Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

PolyViNE: policy-based virtual network embedding across multiple domains

Fady Samuel[1], Mosharaf Chowdhury[2] and Raouf Boutaba[3,4*]

Abstract

Intra-domain virtual network embedding is a well-studied problem in the network virtualization literature. For most practical purposes, however, virtual networks (VNs) must be provisioned across heterogeneous administrative domains managed by multiple infrastructure providers (InPs).
In this paper, we present PolyViNE, a policy-based inter-domain VN embedding framework that embeds end-to-end VNs in a decentralized manner. PolyViNE introduces a distributed protocol that coordinates the VN embedding process across participating InPs and ensures competitive prices for service providers (SPs), i.e., VN owners, while providing monetary incentives for InPs to participate in the process even under heavy competition. We also present a location-aware VN request forwarding mechanism – basd on a hierarchical addressing scheme (COST) and a location awareness protocol (LAP) – to allow faster embedding. We outline scalability and performance characteristics of PolyViNE through quantitative and qualitative evaluations.

1 Introduction

Network virtualization has gained significant attention in recent years as a means to support multiple coexisting virtual networks (VNs) on top of shared physical infrastructures [1-4]. The first step toward enabling network virtualization is to instantiate such VNs by embedding[a] VN requests onto substrate networks. But the VN embedding problem, with constraints on virtual nodes and virtual links, is known to be $\mathcal{NP}$-hard [5,6]. Several heuristics [5-9] have been proposed to address this problem in the single infrastructure provider (InP) scenario. However, in realistic settings, VNs must be provisioned across heterogeneous administrative domains belonging to multiple InPs to deploy and deliver services end to end.

One of the biggest challenges in end-to-end VN embedding is to organize the InPs under a framework without putting restrictions on their local autonomy. Each InP should be able to embed parts or the whole of a VN request according to its internal administrative policies while maintaining global connectivity through mutual agreements with other InPs.

*Correspondence: rboutaba@cs.uwaterloo.ca
[3]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
[4]Division of IT Convergence Engineering, Pohang University of Science and Technology (POSTECH), Pohang 790-784, Korea
Full list of author information is available at the end of the article

Moreover, InPs (i.e., network operators) are notoriously known for their secrecy of traffic matrices and topology information. As a result, existing embedding algorithms that assume complete knowledge of the substrate network are not applicable in this scenario. Each InP will have to embed a particular segment of the VN request without any knowledge of how the rest of the VN request has already been mapped or will be mapped.

Finally, there will be constant tussles between service providers (SPs) and InPs on multiple levels:

- Each InP will be interested in getting as much of the deployment as possible put on its equipment, and then optimizing allocation under given constraints. In addition, InPs will be more interested in getting requests for their high-margin equipment while offloading unprofitable work onto their competitors.
- SPs are also interested in getting their requirements satisfied while minimizing their expenditure. Tussles might arise between SPs and InPs when each party selfishly try to optimize their utility functions.

Any inter-domain VN embedding mechanism must enforce proper incentives and mechanisms to address these tussles.

In this paper, we introduce PolyViNE, a policy-based end-to-end VN embedding framework that embeds VNs across multiple InPs in a globally distributed manner while allowing each concerned InP to enforce its local policies. PolyViNE introduces a distributed protocol that coordinates the participating InPs and ensures competitive pricing through repetitive bidding at every step of the embedding process.

We do not claim PolyViNE to be the best or the only way of performing end-to-end VN embedding. However, to the best of our knowledge, this is the first foray into this unexplored domain in the context of network virtualization, and we believe this problem to be absolutely critical in realizing network virtualization for most practical purposes.

The rest of the paper is organized as follows. Section 2 formally defines the inter-domain VN embedding problem. In Section 3 we describe the design choices and the distributed embedding protocol used by PolyViNE, followed by a discussion of its enabling technologies in Section 5. Section 6 and Section 7 respectively provide preliminary quantitative and qualitative evaluations of PolyViNE. We discuss related work in Section 8. Finally, Section 9 concludes the paper with a discussion on possible future work.

2 Problem formulation

The intra-domain VN embedding problem is well-defined in the literature [5-9]. In this section, we formally define the inter-domain VN embedding problem. For simplicity, we avoid intra-domain aspects (e.g., node and link attributes) wherever we see fit. We use the notation introduced here to discuss the details of the PolyViNE protocol in section 3.

2.1 Substrate networks and the underlay

We consider the underlay to be comprised of D substrate networks (Figure 1a), and we model each substrate network controlled by the i-th InP ($1 \leq i \leq D$) as a weighted undirected graph denoted by $G_i^S = (N_i^S, L_i^S)$, where N_i^S is the set of substrate nodes and L_i^S is the set of *intra-domain* substrate links. Each substrate link $l^S(n^S, m^S) \in L_i^S$ between two substrate nodes n^S and m^S is associated with the bandwidth capacity weight value $b(l^S)$ denoting the total amount of bandwidth. Each substrate network has a (centralized or distributed) logical Controller [10] that performs administrative/control functionalities for that InP. $A_i^S (\subset N_i^S)$ denotes the set of border nodes [10] in the i-th InP that connect it to other InPs through *inter-domain* links based on Service Level Agreements (SLAs) to form the underlay. $A_{i,j}^S \subset A_i^S$ denotes the set of border nodes in InP_i that lead to InP_j. Each InP also has a set of policies $\mathcal{P}_i^S$ that is used to take and enforce administrative decisions.

We denote the underlay (shown in Figure 1b) as a graph $G^U = (N^U, L^U)$, where $N^U (= \sum_i A_i^S)$ is the set containing border nodes across all InPs ($1 \leq i \leq D$) and L^U is the set of physical inter-domain links connecting the border nodes between two InPs.

However, the underlay does not have the full connectivity, which is achieved through simple topology abstraction

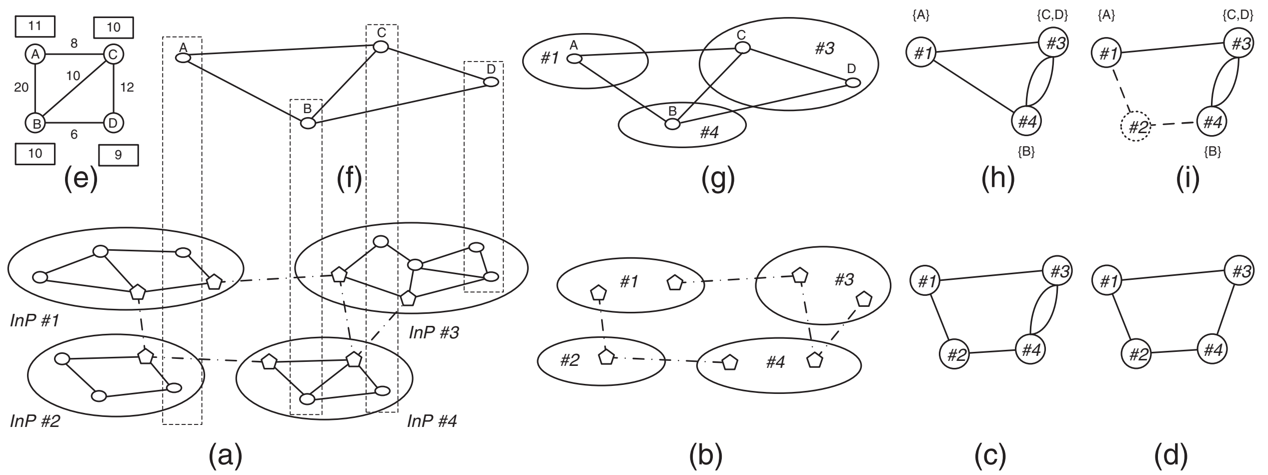


Figure 1 Overview of inter-domain VN embedding: (a) substrate networks ($G_i^S = (N_i^S, L_i^S)$) from four InPs connected using inter-domain links; (b) the underlay ($G^U = (N^U, L^U)$) consisting of border nodes and inter-domain links; (c) the underlay multigraph ($G^W = (N^W, L^W)$) after topology abstraction; (d) controller network ($G^C = (N^C, L^C)$) obtained through simplification; (e) a single VN request ($G^V = (N^V, E^V)$) with CPU constraints in boxes and bandwidth constraints over links; (f) the same VN request (G^V) with location constraints on the virtual nodes shown in vertical boxes covering possible host physical nodes for them; (g) the embedded VN request with virtual nodes mapped into three different InPs; (h) the meta-VN request ($G_M^V = (N_M^V, L_M^V)$); (i) an InP-level view of the embedding (note that, InP #2 has not embedded any virtual node but still it is in the embedding by being in an inter-domain virtual link).

method [11]. All border nodes belonging to a single InP are collapsed to one single node corresponding to that InP (Figure 1c) in this representation resulting in a multi-graph $G^W = \left(N^W, L^W\right)$, where N^W essentially is the set of InPs in the underlay and $L^W (= L^U)$ is a multiset of inter-domain links that connect the InPs. $G^C = \left(N^C, L^C\right)$ is a simple graph (Figure 1d) referring to the controller network [10], where $N^C (= N^W)$ represents the set of Controllers in InPs and L^C is the set of links between Controllers obtained from the multiset L^W.

2.2 VN request

Similar to substrate networks, we model VN requests as weighted undirected graphs and denote a VN request by $G^V = \left(N^V, E^V\right)$. We express the requirements on virtual nodes and virtual links in standard terms [6,8]. Figure 1e depicts a VN request with virtual node and link requirements.

Each VN request has an associated non-negative value R^V expressing how far a virtual node $n^V \in N^V$ can be placed from the location specified by $loc\left(n^V\right)$ [8], which can be interpreted as the preferred geolocation of that virtual node. Figure 1f shows the substrate nodes within the preferred geolocation for each virtual node using dashed vertical boxes.

2.3 VN assignment

From PolyViNE's point of view, an end-to-end VN assignment is performed on the controller network, G^C.

The VN request $G^V = \left(N^V, L^V\right)$ is partitioned into K subgraphs $G_k^V = \left(N_k^V, L_k^V\right)$ such that $N^V = \cup_k N_k^V$ and $L^V = \left(\cup_k L_k^V\right) \bigcup L_M^V$, where L_M^V is the set of virtual links that will cross domain boundaries. In this version of the PolyViNE protocol, we also consider subsets of L_M^V, $L^V_{M^{a_j}_{a_i}}$ where $L_M^V = \cup_{0 \le i \le k-1} \cup_{i<j} L^V_{M^{a_j}_{a_i}}$. We define $L^V_{M^{a_j}_{a_i}}$ to be the set of all virtual links with a single incident virtual node mapped in InP_{a_j} and another node mapped in $InP_{a_l}, 0 \le l \le j$ and an inter-domain path mapping that crosses InP_{a_i}. Thus, $L^V_{M^{a_j}} = \cup_{i \le j} L^V_{M^{a_j}_{a_i}}$ is simply the set of all virtual links crossing inter-domain boundaries with one end mapped InP_{a_j}.

In Figure 1g, $K = 3$: $G_1^V = (\{A\}, \{\})$, $G_2^V = (\{B\}, \{\})$, $G_3^V = (\{C, D\}, \{CD\})$, and $L_M^V = \{AB, AC, BC, BD\}$. Each subgraph G_k^V can be collapsed into a single node to form the meta-VN request $G_M^V = \left(N_M^V, L_M^V\right)$ using a transformation function $\mathcal{F} : G_k^V \rightarrow N_M^V$ (Figure 1h) for simplicity.

Now we can formally express inter-domain VN embedding as two mappings, $\mathcal{M}_N : N_M^V \rightarrow N^C$ that embeds each subgraph to different InP and $\mathcal{M}_L : L_M^V \rightarrow L^C$ that embeds inter-domain links in the InP controller network. Figure 1(i) shows a possible InP-level embedding for the VN request shown in Figure 1(e). Note that, *InP#*2 has not embedded any virtual node but is still in the embedding by being in an inter-domain virtual link.

3 PolyViNE overview

In this section, we discuss PolyViNE design decisions, explain its workflow, and describe the distributed protocol that coordinates the PolyViNE embedding process.

3.1 Design choices

We have made the following design choices for PolyViNE aiming toward decentralization of the embedding process, promotion of policy-based decision making, and support for local agility within a flexible global framework.

3.1.1 Decentralized embedding

PolyViNE argues for using a distributed (decentralized) VN embedding solution over a centralized broker-based one. In a centralized solution, the broker will have to know the internal details and mutual agreements between all the InPs to make an informed embedding. However, InPs are traditionally inclined to share as little information as possible with any party. A distributed solution will allow for embedding based only on mutual agreements. Moreover, in a distributed market there will be no single-point-of-failure or no opportunity for a monopolistic authority (e.g., the broker).

