_j}, \qquad \forall \ell_i \in \mathcal{L}, \forall e_k \in \mathcal{E} \tag{20b}$$

$$u_{ik} = \frac{1}{\sqrt{\theta_k}} \Big(\alpha C_k - \textstyle\sum_{\ell_i \in \mathcal{L}} \mu_i y_{ik} \Big), \forall \ell_i \in \mathcal{L}, \forall e_k \in \mathcal{E} \tag{20c}$$

$$\textstyle\sum_{\ell_i \in \mathcal{L}} \sigma_i^2 y_{ik}^2 \le u_{ik}^2, \qquad \forall e_k \in \mathcal{E} \tag{20d}$$

$$x_{ij}, y_{ik} \in [0,1], \tag{20e}$$

$$\alpha \ge 0. \tag{20f}$$

Lemma 2. *Algorithm 1 (Link Congestion Assignment) runs in* $O\big(|\mathcal{P}| \cdot (\log|\mathcal{P}| + \max_{P_j \in \mathcal{P}} |P_j|)\big)$ *time.*

Proof. Each of the main for loops runs in $O(\sum_{P_j \in \mathcal{P}} |P_j|)$ time, while sorting $|\mathcal{P}|$ paths takes $O(|\mathcal{P}| \log |\mathcal{P}|)$ time. The lemma is established by noting that $\sum_{P_j \in \mathcal{P}} |P_j| \le |\mathcal{P}| \max_{P_j \in \mathcal{P}} |P_j|$. □

Observation. Since each virtual link is mapped to a constant number of paths, the running time of Algorithm 1 can be expressed as $O(|\mathcal{L}| \cdot (\log|\mathcal{L}| + |\mathcal{E}|))$, which is dominated by the number of virtual links.

Approximate Algorithm. By fixing the link congestion probabilities, the optimization problem formulated in Problem 1 is reduced to the SOCP problem presented in Problem 2. Once the routing variables x_{ij} are computed, we may find that some paths are not used by any virtual network. Thus, we can adjust link congestion probabilities accordingly. To adjust link congestion probabilities, we simply remove the unused paths, re-assign link congestion probabilities and solve the optimization problem again. The *Equal Partition VLE* (epVLE) algorithm based on the approximate VLE formulation is presented in Algorithm 2.

Theorem 2. *Algorithm epVLE runs in* $O\big(|\mathcal{P}|(\log|\mathcal{P}| + \max_{P_j \in \mathcal{P}} |P_j|) + |\mathcal{P}|^{3.5}\big)$ *time.*

Proof. A SOCP program with $|\mathcal{P}|$ decision variables can be solved in $O(|\mathcal{P}|^{3.5})$ time using the interior point methods. The proof then follows from Lemmas 1 and 2. □

[1]The results in [43] concern the end-to-end *packet loss probability*, which is directly related to the congestion probability considered in our model.

Algorithm 2: epVLE – Equal Partition VLE.

```
procedure epVLE($\mathcal{L}$, $\mathcal{B}$, $\mathcal{P}$, $\{\epsilon_i\}$)
    $\{\theta_k\} \leftarrow$ epLCA($\mathcal{L}$, $\mathcal{P}$, $\{\epsilon_i\}$)
    SOCP($\mathcal{L}, \mathcal{B}, \mathcal{P}, \{\theta_k\}$)
    foreach $P_j \in \mathcal{P}$ do
        if $\sum_{\ell_i \in \mathcal{L}} x_{ij} = 0$ then
            $\mathcal{P} \leftarrow \mathcal{P} \setminus P_j$
    $\{\theta_k\} \leftarrow$ epLCA
    $\alpha$, $\{x_{ij}\} \leftarrow$ SOCP($\mathcal{L}, \mathcal{B}, \mathcal{P}, \{\theta_k\}$)
    return $\alpha$, $\{x_{ij}\}$
```

TABLE II: Notation Used in Section V.

Symbol	Definition
Input parameters	
$\mathcal{N}$	Set of physical nodes in the substrate network
$\mathcal{V}$	Set of virtual nodes from all virtual networks
v_i	Virtual node i ($v_i \in \mathcal{V}$)
n_k	Physical node k ($n_k \in \mathcal{N}$)
R_k	Compute capacity of physical node n_k
Δ_i	Compute demand of virtual node v_i
η	Acceptable congestion on each physical node
Decision variables	
$z_{ik} \in \{0,1\}$	Virtual node v_i is placed on physical node n_k

V. Virtual Network Embedding

In this section, we discuss some possibilities for how our VLE formulation may be incorporated in a virtual node embedding algorithm to construct a complete VNE solution. First, we demonstrate how our approach can be used to model node embedding under demand uncertainty and then discuss integration with existing VNE algorithms. The notation used in this section is summarized in Table II.

A. Node Embedding under Uncertain Demands

Virtual node demands, *e.g.*, processing demands, could also be uncertain similar to virtual link demands. However, node resources are allocated on a per-node basis, *e.g.*, a virtual node is placed on a single physical node. As such, there is no complication as in virtual link embedding due to end-to-end requirements. To this end, the following formulation can be used to model node demands under uncertainty. Let $\mathcal{N}$ and $\mathcal{V}$ denote the set of physical and virtual nodes, respectively. Also, let Δ_i denote the uncertain node (processing) demand of virtual node $v_i \in \mathcal{V}$. We assume that Δ_i is a random variable with mean $\mathbb{E}[\Delta_i]$ and variance $\mathrm{Var}[\Delta_i]$ (recall that an upper bound on the variance suffices). Define the binary variable z_{ik} to be 1 if virtual node v_i is placed on substrate node n_k. The following inequality expresses the capacity constraints in virtual node embedding,

$$\mathbb{P}\Big\{ \sum_{v_i \in \mathcal{V}} \Delta_i z_{ik} > R_k \Big\} \leq \eta, \tag{21}$$

where $0 < \eta < 1$ is the acceptable bound on the probability of resource violation on each node and R_k denotes the capacity of node n_k. Using the inequality (5), the equivalent robust counterpart of the above constraint is given by the following conic constraint,

$$\sum_{v_i \in \mathcal{V}} \mathrm{Var}[\Delta_i]\, z_{ik}^2 \leq \Big(\frac{R_k - \sum_{v_i \in \mathcal{V}} \mathbb{E}[\Delta_i]\, z_{ik}}{\sqrt{2\ln(1/\eta)}} \Big)^2. \tag{22}$$

B. Integration with VNE Algorithms

As mentioned earlier, numerous algorithms are proposed in the literature for the VNE problem (most consider the deterministic variant, where resource demands are known exactly). These algorithms can be broadly divided into two categories with respect to how node and link embedding decisions are made [2], as follows.

