A General Distributed Approach to Slice Embedding with Guarantees

Flavio Esposito[§] Donato Di Paola [‡] Ibrahim Matta[§]
flavio@cs.bu.edu dipaola@ba.issia.cnr.it matta@cs.bu.edu
[§]Computer Science Department Boston University, MA
[‡] Institute of Intelligent Systems for Automation National Research Council (CNR), Bari, Italy

***Abstract*—To provide wide-area network services, resources from different infrastructure providers are needed. Leveraging the consensus-based resource allocation literature, we propose a general distributed auction mechanism for the (NP-hard) slice embedding problem. Under reasonable assumptions on the bidding scheme, the proposed mechanism is proven to converge, and it is shown that the solutions guarantee a worst-case efficiency of $(1 - \frac{1}{e})$ relative to the optimal solution. Using extensive simulations, we confirm superior convergence properties and resource utilization when compared with existing distributed slice embedding solutions, and we show how by appropriate policy design, our mechanism can be instantiated to accommodate the embedding goals of different service and infrastructure providers, resulting in an attractive and flexible resource allocation solution for network virtualization.**

I. Introduction

The challenge of deploying wide-area virtualization based network services recently spurred interest in both the business and the research communities: from a research perspective, this enables the networking communities to concurrently experiment with new Internet architectures and protocols, each running on an isolated instance of the physical network. From a market perspective, this paradigm is appealing as it enables multiple infrastructure and service providers (InPs and SPs) to experiment with new business models that range from leasing their infrastructure to hosting multiple concurrent network services.

A slice (or virtual network) is a set of virtual instances spanning a set of physical resources, $e.g.$ processes and physical links, and by network service we mean the commodity supplied by the slice, $e.g.$ an online game or the access to a distributed virtual network testbed. Examples of service providers are content delivery networks, high performance computing systems such as cluster-on-demand, or large-scale distributed testbed platforms ($e.g.$ Emulab [23], GENI [2].) InPs may cooperate or compete to provide such services themselves, or they could lease their resources to an SP. We consider a model in which a set of InPs receive a slice request from an SP (or an intermediary "connectivity" provider [26]), and try to embed it in a distributed fashion.

The slice embedding problem [1] consists of three tasks: (1) resource discovery, which involves monitoring the state of the physical resources, (2) virtual network mapping, which involves matching users' requests to the available resources, and (3) allocation, which involves assigning the resources that match the users' requests. These three tasks are tightly coupled, and although there exists a wide spectrum of solutions that solve a particular task, at most two tasks along with their interactions have been considered (see Section II or [8] for a complete survey.)

Distributed virtual network mapping solutions that allow different InPs to collectively embed a slice already exist [5], [12], [25]: some of them focus on the desirable property of letting InPs use their own (embedding) policies [5], while others rely on truthfulness of a virtual resource auction [25]. Although they have systematic logic behind their design, such distributed solutions are still restricted to a subset of the three slice embedding tasks, they have performance ($e.g.$ convergence speed or resource utilization) tightly determined by the chosen heuristic, and they are limited to a single distribution model — the type and amount of information propagated to embed a slice.

Existing embedding solutions are also restrictive with respect to slice's arrival rate and duration: the lifetime of a slice can range from few seconds or minutes (in the case of cluster-on-demand services) to several months and years (in the case of a slice hosting a content distribution service similar to Akamai [20], or a GENI [2] slice hosting a novel architecture looking for new adopters to opt-in.) For instance, in wide-area testbed applications, slices are provided in a best-effort manner, and the inter-arrival time between slice requests and the lifetime of slices are typically much longer than the slice embedding time, so existing solutions assume complete knowledge of the network state, and ignore the overhead of resource discovery and the slice embedding time. In applications with higher churns, $e.g.$, cluster-on-demand such as financial modeling, anomaly analysis, or heavy image processing, where slice providers have rigid Service Level Objectives (SLO) — the technical requirements within a Service Level Agreement (SLA) — or where slices have short lifetime and request short response time, it is desirable that solutions attempt to reduce the slice embedding time, and employ limited resource discovery to reduce overhead.

In summary, due to the wide range of providers' goals and allocation models ($e.g.$, best effort or SLO), a flexible solution that is adaptable to different provider goals and tackles the distributed slice embedding with its three phases does not yet exist. Moreover, none of the previously proposed solutions give guarantees on both the convergence of the slice embedding process, and on allocation performance — ratio of the number of slices successfully allocated on the physical

[1]The term "slice embedding" was coined in [10]. An alternative term is "virtual network provisioning" [11].

infrastructure to the total requested.