3.1.2 Local autonomy with global competition

PolyViNE allows each InP to use its own policies and algorithms to take decisions without any external restrictions. However, it also creates a high level of competition among all the InPs by introducing competitive bidding at every level of distributed VN embedding. Even though each InP is free to make self-serving decisions, they have to provide competitive prices to take part and gain revenue in PolyViNE. To keep track of the behavior of InPs over time, a reputation management mechanism can also be introduced [12,13].

3.1.3 Location-assisted embedding

PolyViNE decision making and embedding process is deeply rooted into the location constraints that come with each VN request. After an InP embeds a part of a VN request, instead of blindly disseminating the rest of the request, it uses geographic constraints as beacons to route the request to other possible providers. PolyViNE aggregates and disseminates location information about how to reach a particular geographical region in the controller network and which InPs might be able to provide virtual resources in that region.

3.2 Workflow summary

PolyViNE is an enabling framework for multi-step distributed embedding of VN requests across InP boundaries. In its simplest form, an SP forwards its VN request to multiple known/trusted InPs; once they reply back with embeddings and corresponding prices, the SP chooses the VN embedding with the lowest price similar to a bidding process.

However, a complete end-to-end VN request may not be mappable by any individual InP. Instead, an InP can embed a part of the request and *outsource* the rest to other InPs in a similar bidding process giving rise to a recursive multi-step bidding mechanism. Not only does such a mechanism keep a VN embedding simple for an SP (since the SP does not need to contact all of the eventual InPs), but it also ensures competitive prices due to bidding at every step.

3.3 Restricting the search space

Through the PolyViNE protocol, when an InP receives a request to map a virtual network, it selects a connected subgraph of the VN to embed, and passes the remaining portion of the VN graph to other InPs. The process is started at the SP where it spawns off k^{SP} instances of the VN request. At each subsequent stage, InP_{a_i} spawns off $k^{InP_{a_i}}$ copies of the remaining portion of the VN request to an appropriate set of InPs determined by LAP.

The search space for all possible virtual network partitionings across the controller network is vast: $O(D^n)$ where D is the number of InPs in the controller network and n is the number of nodes in the virtual network. Thus, it is infeasible to attempt all possible partitionings of a virtual network across all InPs. PolyViNE, instead, takes a best effort approach to mapping virtual networks onto substrate networks by exploring a constant subset of partitionings.

The PolyViNE protocol attempts to navigate InPs for solutions in a breadth-first-like manner, while fixing maximum depth, d, and varying branching factor based on the pricing model discussed in 3.4, giving an upper bound of $O(k^{SP}(k^{InP_{max}})^d)$ visited InPs where $k^{InP_{max}}$ is defined to be the maximum branching factor at participating InPs. As discussed below, the budget for processing is fixed, and effectively, so is $k^{InP_{max}}$.

3.4 Pricing model

The PolyViNE protocol operates under the assumption that every entity in the controller network is behaving in its own best interest, attempting to maximize its profit. Each entity provides a service (embedding reservation) to its predecessor in the recursive process and requests a service from its successors. It then selects the service that provides the best price and rejects the other services. However, when an InP reserves resources for a partial embedding for its predecessor, it incurs an opportunity cost: those reserved resources could have been used to service another VN request. For simplicity, we assume the opportunity cost is some constant per InP per VN request. Thus, an InP charges its predecessor a processing fee. This has the effect of producing a trade-off between exploration of the space of possible embedding solutions, and price. The more InPs visited and solutions explored, the more processing fees incurred. Thus, a high branching factor (k^{InP}) at an InP can be extremely expensive while a lower branching factor reduces the search space (potentially increasing prices), and increases the chance of failure (not finding a feasible solution to the VN constraints in the search horizon).

In this model, the SP sets an upper bound on the processing fees as a ratio relative to the embedding budget (e.g. $1 : 2$ processing fee to embedding fees). For example, an InP may wish to embed a virtual network for a maximum of \$5000 and pay no more than an additional \$2500 for processing. The processing fee cap implicitly limits the search space. We leave it up to the discretion of the InP to choose how to distribute the processing fee allocation to successors, and how many successors to relay the VN request to (k^{InP}). k^{InP} may, for example, be expressed as a function of the processing fee allocation such that as the allocation grows so does the branching factor.

This model disincentivizes entities (SPs and InPs) from flooding the controller network to search for a cheaper solution. As each InP takes a processing fee, we eventually run out of money in the processing fee allocation, effectively reducing search depth. On the other hand, given a fixed fee allocation, a smaller branching factor increases search depth.

This model also provides InPs an additional incentive to participate in finding an embedding for a given virtual network as it will receive compensation for its work. When an entity sends an *EMBED* message to another entity, it enters a contractual agreement to pay a processing fee up to an upper bound it specifies.

3.5 A running example

To illustrate the details of the PolyViNE protocol, we introduce a simple running example in Figure 2. In this example, an SP issues a VN request (Figure 2a) to InP #1. InP #1 proceeds to map virtual node A, and B and virtual link d in Figure 2 (b)(c). It then forwards the remaining portion of the VN request to InP #2. InP # 2 is unable to map nodes from the VN request, and so it serves as a *relay* InP that may allocate bandwidth resources for virtual links that span multiple domains as need be (links a and c in this example). In turn, InP #2 forwards the remaining portion of the VN request to both InP #3 (Figure 2b) and InP # 4 (Figure 2c). In the recursive process, a sequence of InPs that terminates in failure or ultimately finds a feasible solution that spans that sequence is called a *flow* (see section 4.5). In the example in Figure 2, the solution of the

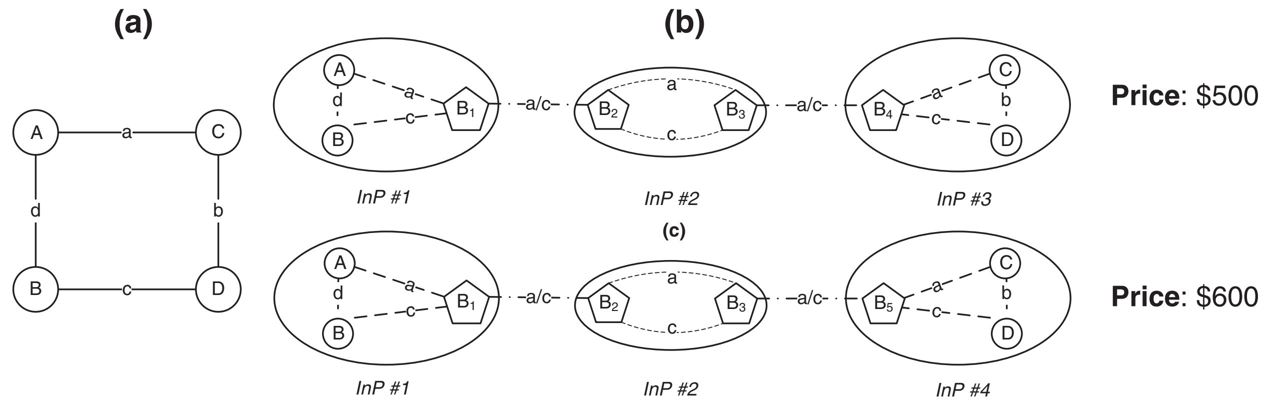


Figure 2 Two mappings and associated prices (b)(c) of a virtual network (a). In the top mapping **(b)**, VN **(a)** is mapped across InP#1, InP#2, and InP#3, while in the bottom mapping **(c)**, the VN is mapped across InP #1, InP #2, and InP #4.

flows in Figures 2 (a) and (b) produce mappings costing \$500, and \$600, respectively.

3.6 PolyViNE embedding protocol

In order to exchange information between the SP and the InPs, and to organize the distributed embedding process, a communication protocol must be established. We refer to this protocol as the PolyViNE Protocol, which is based on eleven types of messages. These messages are sent and received asynchronously between concerned InPs and the SP to carry out the embedding process from beginning to end. The protocol messages are described in the following:

- **EMBED** *(Req_id, G, $\mathcal{M}$, state_table, budget_remaining, processing_allocation_remaining, InPSet)*: This message is sent from the SP to InPs to initiate the embedding process of the VN request G with an empty *InPSet*. Upon receipt of this message, an InP_{a_i} will decide whether to process the request, or report failure based on its policies $\mathcal{P}^S_{a_i}$ as well as processing_allocation_remaining. If InP_{a_i} determines that it requires more money to process the message than is allocated by the SP, then it will report failure. If an InP processes the request but later determines that the allocation would go over budget_remaining, it will cancel the reservation, and report failure.
  An InP also uses this message to outsource the unmapped part of the request after appending itself to *InPSet*, and updating G and the partial embedding $\mathcal{M}$ as necessary. Req_id and state_table together uniquely identify a particular instance of the VN request (see section 4.5).
- **EMBED_SUCCESS** ($pred\_state\_id, \mathcal{M}, Price(\mathcal{M}), succ\_id, InPSet$): Once an embedding is successfully completed, an InP replies back to its predecessor with a price and $\mathcal{M}$. *pred_state_id* is a unique identifier used to call up the relevant state stored at the predecessor entity (SP or InP). The *succ_id* is a unique identifier indicating which InP sent the message.
- **EMBED_FAILURE** (*pred_state_id, succ_id, error_desc*): In case of a failure, an InP replies back with a description outlining the reason of failure using *error_desc*.
- **EMBED_REJECT** (*pred_state_id, pred_id, succ_state_id*): An InP may reject a mapping provided by one of its successors if its mapping does not meet the predecessor InP's policy, $P^S(\mathcal{M}^S) == FAIL$ or a better mapping has been discovered and chosen or the predecessor InP has itself received an *EMBED_REJECT* message and so it must also recursively reject successors.
- **EMBED_REJECT_ACK** (*pred_state_id, succ_id*): When an InP is instructed to reject an embedding, it first recursively rejects any partial embeddings by successors, any inter-domain paths leading to it from predecessors, and finally deallocates all resources allocated locally for the given embedding request instance. Once all that has completed, it reports an acknowledgement to the predecessor that issued the *EMBED_REJECT* message.
- **LINK** $\left(pred\_state\_id, succ\_id, succ\_state\_id, L^V_{M^{a_j}_{a_i}}\right)$: Once an InP, InP_{a_j} finishes mapping a subgraph $G^V_{a_j}$, it sends a *LINK* message to each of its predecessors $InP_{a_i}, 0 \leq i \leq j$ to map $L^V_{M^{a_j}_{a_i}}$ if $L^V_{M^{a_j}_{a_i}} \neq \emptyset$, the set of virtual links that map to inter-domain paths that pass through InP_{a_i} and end in InP_{a_j}.
- **LINK_SUCCESS** (*pred_id, pred_state_id, succ_state_id*): Once InP_{a_i} successfully maps $L^V_{M^{a_j}_{a_i}}$, it reports back the price of the link mapping to InP_{a_j}, along with pred_state_id, a unique identifier used to

call up the latest allocations made at InP_{a_i} for the given VN request instance.

- **LINK_FAILURE** (*pred_id, succ_state_id*): If InP_{a_i} fails to map $L^V_{M^{a_j}_{a_i}}$ due to insufficient resources or policy violations, it reports back *LINK_FAILURE* to InP_{a_j}.
- **LINK_REJECT** (*pred_state_id, succ_id*): If any InP_{a_i} fails to map $L^V_{M^{a_j}_{a_i}}$ or if InP_{a_j}'s partial embedding $\mathcal{M}$ is rejected by an *EMBED_REJECT* message, then InP_{a_j} issues *LINK_REJECT* to all InP_{a_i} requesting they release their reservations for $L^V_{M^{a_j}_{a_i}}$.
- **LINK_REJECT_ACK** (*pred_id, succ_state_id*): Once InP_{a_i} releases $L^V_{M^{a_j}_{a_i}}$, it replies to InP_{a_i} with an acknowledgement for rejecting a successful partial link embedding.
- **EMBED_ACCEPT** (*succ_state_id*): Once an SP decides on an embedding after receiving one or more *EMBED_SUCCESS* messages, it will acknowledge the embedding by directly contacting the InPs involved using this message.