I. Uncoordinated. Algorithms in this category solve the link and node embedding problems separately. An example is [19], in which node embedding is solved first based on the availability of resources at substrate nodes independent of the virtual links. Clearly, it is straightforward to use our VLE algorithm in the second phase of these algorithms. One complication is the uncertainty in node demands, which can be addressed using the formulation (22), which results in a second-order cone program.

II. Coordinated. Algorithms in this category are further divided to:

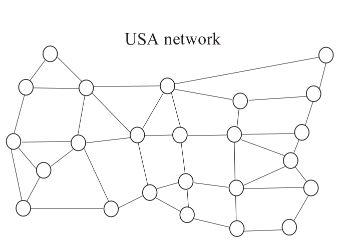
- *Two-stage coordinated:* These algorithms perform node and link embedding sequentially as in uncoordinated algorithms but during node embedding consider link requirements too. An example is [8], in which node embedding is solved by considering both node and link requirements. Once the virtual nodes are mapped, it applies a link embedding algorithm similar to the one proposed in [19]. As such, our VLE algorithm can still be used in the second stage of these algorithms. Of course, modifications needed to be made to consider demand uncertainty in the node embedding stage. Moreover, the resulting optimization problems will be second order instead of linear as in [19].
- *One-stage coordinated:* These algorithms perform node and link embedding jointly. An example is [7], which uses the PageRank algorithm to rank physical nodes for embedding virtual nodes. Most are based on heuristics or solving the exact problem formulation. As such, incorporating our VLE algorithm in these works requires substantial modifications and may not even be feasible. One idea is to apply these algorithms to compute an embedding and then try to modify it to accommodate uncertain demands and congestion requirements heuristically.

VI. Performance Evaluation

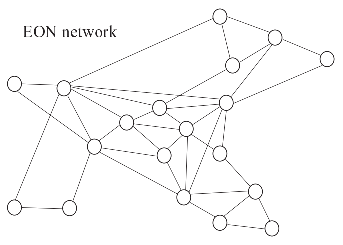
We evaluate the performance of epVLE via simulations and SDN testbed experiments. We start in subsection VI-B by comparing epVLE with the exact VLE model described in Problem 1 using simulations in small-scale network topologies (solving the exact model for large networks is not feasible). Then, in subsection VI-C, we compare epVLE with an existing robust link embedding algorithm that considers congestion on physical links individually [17]. In subsection VI-D, large-scale evaluations on randomly generated network topologies are considered to show the scalability of epVLE. Finally, measurement results based on model-derive as well as trace-driven experiments in a testbed are presented in subsection VI-E.

A. Simulation Setup

Network Topology. For small-scale simulations, the USA (24 nodes and 43 links) and European Optical Network (EON)



(a) USA network topology.



(b) EON network topology.

Fig. 3: Small-scale network topologies.

(19 nodes and 37 links) topologies are used (see Fig. 3). These topologies are widely used in the literature for similar evaluation purposes (*e.g.*, see [17]).

Virtual Networks. For each network topology, a random set of VNs is generated and embedded in the physical network. Generally, the optimization programs take as input the set of virtual network requests that need to be embedded in the physical network. Moreover, virtual nodes can be embedded by any desired algorithm before the link mapping stage. In the simulations, we assume that a virtual network consists of a set of origin-destination pairs connected with virtual links. Each virtual link is assigned to a randomly chosen origin-destination pair in the physical network. Every node in the graph of the physical network can potentially be chosen as an origin or destination for some virtual link.

Exact Solution. To have a baseline for comparison, we also show the results obtained by solving the exact VLE model in Problem 1. Optimization solver Knitro [40] in conjunction with the AMPL [44] modelling language is used to solve the exact VLE model. The epVLE algorithm is a SOCP, which is solved using Gurobi [42] version 7.5.2. Simulations are carried out on a basic desktop machine with an Intel(R) Core(TM) i7-4770 CPU@3.40 GHz with 16 GB RAM.

Simulation Parameters. For the sake of simplicity, we assume uniform bandwidth demands (with mean μ) for all virtual links. Physical link capacities are also assumed to be equal and given by C, where C is scaled with respect to the mean bandwidth demand μ (*i.e.*, $C = \rho$ means that the link capacity is $\rho\mu$ in bps). In our simulations, C varies between 10 and 100, but is set to 20 by default. We note that using unscaled values for link capacities (*e.g.*, 1 Gbps) actually slows down solving the quadratic optimization model considerably, since it leads to a large coefficient range and numerical issues. We also set the variance of bandwidth demands to σ^2 for all virtual links. In this section, we use the Coefficient of Variation, denoted by CoV and defined as CoV $= \sigma/\mu$, to describe the burstiness of bandwidth demands. A high CoV indicates high uncertainty in bandwidth demands, and vice versa.

Performance Measures. We use the following measures to compare the performance of different approaches: 1) the number of admitted virtual links, 2) the achieved congestion probability, and 3) the utilization of the most congested link (denoted by α). To compute the number of admitted virtual links, we solve the problem starting with a small set of virtual link requests. Then, we iteratively increase the number of virtual link requests until the problem becomes infeasible.

TABLE III: Default simulation parameters.

Parameter	Value
C	20
μ	1
K	3
CoV	1.0
ϵ	0.1
Network Topology	USA

While this linear search can be improved, *e.g.*, by a binary search, it takes only seconds to find the maximum number of admitted links for epVLE. For the exact model, however, it is a time-consuming process. Therefore, we have used starting points based on the solutions produced by epVLE to speedup the search process. That is, to compute the number of admitted virtual links under the exact model, we start by solving the model for the number of links that were admitted under the approximate model epVLE. We then linearly search around this number to find the maximum number of links that can be admitted by solving the exact model.