To this end, leveraging properties from the consensus literature [17], we propose a general Consensus-based Auction mechanism for Distributed slice embedding (CAD). The mechanism is general as it supports a large spectrum of applications and providers' objectives along with their distribution models by tuning its policies. CAD iterates over a bidding and an agreement (or consensus) phase to embed virtual nodes, before a third phase embeds virtual links. By only exchanging bids and few other policy-driven information with their neighbors, physical nodes discover available resources, find a mapping solution and agree on a slice assignment.

To demonstrate its flexibility, we compare and analyze the tradeoffs between two different policy configurations of CAD (Section III): the first, that we call *Single Allocation Distributed* slice embedding (SAD), allows bidding on a single virtual node per auction round. The second, called *Multiple Allocation Distributed* slice embedding (MAD), allows bidder physical nodes to win multiple virtual resources simultaneously and therefore leads to faster slice embedding (convergence) time. Using extensive trace-driven simulations, we show the counter-intuitive result that having full knowledge of the entire slice to be allocated before bidding, MAD may yield lower allocation efficiency. Moreover, we show that SAD better balances the load and often has shorter response time — time to identify whether a slice can be embedded — independently from the slice (virtual) topology (Section V.)

It is known that distributed auctions converge to a solution if the bidding function is sub-modular [4], [15]. We obtain the same convergence result relaxing the sub-modularity assumption and using the notion of *pseudo-submodularity* of the utility function that physical nodes use to bid, that is, each physical node is free to use *any* private bidding function for each auction round, and communicates its bids in a way so that they *appear* to be obtained using a sub-modular function. We show that independently from the bidding policy that InPs decide to adopt, CAD has a worst-case convergence time of $D \cdot |V_H|$, where D is the diameter of the physical network and $|V_H|$ the size of the slice H to be embedded (Section IV.) Under the same assumptions, we also show that CAD has a minimum performance guarantee of $(1 - e^{-1})$ relative to the optimal solution.

II. Related work

Distributed slice embedding: to avoid restricting services within a limited single provider's domain, distributed solutions to the slice embedding have been proposed. Some solutions rely on a centralized authority that partitions the slice and orchestrates the mapping [11], [26], while others do not require such orchestration and hence we classify them as fully distributed [12]. The only (to the best of our knowledge) fully distributed embedding solution existing today [12] has discouraging discovery overhead as each mapping information is flooded to all physical nodes. The resource discovery phase is different in PolyViNE [5], where an SP sends the entire slice to a subset of trusted InPs, which can eventually map the slice partially, and forwards the residual virtual subgraph to another set of trusted InPs. The process continues and the slice is rejected if a threshold number of hops is reached before its mapping is complete. The SP does the final allocation, based on the best price among the multiple candidate mapping solutions returned by different sets of InPs. The mapping and the allocation depend on the discovery, that is, on the sequence of visited InPs and therefore the proposed heuristic in practice lead to heavy sub-optimalities or to significant overhead (in case the residual virtual network is flooded to all remaining InPs.)

Our mechanism also supports slice splitting and centralized embedding orchestration, but its bidding mechanism (thanks to the max-consensus strategy) provides a complete resource discovery relying on low overhead nearest-neighbor communications, and furthermore allocation is concurrently done.

Auctions and guarantees: the idea of using auctions for a distributed slice allocation has been floated before: V-Mart [25] ensures a fair market but its auction winner determination algorithm does not guarantee that the sum of provider utilities is maximized. Auction algorithms and their optimality performance have also been theoretically studied in several application domains [3]. In the electronic commerce for example [6], truthful auction strategies are sought when multiple items are released by a centralized auctioneer, and guarantees on an equilibrium are proven to exist [16]. Our approach does not need a centralized auctioneer, and we also prove bounds on the number of iterations to reach an equilibrium (convergence to an embedding), as a function of the physical network diameter, and the size of the slice to allocate. Moreover, in our settings truthful strategies may not work as there is uncertainty on whether more slices, or even more virtual nodes in the same slice, are to be assigned in the future; bidders may have incentives to preserve resources for stronger future bids.

In different settings, Choi *et al.* [4] present a decentralized auction that greedily assigns tasks to a fleet of robots. Our problem formulation allocates virtual nodes and links, and physical nodes do not move as robots do.