3.7 SP Workflow

Since there is no centralized broker in PolyViNE, each SP must know at least one InP to send the VN request it wants to instantiate. However, sending the request to only one InP can encourage monopolistic behavior and reduce the likelihood of finding a feasible mapping. To create a competitive environment, we argue that an SP should send its VN request to $k^{SP}(\geq 1)$ InPs based on direct contact. Figure 3 depicts an SP sending embedding requests using the *EMBED* message to $k^{SP} = 1$ InPs, for the sake of simplicity. As soon as the receiving InPs have viable embeddings ($\mathcal{M}$) with corresponding prices ($Price(\mathcal{M})$) or they fail, the k^{SP} InPs reply back with *EMBED_SUCCESS* or *EMBED_FAILURE* messages. Once the SP selects an embedding, it proceeds toward instantiating its VN by sending *EMBED_ACCEPT* messages to the InPs involved in the selected embedding and sends *EMBED_REJECT* messages to the InPs involved in unwanted embeddings.

4 InP Workflow

While an SP's workflow is straightforward with a single decision at the end, it shifts much more work to the InPs. An InP has to work through several steps of decision making, organizing, and coordinating between heterogeneous policies to complete the embedding process.

4.1 Local embedding

Upon receiving a VN request, an InP must decide whether to reject or to accept the request. It can reject a VN request outright, in case of possible policy violations or insufficient processing budget provided by the predecessor, returning an *EMBED_FAILURE* message to its predecessor. Even if there are no discernible policy violations, it might still need to reject a VN request if it fails to profitably embed any part of that request or if it fails to find an embedding that meets the budget constraints.

In order to decide which part of a VN request to embed, if at all, the InP can use existing intra-domain VN embedding algorithms [6,8] that can identify conflicting resource requirements in a VN request. This can be done iteratively by looking into the output of the linear programs used in both [6,8] without modifying the actual algorithms presented in those work, and trimming out parts of the virtual network until a feasible solution is found. However, we argue that this heuristic may not be sufficient for high quality or even feasible partial embeddings. In particular, we must ensure that if an InP maps a virtual link, it also maps the two nodes incident to it. We also wish to minimize the number of virtual links that map across multiple domains as inter-domain paths tend to be long and thus are more costly.

In case of a failure, the InP will send back an *EMBED_FAILURE* message (optionally with reasons for the failure). However, sometimes the InP might know of other InPs that it believes will be able to embed part or whole of the VN request. In that case, it will *relay* the VN request forwarding the *EMBED* message to that InP after adding itself to the *InPSet*. In Figure 3, *InP*#2 is relaying the VN request *G'* to *InP*#3.

4.2 Reserving inter-domain paths

PolyViNE expects each InP_{a_j} to complete a mapping of all virtual links in the set $L^V_{M^{a_j}}$ prior to forwarding the remainder of the VN request to new participants. This ensures that no additional InPs will be brought in to participate in the mapping before current participants are sure inter-domain paths are feasible between them and satisfy their respective policies.

At a given stage j of the embedding process, InP_{a_j} receives an *EMBED* message that contains an ordered set of InPs participating so far, *InPset* containing $InP_{a_i}, 0 \leq i < j$. For each virtual link, $l^m_k \in L^V_{M^{a_j}}$, InP_{a_j} must identify the predecessor $InP_{a_m} \in InPset$, $0 \leq m < j$ mapping the other node incident to l^m_k. Once identified, InP_{a_j} adds each l^m_k to $L^V_{M^{a_j}_{a_i}}$ $\forall\, m \leq i \leq j$. Subsequently, for each set $L^V_{M^{a_j}_{a_i}} \neq \emptyset$, InP_{a_j} issues *LINK* messages containing that set to each InP_{a_i} instructing it to map the virtual links in $L^V_{M^{a_j}_{a_i}}$ across its domain.

Once InP_{a_i} receives the set $L^V_{M^{a_j}_{a_i}}$, it must decide how to proceed with link mapping. For each virtual link $l^m_k \in$

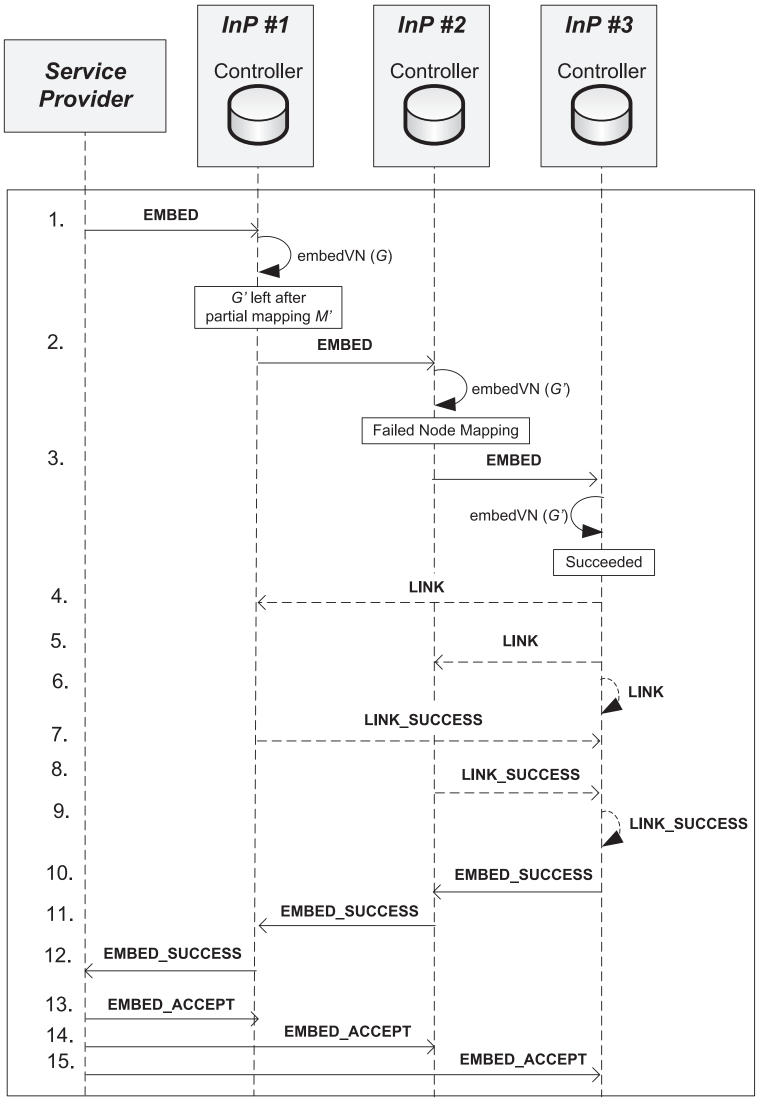


Figure 3 Sequence diagram showing PolyViNE embedding process in action. This diagram illustrates the messages sequence involved in mapping Figure 2(b).

$L^{V}_{M^{a_j}_{a_i}}$, InP_{a_i} considers its index i in *InPset* relative to InP_{a_m} in *InPset*:

1. **i = m:** InP_{a_i} must map a path from the virtual node incident to l^m_k in InP_{a_m} to the border node leading to $InP_{a_{m+1}}$.
2. **m < i < j:** InP_{a_i} must map a path from a border node leading to $InP_{a_{i-1}}$ to the border node leading to $InP_{a_{i+1}}$
3. **i = j:** InP_{a_i} must map a path from the border node leading to $InP_{a_{i-1}}$ to the virtual node incident to l^m_k in InP_{a_i}.

If all the virtual links specified by the *LINK* message are mapped successfully by the receiving InP (no policies are violated and physical resources are available to satisfy the paths), then it responds with a *LINK_SUCCESS* to the sender. Otherwise, the recipient will respond with a *LINK_FAILURE* message to the sender.

An embedding at a given InP is considered successful if and only if at least one node is mapped by the InP and all inter-domain paths are successfully mapped. If one or more predecessor InPs are unable to map inter-domain paths, then the resource reservations must be released. The current InP issues a *LINK_REJECT* message to all InPs that responded with a *LINK_SUCCESS* message. The InP then waits for acknowledgement that resources have been freed through a *LINK_REJECT_ACK* message. Once all pending acknowledgements have been received, the InP releases the resources it allocated locally and issues an *EMBED_FAILURE* message to its direct predecessor (see section 4.9 for more details on roll-back of resource allocations).

4.3 Message complexity

The design of PolyViNE allows for a fairly straightforward analysis of message complexity. If we assume that the implementation of the protocol at each InP does not involve inter-domain paths through new InPs that have not seen the given instance of the VN request, then in the worst case, with n participating InPs, each will visit all predecessors to map virtual links to inter-domain paths. The number of participants is bounded by d. Thus, the message complexity to map a virtual network is $O(min(n,d)^2)$. PolyViNE attempts a total of $O(k^{SP}(k^{InP_{max}})^d)$ mappings (recall section 3.3) giving a total complexity of $O(k^{SP}(k^{InP_{max}})^d min(n,d)^2)$.

4.4 Revisiting our running example

Figure 3 continues our running example by demonstrating the message sequence involved in mapping the VN in Figure 2 (a) corresponding to the flow in Figure 2 (b). A sequence of 15 messages are exchanged:

1. The service provider issues a VN mapping request to InP #1.
2. It performs a partial mapping of the VN request onto its substrate network. The remaining portion of the VN request is forwarded to InP #2.
3. InP #2 fails to map the VN request, and so it *relays* the *EMBED* message to InP #3. Subsequently, InP #3 successfully maps the remaining portion of the request, leaving virtual link to inter-domain path mappings remaining.
4. InP #3 sends a *LINK* message to InP #1 containing the virtual link reference set $L^V_{M^3_1} = \{a, c\}$.
5. InP #3 sends a *LINK* message to InP #2 containing the virtual link reference set $L^V_{M^3_2} = \{a, c\}$.
6. InP #3 sends a *LINK* message to itself containing the virtual link reference set $L^V_{M^3_3} = \{a, c\}$.
7. InP #1 successfully maps paths for the virtual links in $L^V_{M^3_1} = \{a, c\}$ from the substrate node resources reserved for virtual nodes A and B to the B_1 border node. Additionally, it allocates sufficient bandwidth for $L^V_{M^3_1}$ on the B_2B_3 inter-domain link (see Figure 2). It reports back *LINK_SUCCESS* to InP #3 including the total cost of the mapping of $L^V_{M^3_1}$.
8. InP #2 successfully maps paths for the virtual links in $L^V_{M^3_2} = \{a, c\}$ from the B_2 border node to the B_3 border node. Additionally, it allocates sufficient bandwidth for $L^V_{M^3_2}$ on the B_3B_4 inter-domain link (see Figure 2). It reports back *LINK_SUCCESS* to InP #3 including the total cost of the mapping of $L^V_{M^3_2}$.
9. InP #3 successfully maps paths for the virtual links in $L^V_{M^3_3} = \{a, c\}$ from the B_4 border node to the substrate node resources reserved for virtual nodes C and D. It reports back *LINK_SUCCESS* to itself including the total cost of the mapping of $L^V_{M^3_1}$.
10. InP #3 sees that all participating InPs have reported back *LINK_SUCCESS* and so the VN mapping is complete. It accumulates the prices it received from the *LINK_SUCCESS* messages and adds the cost of its own local mapping to produce a total that it sends back to its predecessor InP #2 within an *EMBED_SUCCESS* message.
11. InP #2 receives InP #3's *EMBED_SUCCESS* message. It compares the price it receives from InP #3 (\$300) with that of InP #4 (\$400), rejects InP #4's solution, and selects InP #3's solution (see Figure 4). It adds its own local embedding price (\$0 in this case, as it did not map any nodes, and the link allocations in InP #2 were accounted for by InP #3's offer) to produce a total that it sends back to InP #1 within an *EMBED_SUCCESS* message.
12. InP #1 receives InP #2's *EMBED_SUCCESS* message. In our example $k^{InP_1} = 1$, and so InP #1 immediately adds the cost of its local mapping (\$200) to the price of InP #2's solution (\$300). It reports the total price of the mapping (\$500) back to the SP.
13. After the solution in Figure 2(c) is rejected, the SP accepts InP #1's mapping, allowing the InP to instantiate and setup virtual machines on substrate nodes.
14. The SP accepts InP #2's mapping, allowing the InP to instantiate and setup virtual machines on substrate nodes.
15. The SP accepts InP #3's mapping, allowing the InP to instantiate and setup virtual machines on substrate nodes.

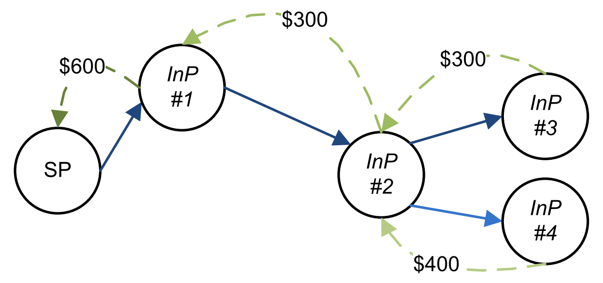


Figure 4 Propagation of multiple instances of the same VN request in the controller network throughout the embedding process. Each InP performs a partial embedding of a request instance and outsources the rest to another InP. The dashed lines demonstrate the back-propagation of accumulated prices toward the SP.