Methodology. In each network scenario, a K-shortest-path algorithm [45] is run to compute K candidate paths between the chosen origin-destination nodes for each virtual link. Next, the optimization models are solved based on the computed candidate paths and network parameters. The default values for the parameters are presented in Table III. The value of a parameter changes only when its impact is investigated. Each point in the plots is the average of 4 simulation runs. The error bars (showing the min and max values) are not presented in cases where the deviation from the average was very small.

B. Comparison with the Exact Model

In this subsection, we compare the performance of epVLE with that of the exact model in terms of the average number of admitted virtual links and achieved congestion probability.

Number of Admitted Links. Fig. 4 shows the the number of admitted virtual links by each model for different coefficients of variation (CoV = $0, 0.5, 1, 1.5$). By increasing CoV, demand uncertainty increases and more bandwidth is reserved per virtual link, which causes a sharp decline in the number of admitted virtual link requests. We observe that the results achieved by epVLE and exact model are very close to each other. The reason for slightly higher number of admitted links under epVLE can be explained by looking at Fig 5. We can see that epVLE generally achieves higher congestion probabilities, which translates to more admitted virtual link requests.

Congestion Probability. Fig. 5 depicts the achieved end-to-end congestion probabilities for epVLE and the exact model. For both models, the objective is to satisfy a maximum congestion probability of $\epsilon = 0.1$. We observe that: 1) both models generally satisfy the target congestion probability, and 2) the exact model generally achieves lower congestion probabilities compared to epVLE. With both models, the achieved congestion probabilities are below the required target probability. The reason is that, the global solver used to solve the exact model may only find a local optimum, which satisfies the target requirement but is far from the global optimal. Also, as presented in Algorithm 1, if a shorter path is fully contained in

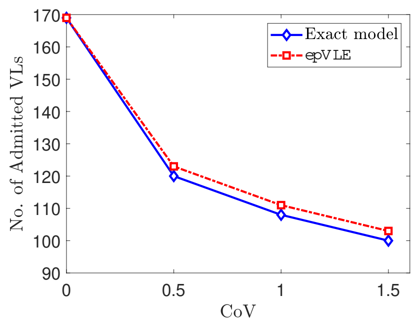
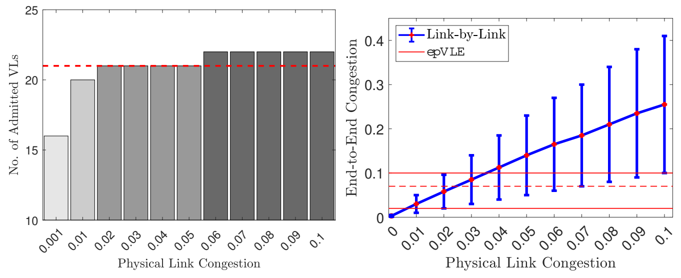


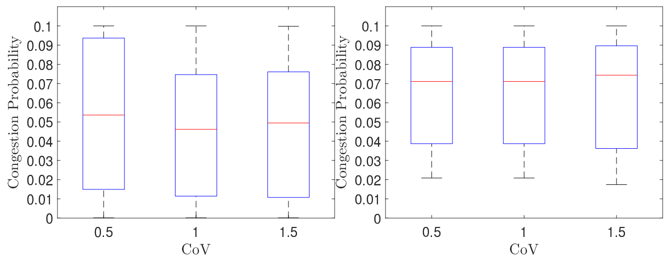
Fig. 4: Average number of admitted virtual links with epVLE and the exact model. As the variance of bandwidth demands increases, the number of admitted virtual links decreases. In all cases, the solution produced by epVLE is very close to the exact solution.



(a) Number of admitted virtual links. (b) End-to-end congestion probability.

Fig. 6: Comparison of epVLE and the robust link-by-link model [17]. The dashed red line shows epVLE results. The link congestion achieved by epVLE is always below 0.1, as required. In contrast, the link congestion achieved by the link-by-link approach either underachieves or overshoots the required 0.1 level.

a longer path, then the achieved congestion probability for the shorter path will be smaller than the required target. Finally, the fluctuation in achieved congestion probability under both models is attributed to random selection of origin-destination pairs in each simulation run.



(a) Exact model. (b) epVLE.

Fig. 5: Congestion probability with epVLE and the exact model. Regardless of the variance of bandwidth demands, both algorithms are able to satisfy the desired end-to-end congestion probability.

C. Comparison with the Link-by-Link Model

In this subsection, the algorithm epVLE is compared with the robust VLE algorithm proposed in [17]. The VLE problem in [17] is formulated as a robust optimization problem, where the placement of virtual nodes is assumed to be known. The objective is to provide a pre-specified congestion probability guarantee on each physical link, which is achieved by modeling demand uncertainty using ellipsoidal uncertainty sets [23]. This algorithm is chosen as a representative of *link-by-link* approaches that are able to deal with demand uncertainty using robust optimization. The main challenge in link-by-link approaches is how to properly set the physical link congestion requirement. Recall that we do not need to deal with this problem in our approach as in our formulation the appropriate congestion requirement for each physical link is determined by the algorithm itself as part of the solution. To show the effect of physical link congestion on the performance of the link embedding algorithms, in this experiment, the physical link congestion probability varies between 0.001 and 0.1. The congestion probability 0.001 (*i.e.*, 0.1% congestion) is selected to represent no congestion, since the congestion probability cannot be set to 0% in the model due to the term $\ln(\frac{1}{\epsilon_k})$ in the optimization program. In the case of the link-by-link model, there is no guarantee that end-to-end congestion probability remains less than the specified target (*i.e.*, $\epsilon = 10\%$) for any of these values, since it depends entirely on the number of physical links in substrate paths.

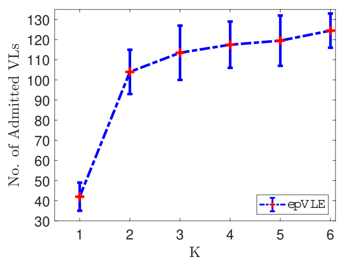
The results are presented in Fig. 6, which shows the number of admitted virtual links and the virtual link congestion probability for the link-by-link approach in comparison with epVLE (indicated by the red line). We observe that as the physical link congestion probability is allowed to increase, slightly more virtual links are admitted with the link-by-link approach. However, this comes at the cost of violating the congestion requirement of virtual links, as presented in Fig. 6(b). In particular, it can be seen that as soon as the physical congestion probability is set to more than 0.04, the end-to-end congestion probability achieved under the link-by-link approach shoots up and violates the required 10%. In contrast, epVLE always satisfies the required congestion regardless of the specific physical link congestion. We note that the error bars in this figure show the min and max values observed across all simulation runs.