III. Consensus-based Auctions for Distributed Slice Embedding

Problem statement. Given a virtual network $H = (V_H, E_H, C_H)$ and a physical network $G = (V_G, E_G, C_G)$, where V is a set of nodes, E is a set of links, and each node or link $e \in V \cup E$ is associated with a capacity constraint $C(e)$, [2] a virtual network (slice) mapping (or embedding) is a mapping of H onto a subset of G, such that each virtual node is mapped onto exactly one physical node, and each virtual link is mapped onto a loop-free physical path p. Formally, the mapping is a function $\mathcal{M} : H \rightarrow (V_G, \mathcal{P})$ where $\mathcal{P}$ denotes the set of all loop-free paths in G. $\mathcal{M}$ is called a *valid mapping* if all constraints of H are satisfied, and for

[2]Each $C(e)$ could be a vector $\{C_1(e), \ldots, C_\gamma(e)\}$ containing different types of constraints, *e.g.* physical geo-location, delay or jitter.

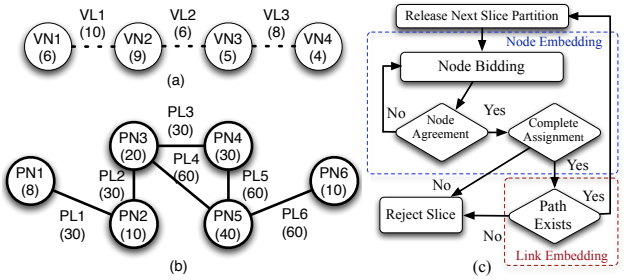

Fig. 1. (a) Slice with capacity constraints to be embedded. (b) Each physical node (PN) can be owned by a different InP, and can have a different capacity. (c) CAD workflow: a virtual link embedding phase follows the virtual node bidding and agreement phases.

each $l^H = (s^H, r^H) \in E_H$, $\exists$ at least one physical loop-free path $p : (s^G, \ldots, r^G) \in \mathcal{P}$ where s^H is mapped to s^G and r^H is mapped to r^G.

Objective: multiple valid mappings of H over G may exist; each physical node i has a utility function U_i. We are interested in finding in a distributed fashion the embedding solution that maximizes the sum of the utilities of all providers $\sum_{i \in V_G} U_i$, $e.g.$, by letting InPs instantiate policies according to their goals and run the auction. A natural objective for an embedding algorithm is to maximize *revenue*. The revenue can be defined in various ways according to economic models. As in [24], we use the notion of a higher economic benefit (reward) from accepting a slice or virtual request that requires more resources ($e.g.$, bandwidth, CPU) from the physical network.

CAD mechanism: consider a slice embedding request by an SP (Figure 1a) on a physical network (Figure 1b) where each physical node (PN) belongs to a different InP. The SP sends to (a subset of) all physical nodes a request with (a subset of) the virtual elements (nodes and links), $e.g.$ virtual nodes VN1 and VN2 connected by virtual link VL1. Each physical node i, where $i \in V_G$, uses a private utility function $\mathbf{U}_i \in \mathbb{R}_+^{|V_H|}$ to bid on the virtual nodes, knowing that it could be the winner of a subset (for example VN1 or VN2 or both), and stores its bids in a vector $\mathbf{b}_i \in \mathbb{R}_+^{|V_H|}$. Each entry $b_{ij} \in \mathbf{b}_i$ is a positive real number representing the highest bid known so far on virtual node $j \in V_H$. Also, physical nodes store the identifiers of the virtual nodes on which they are bidding in a list (bundle vector) $\mathbf{m}_i \in V_H^{T_i}$, where T_i is a target number of virtual nodes mappable on i. After the private bidding phase, each physical node exchanges the bids with its neighbors, updating an assignment vector $\mathbf{a}_i \in V_G^{|V_H|}$ with the latest information on the current assignment of all virtual nodes, for a distributed auction winner determination.

The winner physical nodes communicate the mapping to the SP which, if possible, releases the next slice(s) or the next slice partition if any ($e.g.$ VN3, VN4, VL3 in Figure 1a.) [3]

[3]The slice partitioning problem has been shown to be NP-hard, $e.g$ in [11] and it is outside the scope of this paper.

Once the physical nodes have reached consensus on who is the winner for all the virtual nodes of the (partial or full) slice released for the auction, a distributed link embedding phase is run to embed each virtual link on a set of (one or many) loop-free physical paths (Figure 1c.) The mechanism iterates over multiple node bidding and agreement (consensus) phases synchronously, that is, the second bidding phase does not start until the first agreement phase terminates. Physical nodes act upon messages received at different times during each bidding phase and each consensus phase; therefore, each individual phase is asynchronous. In the rest of the paper, we denote such *rounds or iterations* of node bidding followed by consensus with the letter t and we omit t when it is clear from the context.