The message sequence corresponding to the solution in Figure 2 (c) would be very similar to the sequence in Figure 3 save for the final few message exchanges. The solution at InP #4 costs more than that at InP #3, and so it will be rejected. In particular, at step 11, InP #4 would receive an *EMBED_REJECT* message from InP #2, and would subsequently issue *LINK_REJECT* messages to each of the participating InPs. InP #4 would wait for *LINK_REJECT_ACK* from the participating InPs, and then will report *EMBED_REJECT_ACK* to its predecessor, InP #2 (see section 4.9 for more details on roll-back of resource allocations).

4.5 Message flows

When an InP receives an *EMBED* message with virtual network G and finds a mapping $\mathcal{M}$ of some subgraph of G, the InP reserves those resources for that particular VN request. The resources are reserved until either the PolyViNE protocol determines that there are no successors that can satisfy the remaining portion of the request or a predecessor has rejected the mapping provided by the current InP. Subsequently, when an InP receives a *LINK* message, it must call up the previous resource allocation associated with the current instance of the virtual network mapping request and bundle any link allocations it performs with the previous resource allocations.

Continuing our running example: In Figure 5, InP #3, and InP #4 both map subgraphs of the VN request issued by the SP. They, then, both independently send *LINK* messages to InP #1 to map inter-domain paths from their respective subgraphs to the subgraph mapped by InP #1.

We define a flow to be an ordered set of InPs visited to map a given virtual network VN_{ij} with a unique identifier i, requested by some service provider, SP_j . It is evident that the flow $f_1^{VN_{ij}} = \{InP\#1, InP\#2, InP\#3\}$ and $f_2^{VN_{ij}} = \{InP1, InP\#2, InP\#4\}$ corresponding to our running example in Figure 2 (b) and (c) respectively are mutually exclusive, as they are two different instances of the same VN request and, ultimately, at most one instance will be accepted by the SP. Thus, while the two instances share the same node embedding state produced by the initial $EMBED_1$ message, their subsequent link allocation state produced by the *LINK* messages are different and independent. Thus, any implementation of PolyViNE must be able to identify flows in order to be able to store and call up the appropriate state specific to that flow. We propose a solution to identify flows and their associated state at an InP. At any point in the embedding process, an InP only sees a prefix of a final InP set. After processing an *EMBED* message, an InP will spawn off k^{InP} mutually exclusive instances of the VN request to map its remaining portion. At this point, the k^{InP} flows diverge but they share a common prefix: the state formed in response to the *EMBED* message.

We propose bundling the allocations performed in response to *EMBED* and *LINK* messages in transactional state objects which we will call *EmbeddingState* objects from this point forward. Allocating an *EmbeddingState* object also reserves an associated unique state identifier, called an *EmbeddingId*, which is used to call up state. *EmbeddingIds* at each InP in the flow so far are bundled in *EMBED* messages in a state table. The state table is simply a mapping from a unique InP identifier (such as an IP address), to an *EmbeddingId* for the latest *EmbeddingState* object for the flow at the given InP.

When a given InP, InP_{a_j} sends a *LINK* message to a predecessor InP_{a_i}, it includes the *EmbeddingId* associated with the flow at InP_{a_i} that it received through *EMBED* message's state table. InP_{a_i} creates a new *EmbeddingState* object, as a child of the previous *EmbeddingState* for that flow, along with a new *EmbeddingId*. When the link allocation successfully completes, InP_{a_i} reports *LINK_SUCCESS* back to InP_{a_j} along with the new *EmbeddingId*. As the PolyViNE embedding process progresses, InPs participating in the process begin to form chains of *EmbeddingState* objects associated with the flow. Flows that share a common prefix may share a prefix of an *EmbeddingState* chain, but diverge at some point, thus forming an *EmbeddingState* tree. Flows that do not share a prefix form disconnected state at any given InP. Thus, for a given InP and a given VN request, PolyViNE state consists of a forest of state trees.

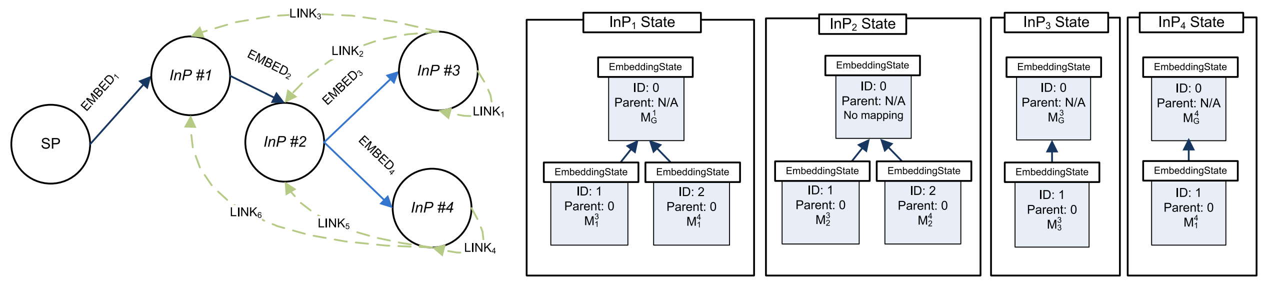


Figure 5 EmbeddingState trees generated by two flows at InP #1, InP #2, InP #3, and InP #4.

In our running example, in Figure 5, the two flows $f_1^{VN_{ij}}$, and $f_2^{VN_{ij}}$ generate four state trees at the four participating InPs. For the sake of brevity, we carefully examine how just InP #1's state tree was formed:

1. When InP #1 maps a part of the VN request VN_{ij}, it allocates an *EmbeddingState* object, stores the mapping $\mathcal{M}_G^1$ with the *EmbeddingId* 0.
2. The mapping $(ID(InP\#1) \rightarrow 0)$ is stored in the state table attached within the message $EMBED_2$.
3. InP #3, and InP #4 send the messages $LINK_3$ and $LINK_6$ respectively to InP #1 with the field *pred_state_id* = *EmbeddingId* 0 allowing InP #1 to call up the state associated with $\mathcal{M}_G^1$.
4. When InP #1 maps the link allocation, $\mathcal{M}_1^3$, it creates a new *EmbeddingState* object with a InP-level unique *EmbeddingId* 1 as a child to the *EmbeddingState* object housing the $\mathcal{M}_G^1$ mapping. Similarly, when InP #1 maps the link allocation $\mathcal{M}_1^4$, it creates another *EmbeddingState* object attached to the same parent with *EmbeddingId* 2.
   InP #1 responds to InP #3 and InP #4 with *LINK_SUCCESS* messages with *pred_state_id* = 1 and 2, respectively.
   When a subsequent message requests state associated with one of the two new *EmbeddingId*s, we are able to disambiguate between the two flows, despite their sharing a common subgraph mapping.

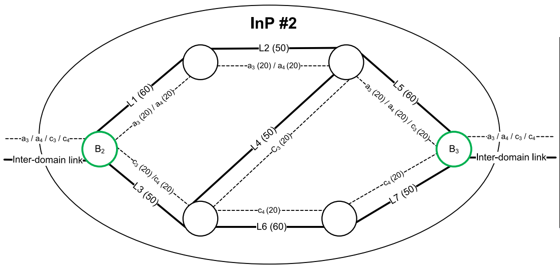
4.6 Resource management

Resource reservation blowup is a major issue in any implementation of PolyViNE. In our implementation of a simulation of the PolyViNE protocol, we quickly realized that in the worst case, the resources allocated in a single InP could grow exponentially as a function of d, the maximum search depth. The total number of flows explored by PolyViNE to find a good inter-domain VN embedding is $O(k^{SP}(k^{InP_{max}})^d)$. In a pathological case, some InP, InP_{a_j} may be involved in as many as $O(k^{SP}(k^{InP_{max}})^d)$ flows. If an InP reserved resources for each flow of a given VN, then we could very quickly end up in a situation where a single, relatively simple VN request drains an InP of all its available resources.

However, we note that for any given VN request, a service provider will ultimately only accept a single flow of the $O(k^{SP}(k^{InP_{max}})^d)$ which will be explored. This means that an InP only needs to reserve sufficient resources for any one such flow per VN request. We denote the capacities on all substrate nodes and links in InP_{a_j}, $C(G_{a_j}^S) = (C(N_{a_j}^S), L(N_{a_j}^S))$ with vectors $C(N_{a_j}^S) = \{C(n_{a_j}^0), C(n_{a_j}^1), ...\}$ for nodes and $C(L_{a_j}^S) = \{C(l_{a_j}^0), C(l_{a_j}^1), \ldots\}$ for links where each component indicates the maximum capacity of that resource. Each *EmbeddingState* object, $E_{a_j}^k$ at InP_{a_j} with *EmbeddingId* k can be thought to be composed of two resource allocation vectors, one for nodes and one for links, indicating the resources allocated by that state object in addition to its parent state object: $C(E_{a_j}^k) = (C(N_{a_j}^k), C(L_{a_j}^k)) + C(Parent(E_{a_j}^k))$.

The InP then simply applies a component-wise maximum (cmax) of the multiset representing all resource allocation state for a given VN request, VN_{b_i}, $E_{a_j}^{b_i} = \{C(E_{a_j}^k) : \forall\ EmbeddingId\ k \in VN_{b_i}\}$ and reserves those resources. As more allocations are made for VN_{b_i} at InP_{a_j}, the InP updates its $E_{a_j}^{b_i}$ multiset and adjusts its resource reservations accordingly, leaving $C(G_{a_j}^S) - C(E_{a_j}^k)$ resources available for other VN requests. As all the mutually exclusive flows are for the same virtual network, the allocations at any given InP will be relatively similar and so, the component-wise maximum will typically not be much more than those required of any one flow.

The internals of our running example: Let's consider the substrate network of InP #2 in Figure 6 of our running example. Recall from Figure 5, InP #2 has three *EmbeddingIds* (0, 1, and 2) associated with state for VN_{ij}. In Figure 6, we look at the resources allocated by the mappings associated with each of the three *EmbeddingIds*. *EmbeddingId* 0 has no associated resource reservations. *EmbeddingId* 1 and 2 refer to mutually exclusive resources allocated by InP #2 on behalf of InP #3 and InP #4



	L1	L2	L3	L4	L5	L6	L7
EmbeddingId 0	0	0	0	0	0	0	0
EmbeddingId 1	20	20	20	20	40	0	0
EmbeddingId 2	20	20	20	0	20	20	20
Capacity	60	50	50	50	60	60	50
CMAX	20	20	20	20	40	20	20
Available Resources	40	30	30	30	20	40	30
Savings	*20*	*20*	*20*	*0*	*20*	*0*	*0*

Figure 6 **A look at the substrate network and resource reservation vectors of InP #2 in our running example.**

respectively. Both resource vectors correspond to mappings of virtual links a, and c from border node B_2 to border node B_3. Note that the two resource vectors correspond to similar, but not identical mappings onto InP #2's substrate network. At most, the SP will accept one of the flows, and so we don't need to allocate resources so that all of the vectors can be satisfied simultaneously. Instead we take the component-wise (i.e. per-resource) maximum resource requirements and reserve that ($CMAX$) vector. Note that the last row of the table in Figure 6 indicates significant resource savings as a result of this technique. Typically, we can expect the savings to grow linearly with the number of flows, enabling much larger search spaces.

4.7 Forwarding

If an InP can only partially embed a VN request, it will have to forward the rest of the request to other InPs in the controller network in order to complete the VN request. An InP should take care to not forward a VN request to another InP already in the *InPSet* to avoid cycles. For example, *InP*#1 in Figure 3 is forwarding the unmapped VN request *G'* to *InP*#2. Similar to SPs, InPs also forward the request to $k^{InP}(\geq 1)$ InPs for similar reasons (e.g., competitive prices). While forwarding a request, an InP can prefer to perform a transformation on the VN request in order to hide the details of its mapping (as in Figure 1h). At this point, it can use one of the two possible methods for unmapped VN request forwarding:

- *Recursive forwarding:* In this case, when an InP forwards a VN request, the receiver InP embeds part of it based on its policies and forwards the rest further away to another InP.
- *Iterative forwarding:* In iterative forwarding, the receiver InP return the control back to the sender InP once it is finished with embedding.

In any case, the forwarding decision is a non-trivial one and requires careful consideration. We believe that instead of blindly forwarding based on some heuristics, we can do informed forwarding by utilizing the location constraints attached to all the virtual nodes in a VN request. Details of this forwarding scheme are presented in the next section.