D. Scalability Analysis

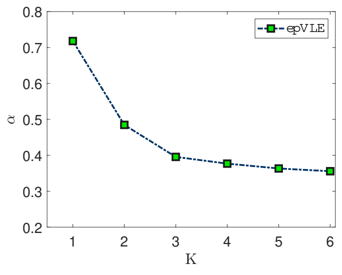
In this section, we focus on epVLE, as it is computationally fast and is reasonably accurate compared to the exact model.

Number of Candidate Paths. Fig. 7 shows the effect of increasing the number of candidate paths K on the performance of epVLE. As can be seen in Fig. 7(a), the maximum number of admitted links increases drastically by increasing the number of candidate paths from 1 to 3. However, the gain diminishes as the number of paths increases beyond 3. Therefore, we set the default number of candidates paths to 3 in the rest of the simulations in this section. Fig. 7(b) illustrates the impact of K on the most congested link, *i.e.*, the link with the highest utilization α. In this figure, the number of virtual links is fixed at $|\mathcal{L}| = 30$. As expected, the highest link utilization drops as we increase the number of candidate paths. Again, there is no considerable reduction after $K = 3$.

Running Time. In this experiment, the Barabasi-Albert model [46] is used to generate random scale-free networks with up to 2000 nodes. The number of virtual links given to the model as input varies from 100 to 1000 virtual links. The results are averaged over 100 runs, where in each run epVLE is



(a) Number of admitted virtual links.



(b) Utilization of the most congested link.

Fig. 7: Effect of the number of paths (K).

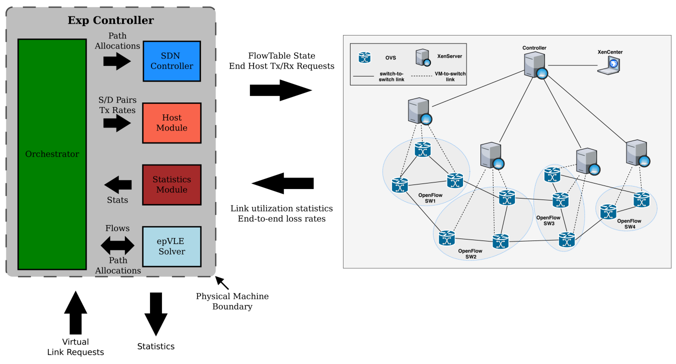


Fig. 9: High level system architecture.

used to compute a link embedding. The resulting embedding may be feasible or infeasible depending on the computed value for the maximum link utilization α. For example, when embedding 1000 flows in a network with only 100 nodes, most computed embeddings are not feasible, but epVLE has to solve the problem anyways. It is observed that, as expected from the analysis presented in Section IV, the running time is dominated by the number of virtual links to be embedded, but in all cases it remains in the order of seconds.

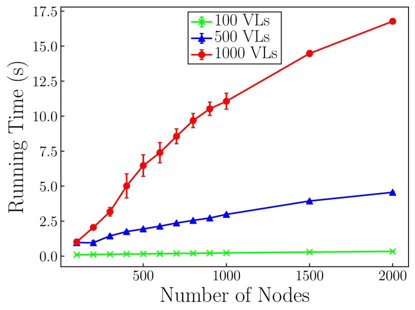


Fig. 8: Running time of epVLE. The running time is dominated by the number of virtual links, but is also affected by the size of the network. In all these cases, the running time is in the order of seconds.

E. Testbed Setup

In this subsection, we describe the Software Defined Networking (SDN) testbed we have built to conduct our experiments. Then, in the subsequent subsections, we present the measurement results that we have obtained from the testbed.

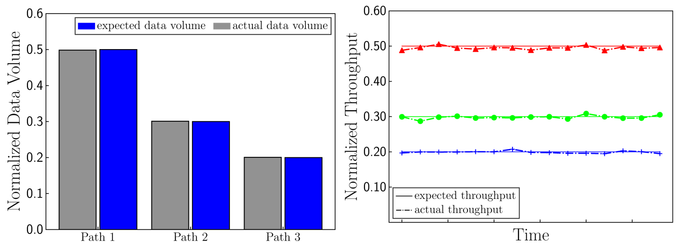
Physical Topology. We used four Aruba 2930F JL259A OpenFlow switches to construct the substrate network into which virtual links are embedded. Each of the physical switches supports OpenFlow version 1.3 and can host up to 16 distinct OpenFlow instances. From the perspective of the SDN controller, each OpenFlow instance appears as a distinct switch in the substrate network. Each of the OpenFlow instances hosted by a particular Aruba switch is assigned a subset of the physical ports present on the switch. This scheme in which multiple OpenFlow instances are co-located in the same physical switch allows for the construction of diverse network topologies using relatively small amounts of physical switching hardware. Specifically, for this set of experiments we configure the testbed to emulate the Abilene topology shown in Fig. 9. All Ethernet interfaces in the testbed are 1 Gbps interfaces. Our SDN testbed also includes five DELL PowerEdge R330 servers equipped with Intel Xeon E3-1240 processors, 16 GB of RAM and 6 Ethernet ports, each of which can forward packets at a rate of 1 Gbps. Each server hosts one or more virtual machines (VMs). A single server is dedicated to hosting the SDN controller as well as the software that implements epVLE. The remaining four servers host VMs that act as sources and sinks for network traffic. In total we provision 11 of these traffic generator VMs, each of which is connected to a distinct switch in the substrate network.

System Architecture. In order to conduct our experiments, we have implemented a generic orchestration framework that will configure both the forwarding plane and the end hosts in response to virtual link requests submitted by users. Fig. 9 provides an overview of our orchestration framework. Virtual link requests are submitted to the orchestrator process in batches either via a REST endpoint exposed by the experiment controller or via a Python API. Upon receiving a new batch of virtual link requests the orchestrator process does two things:

1) Invoke the epVLE solver module to compute the path allocations for the batch of virtual link requests.
2) Configure the end hosts to generate traffic in accordance with the rates specified in the batch of virtual link requests. The orchestrator process carries out this task by invoking functionality provided by the host module.