Adapting the definition of *max-consensus* from the consensus literature [17] to the slice embedding problem we have:

Definition 1: (*max-consensus.*) Given a physical network G, an initial bid vector of physical nodes $\mathbf{b}(0) \triangleq (\mathbf{b}_1(0), \ldots, \mathbf{b}_{|V_G|}(0))^{\mathrm{T}}$, a set of neighbors $\mathcal{N}_i$ $\forall i \in V_G$, and the consensus algorithm for the communication instance $t+1$:

$$\mathbf{b}_i(t+1) = \max_{j \in \mathcal{N}_i \cup \{i\}} \{\mathbf{b}_j(t)\} \ \ \forall i \in V_G, \tag{1}$$

Max-consensus on the bids among the physical nodes is said to be achieved with convergence time l, if $\exists\ l \in \mathbb{N}$ such that $\forall\ t \geq l$ and $\forall\ i, i' \in V_G$,

$$\mathbf{b}_i(t) = \mathbf{b}_{i'}(t) = \max\{\mathbf{b}_1(0), \ldots, \mathbf{b}_{|V_G|}(0)\}, \tag{2}$$

where $\max\{\cdot\}$ is the component-wise maximum.

Assumptions: we assume that physical nodes are aware of the physical outgoing link capacity to reach each of its first-hop neighbors to propagate the highest bids, the routing table for the path embedding phase, and the diameter D of the physical network, useful as a termination condition: if a physical node has received more than D messages the auction phase terminates. [4]

CAD Policies: one of the design goals of CAD is its flexibility — ability to create customizable slice embedding algorithms to satisfy desired policies, rules, and conditions. We describe here such policies, and later in this section we show few examples of how they can be instantiated to satisfy other goals. A straightforward example of policy is the (normalized) utility function $\mathbf{U}$ that InPs use to bid on virtual resources (nodes). In our evaluation (Section V), the bid value of physical node i on virtual node j is equivalent to U_{ij}, where:

$$\mathbb{T}_i = C_i + \sum_{k \in \mathcal{N}_i} C_{ik}, \qquad U_{ij} = \frac{\mathbb{T}_i - S_{ij}}{\mathbb{T}_i} \tag{3}$$

where $\mathbb{T}_i$ is the target virtual (node and links) capacity that is allocatable on i, and S_{ij} the stress on physical node i, namely, the sum of the virtual node capacity already allocated on i, including virtual node j on which i is bidding, plus the capacity of the virtual links allocated on the adjacent physical

[4]Algorithms to compute the diameter of a network in a distributed way are well known [17], and they are outside the scope of this paper.

links. Note that, due to the max consensus definition, the bid b_{ij} at physical node i on virtual node j is the maximum utility value seen so far. The normalization factor $\frac{1}{\mathbb{T}_i}$ ensures that such bids are comparable across physical nodes.

We have seen from related work, $e.g.$ [12], [26], how embedding protocols may require SPs to split the slice. CAD is able to express this requirement by enforcing a limit on the length of the bid vector $\mathbf{b}_i$, so that physical nodes bid only on the released slice partition. Each InP can also enforce a load target on its resources by limiting its target allocatable capacity $\mathbb{T}_i$, which, in turn, limits its bundle size T_i.

Another auction policy is the *assignment vector* $\mathbf{a}_i$, that is, a vector that keeps track of the current assignment of virtual nodes. $\mathbf{a}_i$ may assume two forms: *least* and *most informative*. In its least informative form, $\mathbf{a}_i \equiv \mathbf{x}_i$ is a binary vector where x_{ij} is equal to one if physical node i hosts virtual node j and 0 otherwise. In its most informative form, $\mathbf{a}_i \equiv \mathbf{w}_i$ is a vector of physical nodes that are far winning the hosting of virtual nodes; w_{ij} represents the identifier of the physical node that made the highest bid so far to host virtual node j. Note that when $\mathbf{a}_i \equiv \mathbf{w}_i$ the assignment vector reveals information on which physical nodes are so far the winners of the auction, whereas if $\mathbf{a}_i \equiv \mathbf{x}_i$ physical node i only knows if it is winning each virtual node or not. As a direct consequence of the max-consensus, this implies that when the assignment (allocation) vector is in its least informative form, each physical node only knows the value of the maximum bid so far without knowing the identity of the bidder. We also leave as a policy whether the *assignment vector* is exchanged with the neighbors or not. In case all physical nodes know about the assignment vector of the virtual nodes, such information may be used to allocate virtual links in a distributed fashion. Instead, if $\mathbf{a}_i \equiv \mathbf{x}_i$, to avoid physical nodes flooding their assignment information, i asks the SP about the identity of the physical node hosting the other end of the virtual link and attempts to allocate at least one loop-free physical path.