4.8 Back-propagation

The VN request proceeds from one InP to the next, until either the maximum number of participants d has been reached, there are no available InPs to send the request to or the VN request has been satisfied completely. In case of a successful embedding of a VN request, the *EMBED_SUCCESS* message carries back the embedding details and corresponding price. At each step of this back-propagation of *EMBED_SUCCESS* and *EMBED_FAILURE* messages, the sender InP can select mappings based on internal policies or lower price or some other criteria and rejects the other successful embeddings by issuing *EMBED_REJECT* messages to the appropriate successors.

As VN embeddings follow paths back to the SP, the prices are accumulated and the SP ends up with multiple choices (Figure 4).

4.9 Resource allocation roll-back

Within a single domain, a VN embedding is transactional in nature, as an embedding must be completed as a whole, or not at all. In the multi-domain scenario, each InP is free to map a subgraph of the embedding and so the algorithm used to perform that partial mapping may or may not be transactional (it's up to the discretion of the InP how to implement it). However, from the SP's perspective, the multi-domain scenario is the same as that of the single domain: it expects either a completed embedding reservation or a report of failure. In other words, in a given flow, either all participating InPs succeed in reserving resources or none of them reserve resources. This means that the PolyViNE protocol itself must provide a mechanism to roll-back the work done by other InPs in a given flow once a failure or rejection occurs. An SP is expected to accept only one flow, and reject all other flows. Rejection initiates a roll-back process of all resources allocated for that flow.

Two messages in the protocol can initiate resource allocations within an InP: *EMBED*, and *LINK*. Thus, corresponding roll-back messages must exist in the protocol: *EMBED_REJECT* and *LINK_REJECT*. In order to simplify the implementation of a controller's PolyViNE message handling system and avoid race conditions, associated acknowledgement messages *EMBED_REJECT_ACK* and *LINK_REJECT_ACK* act as barriers to ensure that roll-back occurs in the opposite order to allocation. Note that link allocations corresponding to the set $L^{V}_{M^{a_j}}$ for a given InP_{a_j} are unordered as there are no dependencies among them. However, state dependencies exist between subgraph allocations on one InP and the next, and so PolyViNE ensures that roll-back occurs in the opposite order to allocation through the **_ACK* messages.

As previously discussed, an InP, InP_{a_j}, will report back *EMBED_FAILURE* in case it fails to map the remaining portion of an embedding. Failure reasons include:

1. The embedding is incomplete at InP_{a_j} and no successors are found or all successors fail for some reason.
2. An InP participating in an inter-domain path allocation on behalf of InP_{a_j} responds back to InP_{a_j} with *LINK_FAILURE*.
3. An embedding solution fails to meet the budget constraints.

4. The processing fee budget has been exhausted.
5. Internal InP policies do not allow the embedding to proceed.

A variation of our running example: To illustrate the roll-back process, we consider a variation of our running example shown in Figure 7. In this variation, we assume that InP #2 is unable to map $L^V_{M_2^3} = \{a, c\}$ or $L^V_{M_2^4} = \{a, c\}$ onto its substrate network. Thus, in step 8, InP #2 reports *LINK_FAILURE* to InP #3 (and InP #4, not shown). In the sequence digram, we see that InP #3 observes that InP #1 and InP #3 were able to map their respective *LINK* requests, but InP #2 was not. As an inter-domain embedding must either succeed wholly or release all resources across all participating InPs, InP #3 must now begin the roll-back process.

InP #3 issues *LINK_REJECT* messages to InPs #1 and itself (for consistency purposes) with the appropriate *EmbeddingIds* informing them to deallocate their respective allocations for virtual links a and c. InP #1 and InP #3 release their link mappings, and report *LINK_REJECT_ACK* to InP #3 (steps 12-13). Subsequently, InP #3 releases its subgraph embedding for the VN request and reports *EMBED_FAILURE* to its predecessor, InP #2 (step 14). InP #2 also sees that InP #4 has failed. Thus, InP #2 has seen that all its successors fail, and so it must also fail. InP #2 has no subgraph embedding,

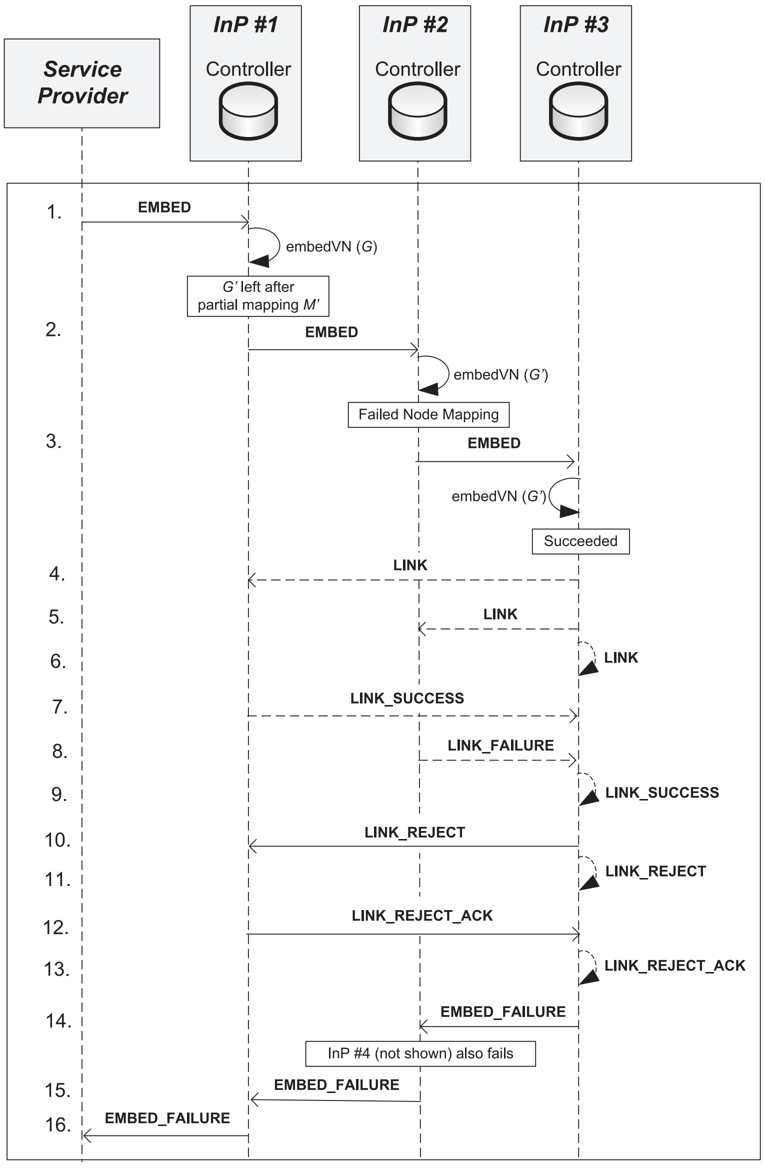


Figure 7 A variation of the running example: sequence diagram showing the roll-back of embedding reservations.

and so it simply reports *EMBED_FAILURE* to its predecessor InP #1 (step 15). InP #1 has only one successor and so it must fail as well. It releases its subgraph embedding for the VN, and reports *EMBED_FAILURE* to the SP.

5 Location aware forwarding

Naïvely an InP can forward a VN request to a set of InPs in the controller network at random. However, this decision is blind to the location requirements of the virtual nodes and the availability of virtual resources at the destination InP to satisfy the constraints for the VN request. This may result in high failure rate or prices well above the fair value. To avoid flooding a VN request or sending it to random InPs which might be unable to meet the constraints of the request, we propose using location constraints associated with unassigned virtual nodes to assist an InP in making this decision. Location constraints of the virtual nodes together with the location information of the underlay will allow informed VN request forwarding in the controller network.

To accommodate such location aware forwarding, we introduce a hierarchical geographic addressing scheme with support for aggregation, named COST. InPs in PolyViNE must associate COST addresses with all the substrate nodes and SPs must express location requirements in terms of COST. Controllers in different InPs publish/disseminate information about the geographic locations of their nodes along with the unit price of their resources. They can then aggregate and disseminate data collected from all neighboring Controllers to build their own knowledge bases of location to InP mappings, each accompanied by path vectors of InPs in the controller network and corresponding prices. We propose Location Awareness Protocol (LAP) to perform this task. Careful readers will notice in the following that COST and LAP are significantly influenced by BGP.

5.1 COST addressing scheme

As outlined in the problem formulation (Section 2), each virtual node in a VN request comes with a permissible geographic region in which it must be embedded. One design question at this point is how to represent and encode the geolocation. We have chosen a hierarchical geolocation representation scheme similar to [14] with the form *Continent.cOuntry.State.ciTy* (hence the name *COST*). Even though in this paper we are using a simple postal address like scheme for simplicity, any hierarchical geolocation representation system will work with PolyViNE.

A virtual node may restrict its location preference to any prefix in this addressing scheme. For example, to restrict a node within Canada, one may assign the address NA.CA.* to a virtual node. This indicates that beyond requiring that the node be mapped within Canada, the SP does not care where in the country it is ultimately mapped.

On the other hand, each substrate node has a complete COST address associated with it. This address indicates within which city lies the given substrate node. If an InP is not willing to share the exact location, it can always choose a higher level address. For example, instead of announcing nodes in Toronto using NA.CA.ON.Toronto, the InP can announce NA.CA.ON.*. However, such announcements can result in receiving of VN requests that it may never be able to satisfy, which will affect its reputation among other InPs.

5.2 Location awareness protocol (LAP)

Location Awareness Protocol (LAP) is a hybrid of Gossip and Publish/Subscribe protocols that assists an InP in making informed decisions about which InPs to forward a VN request to without making policy violations, and thus progressing toward completing the VN embedding. Controllers in different InPs keep track of the geolocations of their internal substrate nodes in COST format and announce availability and prices of available resources to their neighbors using LAP updates in the controller network. This information is aggregated and propagated throughout the controller network to create global view of the resources in the underlay in each Controller's LAP database.

Initially, LAP operates as a path vector based gossip protocol. Every InP in the controller network informs its neighbors of where its nodes are located along with estimated unit prices for its resources on a per location basis. Whenever a Controller receives a LAP update, it updates its LAP database and before announcing updates to its neighbors it adds itself to the path vector. Note that keeping complete paths allows avoiding unnecessary forwarding toward and through InPs that might violate SP's policies or originating InP's policies. InPs can also tune this price to encourage or discourage VN request forwarding to them. In steady-state, each InP should know about all the InPs with nodes in a given geographic region along with price estimations of embedding on their substrate networks. Figure 8 shows an example LAP database.

However, in a rapidly changing environment with continuously fluctuating prices, gossip may not be sufficient to disseminate updated prices in a timely fashion. To reduce the number of failures stemming from staleness of pricing information, we propose extensions to LAP using a Publish/Subscribe mechanism along with its basic gossip protocol. By using this mechanism, any InP will be able to subscribe to announcements of Controllers that are not its direct neighbors. While we leave VN request routing decisions to the discretion of InPs, an InP may use the pricing information to prefer forwarding the VN request to a lower priced InP, all other things being equal.

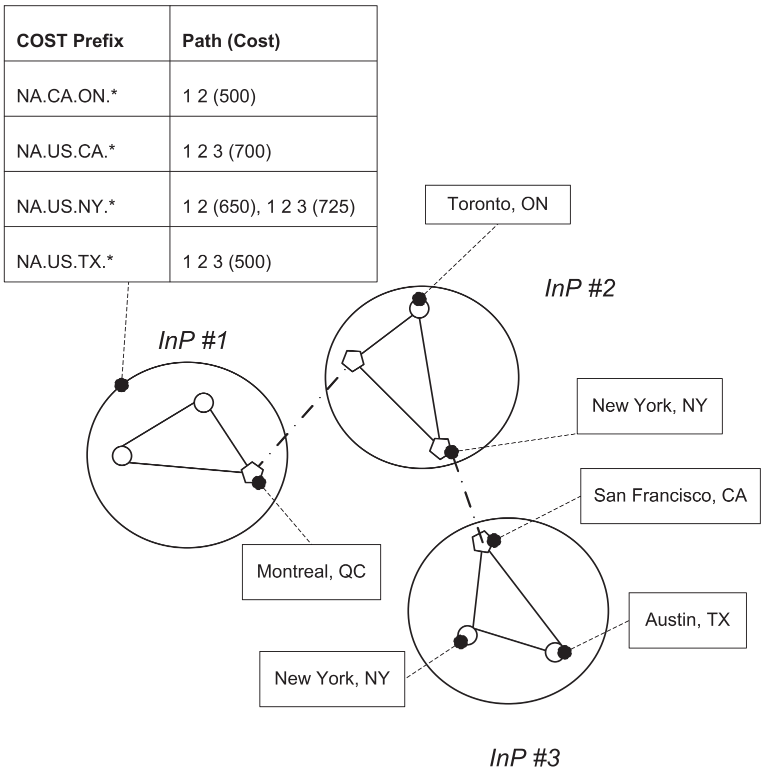


Figure 8 LAP database at InP #1. InP #1 has two choices to forward to an InP with a node in New York state.