After receiving path allocations from the epVLE solver, the orchestrator process instructs the SDN controller to update the flow table state of the switches in the network according to the path allocations generated by the epVLE solver. Once the forwarding plane state updates are complete, the orchestrator signals the host module to begin traffic generation.

Virtual Network Generation. Virtual networks are generated randomly such that the number of virtual nodes and links for a VN are uniformly chosen from the range $[2, 3, 4]$. Then the Barabasi-Albert model [46] is used to generate the corresponding virtual network topology. The next step is to assign CPU and uncertain bandwidth demands to virtual nodes and links respectively. Since our work focuses on link embedding, CPU demands are assumed to be insignificant compared to actual processing power capacity of servers so that they do not impose any capacity restrictions during the node embedding process. To embed each virtual node, a physical node is selected uniformly randomly from the set of substrate nodes. On the other hand, the uncertain bandwidth demand for each virtual link ℓ is defined by its mean and standard deviation,



(a) Normalized data volume per path. (b) Normalized throughput per path.

Fig. 10: Flow splitting in SDN testbed.

i.e., (μ_ℓ, σ_ℓ). We consider different sets of experiments with varying demand characteristics, as described later.

Multipath Routing Implementation. In accordance with the model outlined in section III, each virtual link (corresponding to a flow) is mapped to K physical paths that traverse the substrate network. Typically the policies of traditional routing protocols lack the expressiveness required to implement this type of routing since a packet's path through the network is entirely determined by its layer three destination address. As such we chose to implement the multipath routing scheme using relatively inexpensive OpenFlow switches. Since the OpenFlow agent implementation present in our switching hardware mandates that the mapping from header space to OpenFlow actions must be deterministic (*i.e.*, the same header must always yield the same action under a given flowtable configuration) each packet header of a particular flow must be augmented with additional information indicating which of the K physical paths through the substrate network the packet should transit. For this set of experiments, we set the value of the *Differentiated Services Codepoint* (DSCP) field in the IP header to indicate which of the K physical paths the packet should transit. While this solution sufficed to allow us to conduct our experiments, we note that the use of the DSCP field in this unorthodox fashion could interfere with preexisting QoS policies in real network deployments. However we do not see this as a major issue since any one of the commonly available traffic tunneling methods could be used to implement the multipath routing scheme described without the need to infer the route a packet should take through the substrate network from its DSCP value.

Flow Splitting. Fig. 10 demonstrates the granularity of flow splitting that we were able to achieve using the multipath routing scheme described above. Fig. 10(a) shows the expected and actual data volumes per path for a flow split over three distinct paths through the substrate network. We note that the maximum deviation of the actual data volume from the expected data volume was less than 2%. Similarly, Fig. 10(b) shows the expected and actual throughputs over time on each of the three constituent paths in the flow. Again, se see that the actual throughput on each of the paths exhibits very small deviation from the expected value.

Traffic Generation. Because it was necessary to implement the DSCP tagging scheme described previously, we were not able to utilize any existing traffic generation tools. As such, we implemented a basic traffic generation application that allows packets to be generated and transmitted at rates sampled from a number of common statistical distributions. Our traffic generation application is also able to adjust virtual link transmission rates in accordance with values derived from traffic traces. In addition, the traffic generation application is also responsible for tagging generated packets according to the path splitting ratios, *i.e.*, x_{ij} variables. The traffic generator is implemented in Python using the Python sockets API.

Measurements and Statistics Collection. In order to evaluate the performance of our VLE solution in the physical testbed we collect a number of metrics, specifically we tabulate:

1) Utilization of each link in the substrate network over time.
2) End to end packet loss rates for each of the virtual links embedded in the substrate network.

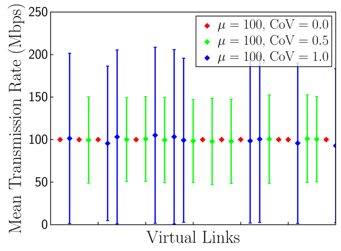
In order to obtain the data necessary to compute the utilization of each link in the substrate network, the statistics module periodically instructs the SDN controller to query the values of the packet and byte counters for each of the ports on each of the switches that constitutes the substrate network. The statistics module then computes utilization for each link based on the values of the counters retrieved from the SDN controller and can report utilization statistics to users via the orchestrator process. The statistics module computes end to end packet loss rates by querying the host module to obtain transmitted and received packet counters for each of the virtual links embedded in the substrate network. Using the values of these counters the statistics module can report end to end packet loss rates for each virtual link to users via the orchestrator process. We note that congestion events are the primary cause of packet drops in our experiments. However, during each congestion event, *multiple packets* could be dropped. Thus, while there is a strong correlation between packet drop probability and congestion probability, their values do not necessarily match.

Implemented Algorithms. In addition to epVLE, we have implemented the following VLE algorithms as well:

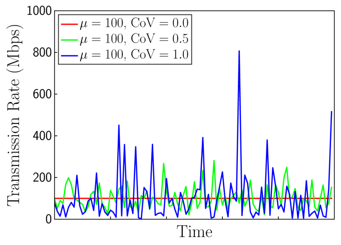
- Average: This algorithm ignores demand variability and bases its link mapping decisions on the mean bandwidth demand only. Specifically, it allocates μ bps bandwidth to each virtual link. This algorithm can be considered as a *deterministic* VLE algorithm.
- 95-Percentile: This algorithm actually considers demand variability when mapping virtual links. Specifically, it assumes that the bandwidth demands follow a Normal distribution with mean μ and variance σ^2 (which are assumed to be known). It then computes an effective demand for each virtual link, which is equal to the 95-percentile of the bandwidth demand given by $\mu + 1.65\sigma$. We chose the 95-percentile so that if a virtual link is mapped to a path of length 2, then the end-to-end congestion probability of the link satisfies the requirement $\epsilon = 0.1$. This algorithm can be considered as a *link-by-link* robust VLE algorithm.

Methodology. We compare the performance of algorithms epVLE, Average and 95-Percentile using model-driven and trace-driven experiments, as described below:

- *Model-Driven Testbed Experiments:* In these experiments, traffic demands of virtual links are generated randomly



(a) Mean transmission rates.



(b) Instantaneous transmission rates.