A. Phase 1: CAD Bidding (Auction) Phase

Consider procedure 1: after the initialization of the assignment vector $\mathbf{a}_i$, the bid vector $\mathbf{b}_i$ and the bundle vector $\mathbf{m}_i$ for the current iteration t (line 3) [5], each physical node checks if another bidding phase is needed (line 4), for example because there is enough capacity or because the auction policy allows another bidding, or else terminates. If a physical node can bid, but cannot outbid any virtual node, the bidding phase terminates. If instead there is at least one biddable virtual node j $i.e.$ if $U_{ij}(t) > b_{ij}$ (line 5), [6] physical node i registers in its bid vector the bid with the highest reward $\eta = \underset{j \in V_H}{\operatorname{argmax}}\{h_{ij} \cdot U_{ij}\}$ (line 6) and updates the state vectors (lines $7-9$.) At the end of the bidding phase, the current winning bid vector (line 10) and if the auction policy allows it (lines $11-13$), the assignment vector $\mathbf{a}_i$ are exchanged with each neighbor. Depending on the configured policies,

[5] We elaborate on the need to reset $\mathbf{m}_i$ at the end of Remark 2, Section III-C.

[6] $\mathbb{I}(\cdot)$ is an indicator function, unitary if the argument is true and 0 otherwise.

Procedure 1 CAD `biddingPhase` for physical node i at iteration t

1: **Input:** $\mathbf{a}_i(t-1)$, $\mathbf{b}_i(t-1)$
2: **Output:** $\mathbf{a}_i(t)$, $\mathbf{b}_i(t)$, $\mathbf{m}_i(t)$
3: $\mathbf{a}_i(t) = \mathbf{a}_i(t-1)$, $\mathbf{b}_i(t) = \mathbf{b}_i(t-1), \mathbf{m}_i(t) = \emptyset$
4: **if** `biddingIsNeeded(`$\mathbf{a}_i(t), \mathbb{T}_i$`)` **then**
5: **if** $\exists\ j : h_{ij} = \mathbb{I}(U_{ij}(t) > b_{ij}(t)) == 1$ **then**
6: $\eta = \text{argmax}_{j \in V_H}\{h_{ij} \cdot U_{ij}\}$
7: $\mathbf{m}_i(t) = \mathbf{m}_i(t) \oplus \eta$ // append η to bundle
8: $b_{i\eta}(t) = U_{i\eta}(t)$
9: `update(`$\eta, \mathbf{a}_i(t)$`)`
10: `Send / Receive` $\mathbf{b}_i$ `to / from` $k\ \ \forall k \in \mathcal{N}_i$
11: **if** $\mathbf{a}_i \equiv \mathbf{w}_i$ **then**
12: `Send / Receive` $\mathbf{w}_i$ `to / from` $k\ \ \forall k \in \mathcal{N}_i$
13: **end if**
14: **end if**
15: **end if**

the functions `biddingIsNeeded()` and `update()` of Procedure 1 may behave differently.

SAD configuration: in particular, let us consider a scenario in which InPs (1) wish to reveal the least possible information to other (competitor) InPs, and (2) they are interested in the quickest possible response time on a slice request. To accommodate these goals, we set the assignment vector policy to its least informative form, the partition size to two (so that a slice is rejected as soon as one of the two virtual nodes or their adjacent virtual link is not allocatable), and the bundle vector size to one, so that the auction is on a single item. As we are forcing physical nodes to bid on a *single* virtual node per auction round, we refer in the rest of the paper to this policy configuration as Single Allocation for Distributed slice embedding (SAD).

SAD bidding: given such policy configuration, the `biddingIsNeeded()` function can be implemented by only verifying if $A(t) = \sum_{j \in V_H} x_{ij}(t) = 0$, knowing that bidders are only allowed to win one virtual node per round "t", that is, $A(t) \leq 1$. Given the SAD policy configuration, the `update()` function implementation simply changes the assignment vector from $x_{i\eta}(t) = 0$ to $x_{i\eta}(t) = 1$.