The question that remains open to more investigation is why would an InP be honest when announcing pricing estimates? We believe that a reputation metric – indicating long-term accuracy of an InP's pricing estimate to the actual cost of establishing a VN request – is necessary to remedy this situation. We would like to integrate such a reputation metric within LAP to allow dissemination of path vectors attributed with corresponding prices and overall reputation score of the InPs on the paths. An InP will then be able to use pricing and reputation scores to rank multiple paths to a common destination to make a forwarding decision.

6 Numerical evaluation

We have written a 12000 line multi-threaded C++ simulator that allows independent responses from various entities in the controller network. The simulation comprises a complete implementation of the entire set of protocol messages discussed in section 3.6.

We examine four sets of experiments. In our first set of experiments, we look at some of the properties of inter-domain embeddings generated by PolyViNE as the VN request size (node count) is varied. In our second set of experiments, we look at some of the properties of the PolyViNE embeddings as the pricing model attributes are varied (embedding and processing budgets). In our third set of experiments, we look at properties of PolyViNE embeddings as we vary the maximum number of InPs involved in a mapping. In our last set of experiments, we examine the reliability of the PolyViNE protocol and the cost of embeddings generated in the case of lost or stale LAP information.

Each experiment is run to completion (a VN request has completed) multiple times per data point to produce the averaged results presented here. We found that that variation in tests was very small and so we did not include confidence intervals. Unless otherwise specified, we have used the following settings: For each experiment, we randomly create a controller network with 60 InPs. Each InP network consists of 120 to 150 nodes and 540 to 600 links on average. Each node has a maximum CPU capacity uniformly chosen from 1 to 100 CPU units, and each link has a maximum bandwidth capacity of 100 bandwidth units. Locations of substrate nodes are sampled from a normal distribution with a mean, and variance chosen uniformly from 0 to 255 representing 256 different major cities. InPs with low variance location distributions are effectively local or regional InPs, while high variance InPs have nodes that span the globe. The per unit cost per resource is chosen from a normal distribution with a mean sampled from a prior, per InP, normal distribution (of mean 4, variance 1) and a variance of 1. This means that some InPs will tend to be cheaper than others, on average.

Unless otherwise specified, each VN request has an expected value of 30 nodes, and 120 links (±20%). Each virtual node has a maximum CPU capacity uniformly chosen from 1 to 25, and each virtual link has a maximum bandwidth capacity of 1 to 15, chosen uniformly as well. Locations are chosen uniformly from 256 major cities represented in the controller network.

Each InP charges an *embedding processing fee* of 1 unit (±20%). A maximum InP count per flow (d) is a property of the VN request. Based on this property, an InP estimates the maximum branching factor it can use so that up to d InPs can be involved in a flow and uses that branching factor. Thus, the entire processing budget is always consumed. The processing fees are not refunded if the flow fails.

6.1 Varying VN request size

We begin by varying the number of nodes in a VN request and observing the properties of the embeddings and the success rate of the flows. In each experiment, VN requests have n nodes (where n is varied) and $4n$ links. We also fix $K^{SP} = 7$, maximum InPs per flow to 9, processing budget : embedding budget to 0.75:1, and embedding budget to 25000. The goal of this experiment is to test the limits of PolyViNE's ability to find solutions given a fixed set of resources available to it (Large VN requests relative to InP resources, limited processing and embedding budgets, and thus limited search space). We expect that after a certain point, VN requests will get so large that no mapping will be possible, either because the request hits InP resource limits or it hits budget limits.

In our first experiment in Figure 9, we look at the number of nodes mapped by the first set of InP neighboring the request-generating SP as we vary the VN request size. Figure 9 demonstrates that the number of nodes mapped by the first-hop InPs grows linearly with the size of the VN request. With small requests, virtually the entire network is mapped by the first InP. As request sizes approach the limits of the resources available at the first-hop InP, the number of nodes mapped by the first-hop InP flattens out at about 35 nodes. When we attempted random VN requests larger than 45 nodes, we found that no solutions are found by the PolyViNE protocol, regardless of the size of the search space.

In our second experiment in Figure 10, we looked at the number of InPs that are involved in a successfully satisfied VN request. In this experiment, we only consider InPs that contribute substrate node resources to the VN mapping, and not InPs that simply reserved bandwidth as *relays*. We see that the the number of InPs involved appears to grow linearly with the size of the VN request but with a very small slope.

In our third experiment in Figure 11, we look at the fraction of successful flows relative to the total number of flows of a VN request. We see that up to 35 nodes, all flows are successful. After 35 nodes the success rate drops dramatically, and after 50 nodes (not shown), it reaches 0.

In our fourth experiment in Figure 12, we look at the the impact on embedding cost as we increase the VN request size. We see a linear relationship between the VN request size and the embedding cost (as is expected), where roughly each node added to a VN request costs about 250 units. Somewhat surprisingly, the higher cost of virtual links mapped across inter-domain paths is not apparent here. This may be due to the relative sparseness of the VN requests we have examined.

6.2 Varying embedding budget and processing fee budget

In this experiment, we vary the embedding budget and processing budget of a VN request and observe

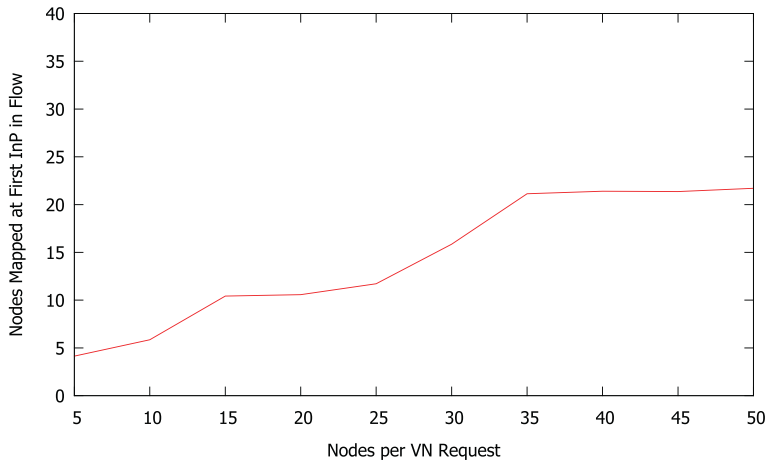


Figure 9 The number of nodes mapped by first-hop InPs increases linearly with the size of sparse VN requests (*n* nodes, 4*n* links).

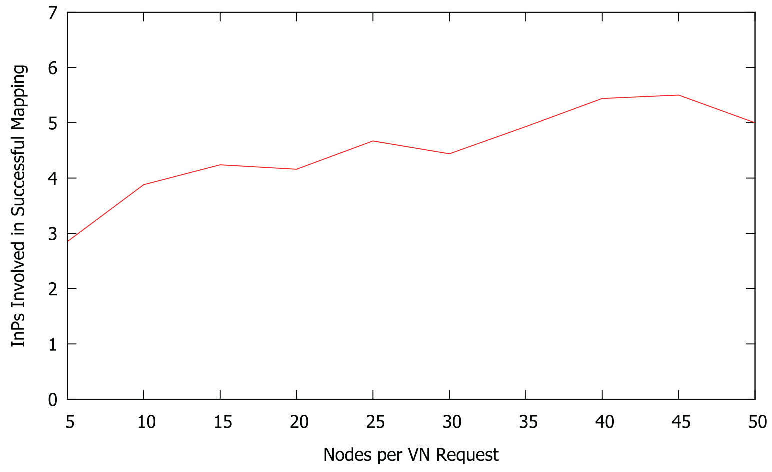


Figure 10 **Number of InPs involved in a successful mapping with increasing VN request size.**

their impact on the success rate and total embedding cost (embedding + processing). In each experiment, VN requests have 30 nodes and 120 links (± 20%). We also fix $K^{SP} = 7$, and maximum InPs per flow to 8.

In our first experiment of this set (Figure 13), we vary both the processing budget and the embedding budget, and observe their impact on the success rate of flows of the VN request. We observe that relatively small changes to the processing budget have little effect on the success rate of a VN request flow, with significant variance up and down as the processing budget increases. This can be attributed to the high cost of increasing k^{InP} at any given InP. Given d, the target maximum number of InPs visited per flow, each InP picks a k^{InP} so that sufficient money remains in the processing budget so that up to d InPs are involved in a flow, if necessary. In other words, for an InP to increase the branching factor k^{InP} by x%, its processing budget must increase by $O((1 + \frac{x}{100})^{d_{remaining}})$ where $d_{remaining}$ is the number of hops remaining to reach the maximum d. Thus, small increases in processing budget will have negligible impact on the search space explored, except in $InP_{a_{d-1}}$ of a given flow.

Budget allocation is distributed at each InP assuming a full n-ary subtree. Thus, at least half of the InPs in the search space are last hop InPs given a fixed search depth. Since processing budget is evenly distributed across all subtrees, to explore more space, the processing budget would need to increase significantly (dependent upon the current average branching factor on the second last hop).

However, we observe a much clearer correlation between the embedding budget and the success rate. A higher embedding budget tends to improve the flow success rate. Also, we see that as we increase the embedding budget, the variation in success rate between processing budgets decreases, suggesting that a larger processing budget does allow for more flows, but most of those new

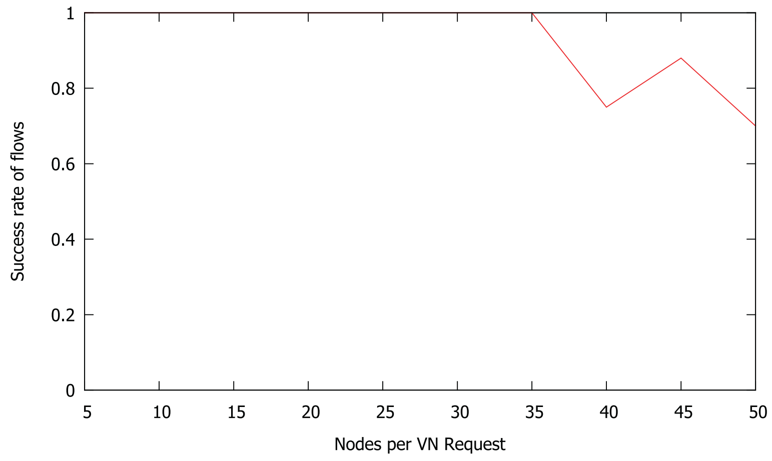


Figure 11 **Rate of successful flows with increasing VN request size.**

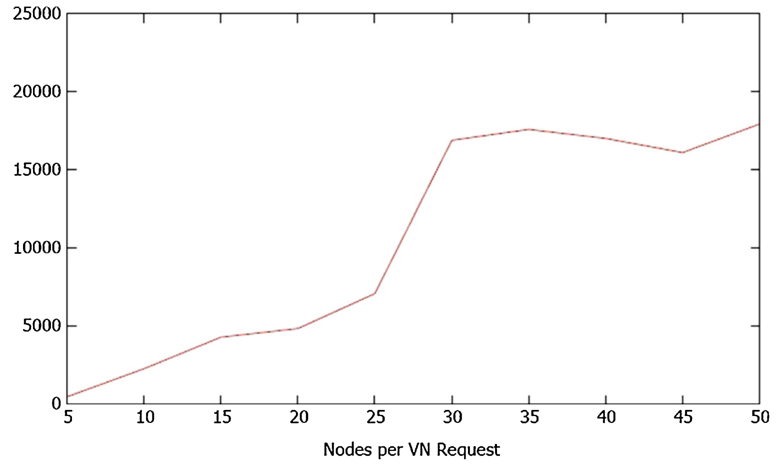


Figure 12 **Embedding cost with increasing VN request size.**

flows go above the embedding budget. As we increase the embedding budget, fewer of the new flows go over the embedding budget.

In our second experiment of this set (Figure 14), we vary the processing and embedding budgets again but this time, we observe their impact on the total cost of an embedding. We observe a linear growth in total embedding cost as we increase the processing budget and the embedding budget. In our experiments we were unable to find a peak that balances the tradeoff between the processing budget (and hence the size of the search space) and the total embedding cost. Looking at the results in Figure 14 we see that much of the total embedding cost is going into processing fees. As the processing budget is on a per-first-hop InP basis, increasing k^{SP} also increases the processing costs.