Fig. 11: Properties of model driven virtual links.

based on a distribution model. Specifically, we generate demands that follow a Gamma distribution (which is defined over $\mathbb{R}^+$) with given mean and variance.
- *Trace-Driven Testbed Experiments:* In these experiments, traffic demands of virtual links are generated based on real traffic traces collected from an ISP backbone link. The traffic traces have packet-level information that can be used to derive the traffic rate for each virtual link.

The results of these experiments are presented in the next subsections. The duration of each experiment is 15 minutes. In these experiments, candidate paths for each virtual link are the $K = 3$ shortest paths between origin and destination nodes of each virtual link.

F. Model-Driven Testbed Experiments

In these experiments, we use the Gamma distribution to generate virtual link demands based on a given mean and variance. Fig. 11 illustrates some basic statistical properties of the virtual links that were generated during the model-driven experiments. More precisely, Fig. 11(a) shows the mean and standard deviation of the transmission rates for a selection of model-driven virtual links under various model parameters. Fig. 11(b) shows the instantaneous transmission rates of a selection of model-driven virtual links under various model parameters. During each experiment, the transmission rate of each virtual link is changed every 10 seconds. That is, every 10 seconds, we generate a new traffic rate for each virtual link using the corresponding Gamma distribution and reconfigure our traffic generation application with the new rates.

We consider two different scenarios for this set of experiments:

- **Homogeneous Scenario:** In this scenario, the mean and variance of demands for all virtual links are equal. That is, for every virtual link, we set $(\mu = 100, \sigma = 100)$ Mbps. In this scenario, origin-destination nodes of virtual links are chosen randomly (*i.e.*, random embedding).
- **Heterogeneous Scenario:** In this scenario, demands of different virtual links have different mean and variance. Specifically, the mean demand is randomly chosen from $\{100, 200, 300\}$ Mbps, while CoV $(= \sigma/\mu)$ is chosen randomly from the set $\{0, 0.5, 1\}$. In this scenario, origin-destination nodes of virtual links are chosen by the TK-Match algorithm [47].

Results and Discussion. The results for homogeneous and heterogeneous scenarios are presented in Figs. 12 and 13. The main observations are: 1) epVLE is more aggressive compared to 95-Percentile, which achieves the lowest utilization, and 2) epVLE is more conservative than Average, which admits the most number of links at the cost of overshooting the packet drop probability. Next, with respect to the packet drop probability, we see that in the homogeneous scenario, both epVLE and 95-Percentile are able to bound the packet drop probability below 10%. However, in the heterogeneous scenario, both algorithms result in higher packet drop probabilities. As discussed earlier, this does not necessarily mean that the achieved congestion probabilities are higher than 10%. Recall that during a single congestion event, multiple packets could be dropped. As such, in general the packet drop probability provides an upper bound on the actual congestion probability.

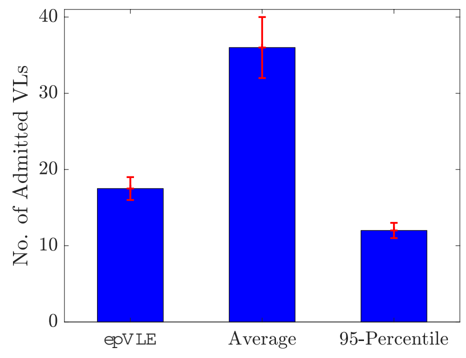
We can also see the effect of increased traffic burstiness on the performance of the algorithms. Recall that there is more traffic burstiness in the heterogeneous scenario. As a result, we see that the packet drop probability for all algorithms has increased compared to the first experiment. Fig. 13(a) illustrates the number of admitted VN requests and their corresponding virtual links for all three algorithms. Interestingly, we observe that, compared to the homogeneous scenario, more virtual links are embedded in the network on average. This is due to the fact that some virtual link demands actually have less variability compared to the virtual link demands in the homogeneous scenario (when CoV is 0 or 0.5).

Another interesting behavior is observed in the link utilization plots under 95-Percentile and epVLE. While the difference between the number of virtual links admitted by each of these algorithms is negligible, the difference between their link utilizations is somewhat more substantial. Specifically, the link utilization under epVLE is higher than the link utilization under 95-Percentile. The reason is that the order and type of VNs becomes important as we consider heterogeneous VNs in this experiment. This means that a different set of VNs is admitted under each algorithm. Therefore, we can have situations where the 95-Percentile algorithm rejects some large VNs with high bandwidth demand, while epVLE admits some of the large VNs. As a result, while both algorithms admit roughly the same number of VNs, the link utilization under epVLE is higher.

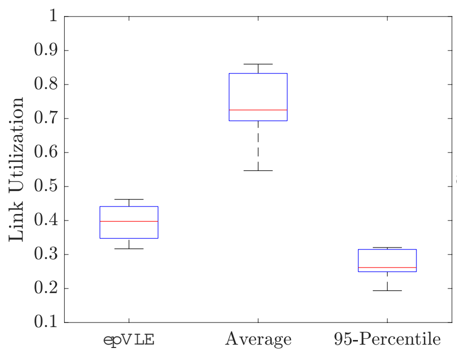
G. Trace-Driven Testbed Experiments

In these experiments, we use real traffic traces collected from an ISP backbone link [48] to generate traffic demands for virtual links.

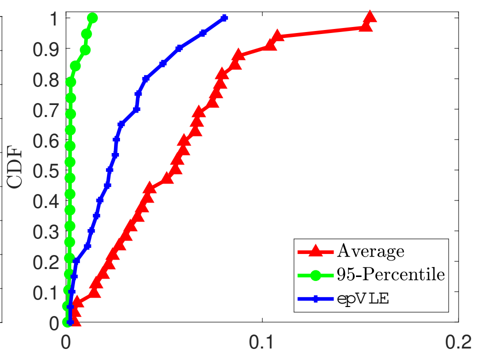
Traffic Traces. The traffic traces contain packet-level information about traffic traversing the ISP link during a 60 minute period. For the entire trace, the mean and standard deviation of the throughput on the link are 89.05 Mpbs and 7.74 Mpbs, respectively. To generate virtual link demands, we first divide the trace into segments such that each segment represents traffic flow on the ISP link during a 1 minute period. Each of these segments is mapped to a virtual link. Transmission rates for a virtual link are sampled from its corresponding traffic trace segment. Recall that the rate of each virtual link is re-sampled every 10 seconds during the experiments. Fig. 14



(a) Average number of admitted virtual links.

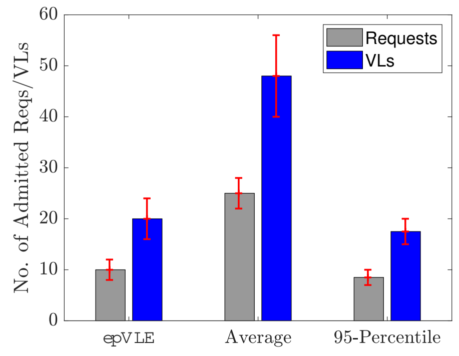


(b) Average utilization of physical links.

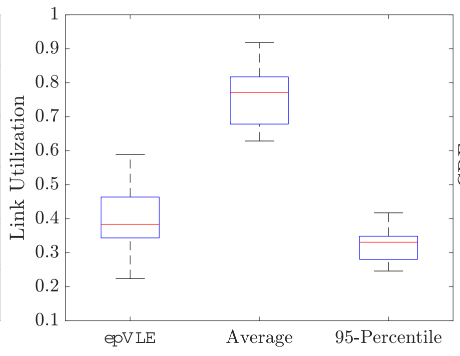


(c) CDF of packet drop probability.

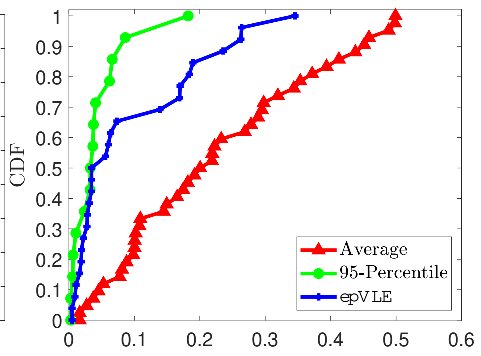
Fig. 12: Homogeneous model-driven testbed experiments.



(a) Average number of admitted virtual links.



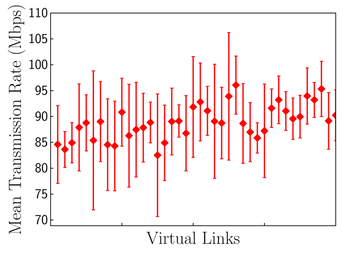
(b) Average utilization of physical links.



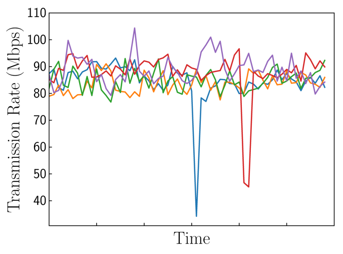
(c) CDF of packet drop probability.

Fig. 13: Heterogeneous model-driven testbed experiments.

shows some basic properties of the virtual link transmission rates that were derived from these traces. Specifically Fig 14(a) shows the mean and standard deviation of transmission rates of a collection of virtual links and Fig. 14(b) shows the transmission rates of a selection of virtual links over time.



(a) Mean transmission rates.



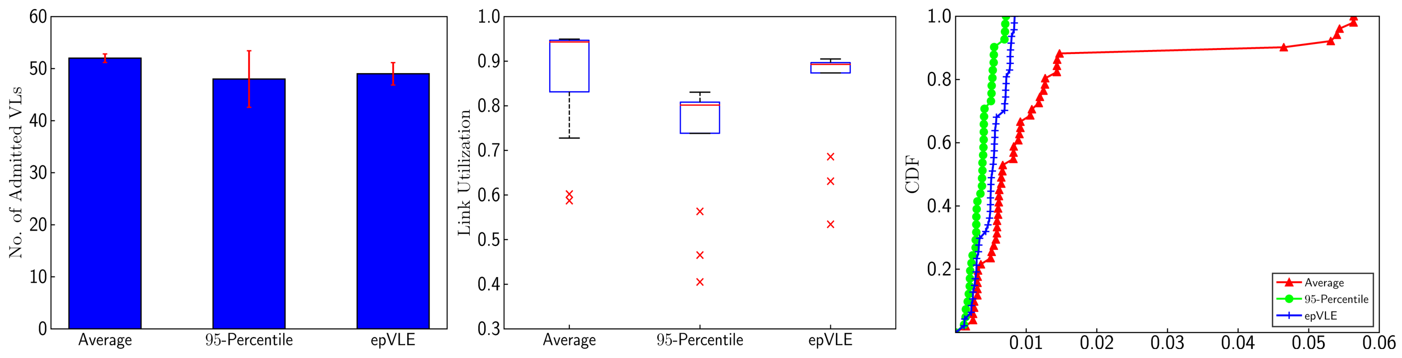
(b) Instantaneous transmission rates.

Fig. 14: Properties of trace driven virtual links.

Results and Discussion. The results for this set of experiments are shown in Fig. 15. The main observations are: 1) the relative performance of the three algorithms is similar to that seen in the previous model-driven scenarios, 2) since the trace-based demands have less variability compared to the model-driven experiments, even epVLE and 95-Percentile are able to achieve significantly higher link utilization and admit more virtual links, and 3) epVLE is able to bound the packet drop probability to 10%. We note that in Fig. 15(a), the number of admitted virtual links in this scenario was considerably higher than in the previous experiments. This is a result of both the diminished variability and the reduced mean transmission rates of the trace-driven virtual links when compared with those of the model-driven virtual links. Similarly, Fig. 15(b) demonstrates that, as a result of the diminished variability in the transmission rates, algorithm epVLE as well as the other algorithms, are able to pack virtual links into the substrate network in a denser fashion. Finally, the CDF of the packet drop probability for this scenario is shown in Fig. 15(c). The observed packet loss probabilities are significantly lower in this scenario when compared with those observed in the other scenarios, which is again explained by smoother traffic rates in this experiment.

VII. Conclusion

In this paper, we considered the problem of probabilistic virtual link embedding with uncertain demands. We formulated the problem as an optimization problem, where link demands are described by random variables for which only the mean and variance are known. We showed that the problem can be well approximated by a second order cone program, which can be solved efficiently even for large networks. Using a combination of simulations and testbed measurements, we then studied the performance and scalability of our approximate algorithm in a variety of network scenarios. Our results show that the approximate algorithm achieves near-optimal performance, while easily scaling to large-scale problem instances. They also confirm that our algorithm is able to satisfy a required congestion probability fo each virtual link in realistic network scenarios. An interesting extension of this work is to consider an online version of the problem in which virtual network requests arrive one-by-one over time.