In Figure 15, we drop the processing fees, and look at the impact varying the budget and processing fees has on just the embedding cost and not the total cost to the SP. We observe that varying the processing budget has relatively little impact on the cost of the embedding. This supports the argument above that states that a large change to the processing budget is necessary to observe a measurable change to the search space. This suggests that under this pricing model, an SP should pick its processing budget high enough to produce an acceptable success rate, and not to find cheaper solutions.

6.3 Varying maximum InPs per flow

The maximum number of InPs involved in a single flow (d) impacts the size of the search space explored. As d decreases, the branching factor per InP tends to increase.

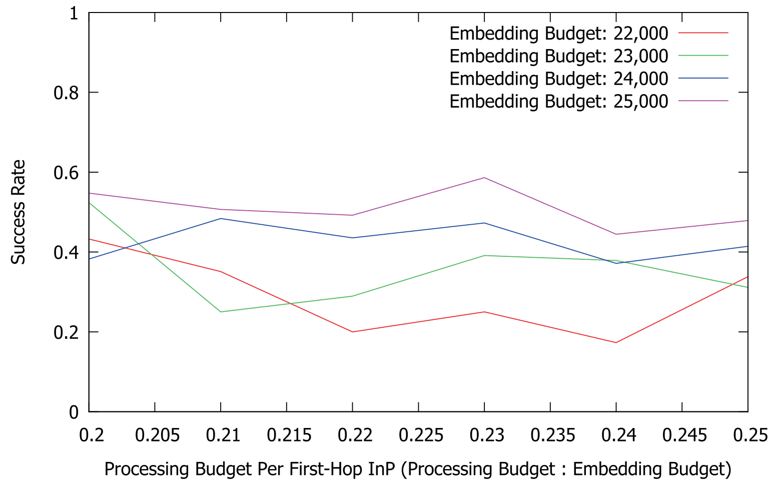


Figure 13 **Success rate with varying embedding and processing budgets.**

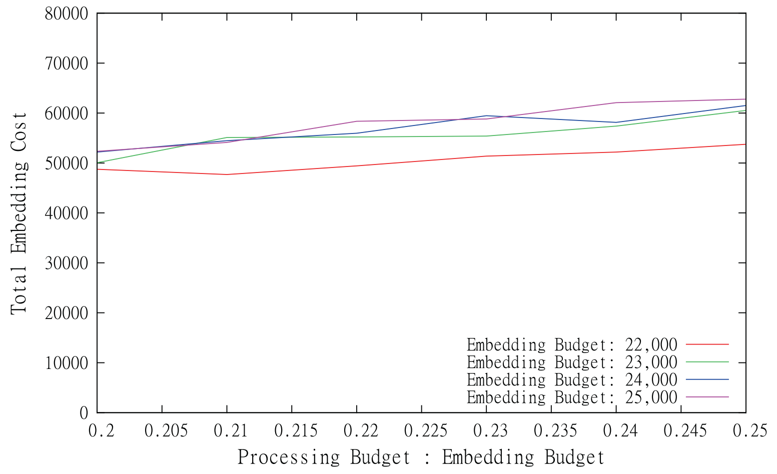


Figure 14 **Total embedding cost with varying embedding and processing budgets.**

In this experiment, we investigate the tradeoff between increasing the number of flows (k^{InP}) and the maximum number of InPs involved per flow (d) given varying processing budgets ratios (processing budget : embedding budget), and a fixed embedding budget of 25,000.

In the experiment in Figure 16, we see that there is a notable correlation between the maximum number of InPs involved in a flow, and the success rate. Given a fixed processing budget, increasing the maximum number of InPs involved (and thereby decreasing the branching factor k^{InP}) tends to increase the success rate. This is significant because it suggests it may be possible to lower the processing budget without impacting the success rate by increasing the value of d.

In Figure 17, we see there is a very weak correlation between d and the total embedding cost on random graphs.

6.4 Varying LAP update message drop rate

In the final set of experiments, we wish to assess the reliability of the PolyViNE protocol when faced with InPs that have stale LAP information. The propagation rate of LAP data can vary by relationships between InPs and by location and so we wish to ensure that PolyViNE is able to function under a variety of conditions.

In Figure 18, we see that PolyViNE is extremely resilient to dropped LAP update messages. The success rate is largely unimpacted by dropped LAP updates until about 95% of updates are dropped after which, we see a significant drop in success rate of flows. PolyViNE is designed to always forward VN requests to some neighboring InP, even if it cannot find an InP that matches the location constraints of any of the unmapped nodes in the VN request. If an InP cannot map any nodes, it acts as a *relay* and then uses its own LAP data to determine where to forward

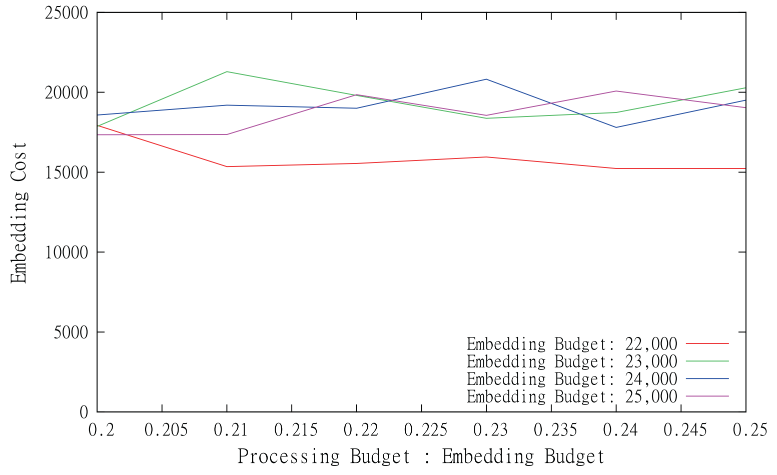


Figure 15 **Embedding cost only (no processing fees included) with varying embedding and processing budgets.**

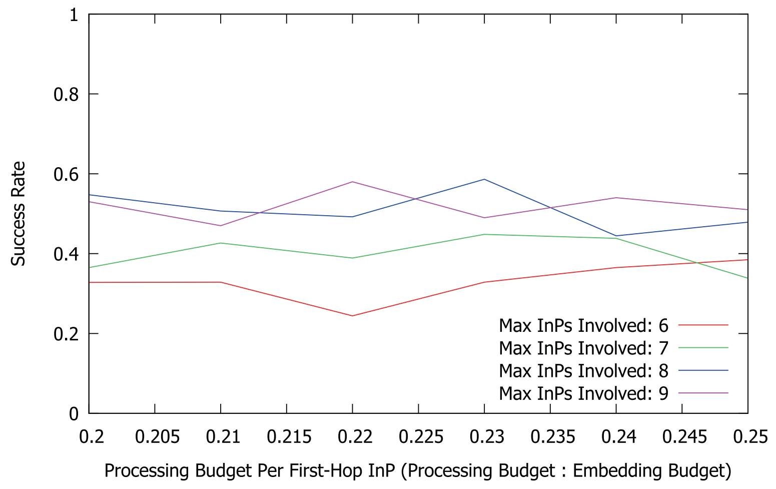


Figure 16 Rate of successful flows with varying *d* and processing budgets.

next. This makes PolyViNE extremely resilient to failing as a result of stale or lost LAP data. However, missing LAP data might affect the quality (cost) of an embedding.

In Figure 19, we delve a bit deeper to see how PolyViNE is so resilient. We observe that the number of flows explored actually **increases** as the drop rate of LAP messages increases (with no changes to the processing budget) This is very counterintuitive. How are we able to explore more flows? Figure 20 sheds some light on this. We see that the number of InPs involved per flow increases as the LAP drop rate increases. This means that each InP is mapping a smaller portion of the VN request, and so the embedding budget allows for more InPs to be involved per flow. Each additional InP spawns off k^{InP} more flows.

In Figure 21, we look at the impact dropped LAP messages have on the embedding cost. Presumably, with less LAP data at every InP, VN request forwarding is effectively blind. We see that this intuition appears to be correct, after about 80% of LAP updates are dropped. As we lose LAP information, forwarding becomes less informed and so partial mappings are not always done at the cheapest InPs. 80% is also about when we begin to notice additional flows (Figure 19), and so at least some of the increase can be attributed to the additional inter-domain paths required by partitioning the VN across more InPs.

7 Discussion

7.1 Pricing model

The simple pricing model we suggested in this report succeeds in accomplishing the goal of incentivizing InPs to participate in a highly competitive environment and disincentivizing flooding the controller network to find feasible, low-cost embeddings. However, we did not study the practical implications of this pricing model. It may be

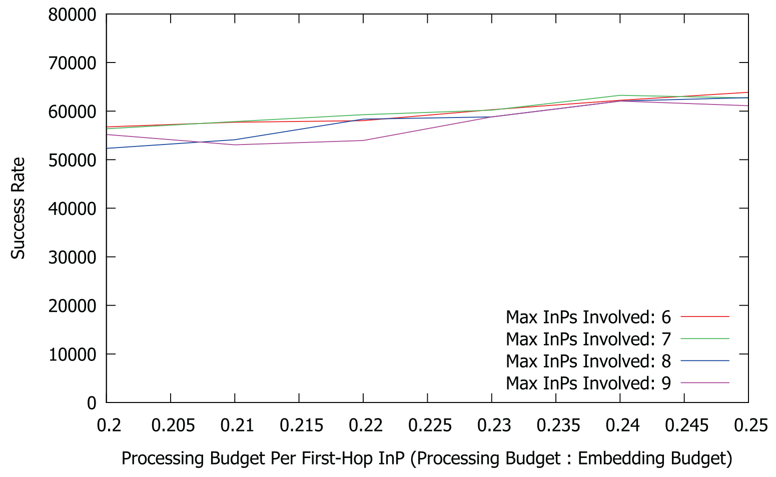


Figure 17 Total embedding cost with varying *d* and processing budgets.

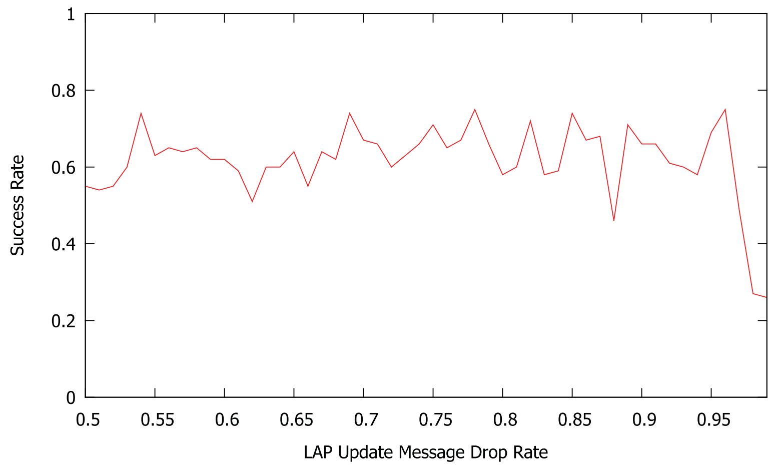


Figure 18 **Rate of successful flows with increasing LAP Update message drop rate.**

possible for an InP to abuse this model. For example, the d^{th} InP may be able to gouge prices, leaving the predecessor with no option but to accept the higher price. In the future, we will investigate alternative pricing models that will accomplish our primary goals while studying the strengths and weaknesses of each model.

7.2 Scalability

Scalability concerns in PolyViNE come from several fronts: size of the search space, dissemination time of location information, and storage of location and price information among others. As the number of InPs increases in the controller network, the amount of control traffic will increase even with the tweaks proposed in this paper. Moreover, the size of stored location and path information will grow very quickly with more and more InPs joining the controller network. We can limit the number of stored paths to a certain destination based on some heuristics (e.g., keep only the top M paths and flush the rest after each update), but such loss can result in degraded embedding. Finally, the freshness of the location information is dependent upon the update frequency and the total number of InPs in the controller network.

7.3 Performance

7.3.1 Response time

Recursive processes, by definition, can go on for a long time in the absence of proper terminating conditions resulting in unsuitable response times. Combining iterative mechanism wherever possible and limiting the level of recursion at the expense of search completeness can improve the response time of PolyViNE. However, the question regarding suitable response time depends on the arrival rate and the average life expectancy of VN requests.