(a) Average number of admitted virtual links (b) Average utilization of physical links. (c) CDF of packet drop probability.

Fig. 15: Trace-driven testbed experiments.

References

[1] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *Proc. IFIP Networking*, May 2018.

[2] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, 2013.

[3] H. Cao *et al.*, "Exact solutions of VNE: A survey," *China Communications*, vol. 13, no. 6, 2016.

[4] J. Inführ and G. R. Raidl, "Introducing the virtual network mapping problem with delay, routing and location constraints," in *Proc. INOC*, Jun. 2011.

[5] J. Botero *et al.*, "Energy efficient virtual network embedding," *Comput. Netw.*, vol. 16, no. 5, 2012.

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

[7] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM CCR*, vol. 41, no. 2, 2011.

[8] N. M. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE INFOCOM*, Apr. 2009.

[9] ——, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, 2012.

[10] V. N. Nikhil Bansal, Kang-Won Lee and M. Zafer, "Minimum congestion mapping in a cloud," *SIAM J. Comput.*, vol. 44, no. 3, 2015.

[11] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," in *Proc. IFIP Networking*, May 2018.

[12] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE INFOCOM*, Apr. 2011.

[13] L. Yu and H. Shen, "Bandwidth guarantee under demand uncertainty in multi-tenant clouds," in *Proc. IEEE ICDCS*, Jun. 2014.

[14] G. Sun *et al.*, "Exploring online virtual networks mapping with stochastic bandwidth demand in multi-datacenter," *Photon. Netw. Commun.*, vol. 23, no. 2, 2012.

[15] Z. Zhang *et al.*, "Energy aware virtual network embedding with dynamic demands: Online and offline," *Computer Networks*, vol. 93, 2015.

[16] C. Qiu, H. Shen, and L. Chen, "Probabilistic demand allocation for cloud service brokerage," in *Proc. IEEE INFOCOM*, Apr. 2016.

[17] S. S. W. Lee *et al.*, "Design of bandwidth guaranteed OpenFlow virtual networks using robust optimization," in *Proc. IEEE GLOBECOM*, 2014.

[18] S. Coniglio, A. Koster, and M. Tieves, "Data uncertainty in virtual network embedding: Robust optimization and protection levels," *Network and Systems Management*, vol. 24, no. 3, 2016.

[19] J. R. Minlan Yu, Yung Yi and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM CCR*, vol. 38, no. 2, 2008.

[20] O. Heckmann, J. Schmitt, and R. Steinmetz, "Robust bandwidth allocation strategies," in *IEEE IWQoS*, May 2002.

[21] C. Fuerst, M. Rost, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," *ACM SIGCOMM CCR*, Jul. 2015.

[22] L. S. Carlo Fuerst, Stefan Schmid and P. Costa, "Kraken: Online and elastic resource reservations for cloud datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, 2018.

[23] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009.

[24] M. S. Dimitris Bertsimas, "The price of robustness," *Operations Research*, vol. 52, no. 1, 2004.

[25] G. S. Guy Evena, Moti Medinaa and S. Schmid, "Competitive and deterministic embeddings of virtual networks," *Theoretical Computer Science*, vol. 496, 2013.

[26] P. v. d. S. Andreas Blenk, Patrick Kalmbach and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *Proc. IEEE CNSM*, Nov. 2016.

[27] M. G. Mahdi Dolati, Bahareh Hassanpour and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM, Workshop on Network Intelligence*, Apr. 2019.

[28] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, 2016.

[29] D. Chemodanov, P. Calyam, and F. Esposito, "A near optimal reliable composition approach for geo-distributed latency-sensitive service chains," in *Proc. IEEE INFOCOM*, Apr. 2019.

[30] M. Nguyen, M. Dolati, and M. Ghaderi, "Proactive service orchestration with deadline," in *Proc. IEEE NetSoft*, Jun. 2019.

[31] A. Marotta *et al.*, "A fast robust optimization-based heuristic for the deployment of green virtual network functions," *Network and Computer Applications*, vol. 95, no. 1, 2017.

[32] ——, "On the energy cost of robustness for green virtual network function placement in 5G virtualized infrastructures," *Computer Networks*, vol. 125, 2017.

[33] S. S. Srikanth Kandula, Dina Katabi and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM CCR*, vol. 37, no. 2, 2007.

[34] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, Aug. 2013.

[35] N. Katta *et al.*, "HULA: Scalable load balancing using programmable data planes," in *Proc. ACM SOSR*, Mar. 2016.

[36] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proc. IEEE ICNP*, Nov. 2003.

[37] M. Roughanx *et al.*, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *Proc. ACM IMC*, Oct. 2004.

[38] M. Wainwright, "Basic tail and concentration bounds," https://www.stat.berkeley.edu/~mjwain/stat210b/Chap2_TailBounds_Jan22_2015.pdf, 2015, online; accessed 28 May 2018.

[39] H. U. Vahab S. Mirrokni, Marina Thanan and S. Paul, "A simple polynomial time framework for reduced-path decomposition in multipath routing," in *Proc. IEEE INFOCOM*, Mar. 2004.

[40] Artelys Knitro, https://www.artelys.com/en/optimization-tools/knitro.

[41] F. Alizadeh and D. Goldfarb, "Second-order cone programming," *Math. Program.*, vol. 95, no. 1, 2001.

[42] Gurobi Optimization. [Online]. Available: http://www.gurobi.com/

[43] R. Nagarajan, J. kurose, and D. Towsley, "Local allocation of end-to-end quality-of-service in high-speed networks," in *Proc IFIP Workshop on Performance Evaluation of Communication Systems*, Jan. 1993.

[44] AMPL. [Online]. Available: https://ampl.com/

[45] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, 1971.

[46] A. Barabas and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, 1999.

[47] X. Li *et al.*, "Resource allocation with multi-factor node ranking in data center networks," *Future Generation Comput. Sys.*, vol. 32, no. 4, 2014.

[48] The CAIDA Anonymized Internet Traces Dataset, https:www.caida.org/data/passive/passive_trace_statistics.xml.