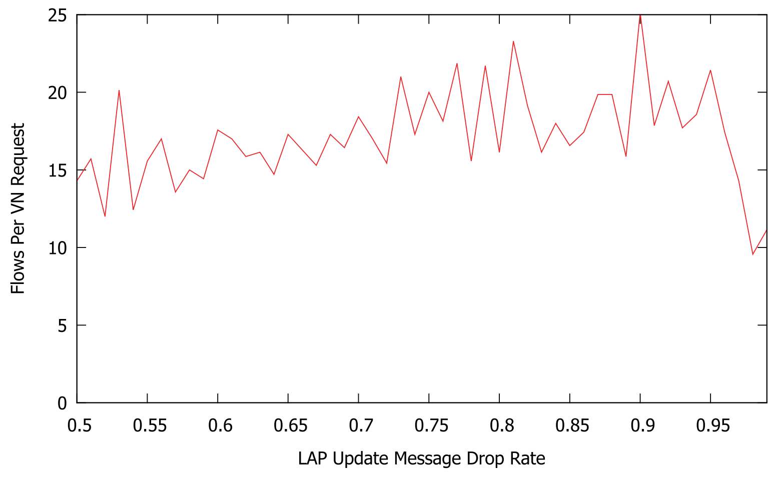


Figure 19 **Number of flows with increasing LAP Update message drop rate.**

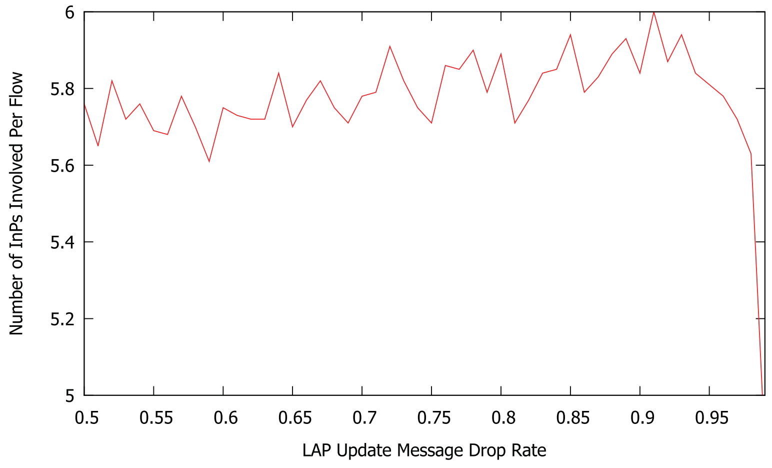


Figure 20 **InPs per Flow with increasing LAP Update message drop rate.**

7.3.2 Overheads

InPs participating in a PolyViNE embedding will face major computation overheads while trying to map the VN request and minor communication overheads due to relaying of the rest of the request. Since for each VN embedding every InP in each step except for the winning bidder will fail to take part in the embedding, the overheads can be discouraging. We are working toward finding incentives for the InPs to partake in the embedding process.

7.4 Trust and reputation

Since each InP will try to selfishly improve its own performance and will not expose its internal information, InPs can lie to or hide information from each other. From previous studies it is known that it is hard to use mechanism design or game theory to thwart such behaviors in a large scale distributed system [15]. Our solution against such behavior is the use of competitive bidding at each step of embedding to expose the market price of any leased resource.

7.5 More informed forwarding

PolyViNE currently uses LAP for informed forwarding of VN requests. However, location information is not the only information available to an InP about other InPs. An InP should be capable of "learning" from past experience. That is, it should be able to collect data on previous embeddings, and make more informed decisions in the future based on its observations of the past.

8 Related work

The VN embedding problem, with constraints on both virtual nodes and virtual links, is known to be $\mathcal{NP}$-hard

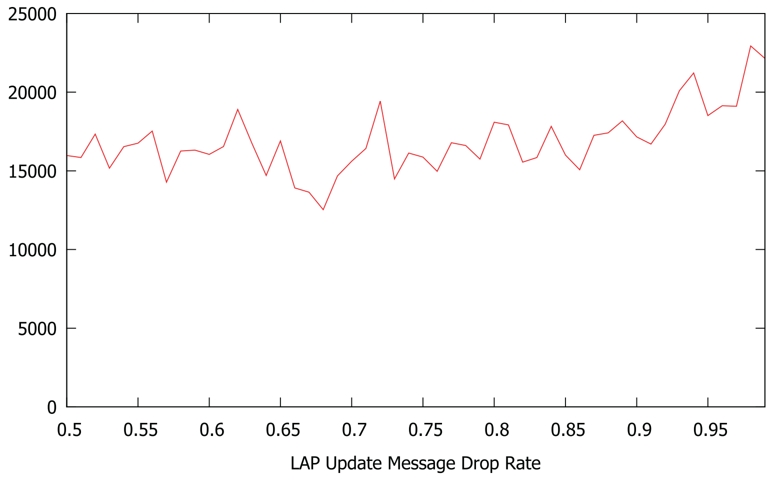


Figure 21 **Embedding cost with increasing LAP Update message drop rate.**

[5,6]. A number of heuristics have appeared in the literature based on the complete separation of the node mapping and the link mapping phases [5-7]. Existing research has also been restricting the problem space in different dimensions: [5,7] consider the offline version of the problem; [7] ignores node requirements; [5,7] assume infinite capacity in substrate nodes and links to obviate admission control; and [7] focuses on specific VN topologies. Chowdhury et al. [8] proposed a pair of algorithms that provide improved performance through increased correlation between the two phases of VN embedding, while [9] proposed a graph isomorphism-based integrated solution that can take exponential time in the worst case. All these algorithms address VN embedding as an intra-domain problem and take advantage of a centralized embedding entity.

Recently proposed V-Mart [16] framework approaches the inter-domain VN embedding problem using an auction-based model, where the SP performs the partitioning task using heuristics for simplification. As a result, V-Mart cannot enable local and inter-InP policy enforcement and fine-grained resource management.

Unlike inter-domain VN embedding, inter-domain lightpath provisioning [11,17] as well as cross-domain QoS-aware path composition [12,18] are well studied areas. UCLP [17] allows users to dynamically compose, modify, and tear down lightpaths across domain boundaries and over heterogeneous networking technologies (e.g., SONET/SDH, GMPLS etc.). Xiao et al. have shown in [18] that QoS-assured end-to-end path provisioning can be solved by reducing it to the classic k-MCOP (k-Multi Constrained Optimal Path) problem. iREX architecture [12], on the other hand, uses economic market-based mechanisms to automate inter-domain QoS policy enforcement through negotiation between participating domains. PolyViNE is similar to iREX in its allowance of intra-domain policy-enforcement and in using market-based mechanisms, but iREX is concerned about mapping simple paths whereas PolyViNE embeds more complicated VN requests. PeerMart [19] is another auction-based marketplace for resource trading in a network virtualization environment, but it basically deals only with virtual links.

The geographic location representation and related information dissemination protocol proposed in PolyViNE is inspired by the previous proposals of geographic addressing and routing in IPv6 networks [14,20] as well as the predominant global routing protocol in the Internet, BGP [21]. However, unlike these works, PolyViNE does not use the information for addressing or routing purposes; rather it uses the location information to find candidate InPs that will be able to embed part or whole of the remaining unmapped VN request. Moreover, such location information is disseminated between and stored in Controllers instead of border routers as in BGP or GIRO [14]. The concepts of Controllers in InPs and controller network connecting multiple InPs' Controllers are discussed in the iMark framework [10].

9 Conclusions and future work

In this paper we have formally defined the inter-domain VN embedding problem and presented PolyViNE – a novel policy-based inter-domain VN embedding framework – to address it. PolyViNE allows embedding of end-to-end VNs in a distributed and decentralized manner by promoting global competition in the presence of local autonomy. We have laid down the workflows of InPs and SPs throughout the PolyViNE embedding process and identified the most crucial stage in the InP workflow, VN request forwarding. In this respect, we have proposed a hierarchical addressing system (COST) and a location dissemination protocol (LAP) that jointly allow InPs to make informed forwarding decisions. We have also presented preliminary performance characteristics of PolyViNE through simulation.

In the future we would like to address issues such as pricing models, InP interactions, reputation management, and incentives for InP truthfulness. Relative advantages and disadvantages of contrasting choices (e.g., recursive vs iterative forwarding) in different stages of InP workflow should also be scrutinized. Finally, the scalability, stability, and performance characteristics of PolyViNE require further studies through larger simulations and distributed experiments with a heterogeneous mix of intra-domain VN embedding algorithms and policies.

Another interesting direction of research for this problem would be to model it as a distributed constrained optimization problem (DCOP) and to try to solve that with minimal information exchange between InPs.

10 Endnotes

[a]The words 'embedding', 'mapping', and 'assignment' are used interchangeably throughout this paper.
[b]We will use the terms InP and substrate network interchangeably throughout the rest of this paper.
[c]Each InP uses its own pricing mechanism by which it attaches a price to any embedding it provides.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors contributed to the idea of inter-domain virtual network embedding. MC and FS proposed and designed the required protocols and algorithms to perform such an embedding. FS wrote the simulator and carried out all the simulation experiments. All authors read and approved the final manuscript.

Acknowledgements
This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network, and in part by the World Class

University (WCU) Program under the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0).

Author details
[1]Google Canada Inc., Kitchener, ON, Canada. [2]Computer Science Division, University of California, Berkeley, CA, USA. [3]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada. [4]Division of IT Convergence Engineering, Pohang University of Science and Technology (POSTECH), Pohang 790-784, Korea.

Received: 30 November 2012 Accepted: 8 February 2013
Published: 11 March 2013

References

1. Anderson T, et al. (2005) Overcoming the Internet impasse through virtualization. Computer 38(4): 34–41
2. Turner J, Taylor D (2005) Diversifying the Internet. In IEEE GLOBECOM, St. Louis, MO, USA, pp. 1-6
3. Feamster N, et al. (2007) How to lease the Internet in your spare time. ACM SIGCOMM Computer Communication Review 37: 61–64
4. Chowdhury NMMK, Boutaba R (2010) A survey of network virtualization. Computer Networks 54(5): 862–876
5. Zhu Y, Ammar M (2006) Algorithms for assigning substrate network resources to virtual network components. In IEEE INFOCOM, Barcelona, Spain, pp 1–12
6. Yu M, et al. (2008) Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM CCR 38(2): 17–29
7. Lu J, Turner J (2006) Efficient mapping of virtual networks onto a shared substrate. Tech. Rep. WUCSE-2006-35. Washington University in St. Louis, pp 1-10
8. Chowdhury NMMK, et al. (2009) Virtual network embedding with coordinated node and link mapping. In IEEE INFOCOM, Rio de Janeiro, Brazil, pp 783–791
9. Lischka J, Karl H (2009) A virtual network mapping algorithm based on subgraph isomorphism detection. In ACM SIGCOMM VISA, New Delhi, India, pp 81–88
10. Chowdhury NMMK, et al. (2009) iMark: An identity management framework for network virtualization environment. In IEEE IM, New York, NY, USA, pp. 335–342
11. Liu Q, et al. (2007) Distributed inter-domain lightpath provisioning in the presence of wavelength conversion. Comput Commun 30(18): 3662–3675
12. Yahaya A, et al. (2008) iREX: Efficient automation architecture for the deployment of inter-domain QoS policy. IEEE TNSM 5: 50–64
13. Mekouar L, et al. (2010) Incorporating trust in network virtualization. In IEEE CIT, Bradford, UK, pp 942–947
14. Oliveira R, et al. (2007) Geographically informed inter-domain routing. In IEEE ICNP, Beijing, China, pp 103–112
15. Mahajan R, et al. (2004) Experiences applying game theory to system design. In SIGCOMM PINS, Portland, OR, USA, pp 183–190
16. Zaheer F, et al. (2010) Multi-provider service negotiation and contracting in network virtualization. In IEEE/IFIP NOMS, Osaka, Japan, pp 471–478
17. Boutaba R, et al. (2004) Lightpaths on demand: A Web-services-based management system. IEEE Communications Magazine 42(7): 101–107
18. Xiao J, Boutaba R (2005) QoS-aware service composition and adaptation in autonomic communication. IEEE JSAC 23(12): 2344–2360
19. Hausheer D, Stiller B (2007) Auctions for virtual network environments. In workshop on management of network virtualisation, Brussels, Belgium
20. Hain T (2006) Application and use of the IPv6 provider independent global unicast address format. Internet Draft. (http://tools.ietf.org/html/draft-hain-ipv6-pi-addr-10)
21. Rekhter Y, et al. (2006) A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard)

doi:10.1186/1869-0238-4-6
Cite this article as: Samuel *et al.*: **PolyViNE: policy-based virtual network embedding across multiple domains.** *Journal of Internet Services and Applications* 2013 **4**:6.