Example 1: (SAD bidding.) Consider Figure 1: virtual nodes VN1 and VN2 are released by the SP. Assuming that all nodes use as utility their residual node capacity, PN1, PN3 and PN5's initial bidding vectors are $\mathbf{b}_{PN1}(0) = (8, 0)$, $\mathbf{b}_{PN3}(0) = (0, 20)$, and $\mathbf{b}_{PN5}(0) = (0, 40)$. Note that the first bid of each physical node is its initial capacity, and PN1 could not bid on VN2 since VN2 requires 9 capacity units whereas PN1's capacity is 8. Also we assume that a physical node, whenever feasible, bids on the virtual node with highest residual capacity as this brings higher reward (revenue.) In their first bidding phase, physical nodes assign themselves as winners for the virtual nodes as they do not know yet each other's bids, and so $\mathbf{x}_{PN1} = (1, 0)$ and $\mathbf{x}_{PN3} = \mathbf{x}_{PN5} = (0, 1)$.

MAD configuration: let us now consider a scenario in which embedding slices with the least possible auction iterations

(convergence time) is more desirable than hiding information from other physical nodes. To this end, we remove the limit on the number of biddable virtual nodes within the same auction round, and we do not partition the slice so that each physical node has an offline knowledge of the entire slice (as opposed to SAD that releases the slice components in an online fashion, $i.e.$ the slice embedding algorithm runs without a complete knowledge of the input.) Moreover, we set the assignment vector policy to its most informative form, so that the consensus is run simultaneously on both the bid vector and on the assignment vector.

MAD bidding: under these settings, the function `biddingIsNeeded()` is implemented so that it returns true while there is still room for additional virtual resources. The amount of virtual resources that physical node i is attempting to host can be expressed either in terms of the total number of virtual nodes in its current bundle $\mathbf{m_i(t)}$, $i.e.$ $|\mathbf{m_i(t)}|$, or in terms of the resulting virtual capacity stress on physical node i as in (3). Also under these settings, the `update()` function implementation updates the allocation vector with $w_{i,\eta}(t) = i$ (not just with 1 or 0 but with the identifier of the winning physical node.)

Example 2: (MAD bidding.) Let us consider Figure 1 and let us assume that the target allocated capacity of PN3 is 16 units, and that the requested virtual capacity is equivalent to the reward that a physical node gets if it wins the hosting of that virtual node. In this example, let us also assume that physical node bids are equivalent to their residual physical capacity, $e.g.$, a physical node with residual capacity 10 units bids 10 to attempt the hosting of a virtual node whose requested capacity is no higher than 10 units. Let us apply MAD to construct the bundle of PN3. First PN3 bids on VN2, as it is the virtual node with the highest requested capacity (reward) and so $\mathbf{b}_{PN3} = (0, 20, 0, 0)$. After filling its bundle with VN2, PN3 updates its residual capacity from 20 to 11 (as VN2 requested capacity is 9). The next virtual node to be inserted in the bundle is hence VN1, as it has the highest requested capacity among the remaining virtual nodes. PN3 bidding phase terminates with $\mathbf{b}_{PN3} = (11, 20, 0, 0)$, $\mathbf{w}_{PN3} = (PN3, PN3, -, -)$ and bundle $\mathbf{m}_{PN3} = (VN2, VN1)$, as embedding more virtual nodes would increase the allocated capacity beyond the target.

B. Phase 2: CAD Agreement Phase

In this phase, physical nodes make use of a maximum consensus strategy to converge on the winning bids $\bar{\mathbf{b}}$, and to compute the allocation vector $\bar{\mathbf{a}}$ (Procedure 2.)

The consensus, for example on the bid vector $\mathbf{b}_i$ after receiving the bids from each physical node k in i's neighborhood $\mathcal{N}_i$, is performed by comparing the bid b_{ij} with b_{kj} for all k members of $\mathcal{N}_i$. This evaluation is performed by the function `IsUpdated()` (line 5.) In case the auction requires consensus only on a single virtual node at a time, $i.e.$ $|\mathbf{m}_i| = 1$ as in SAD, the function `IsUpdated()` merely checks if there is a higher bid, that is, if $\exists\, k, j : b_{kj} > b_{ij}$. This means that when a physical node i receives from a neighboring physical node k a higher bid for a virtual node j, the receiver i always updates its bid vector $\mathbf{b}_i$ ($b_{ij} \leftarrow b_{kj}$), no matter when the higher bid was generated. In general, $i.e.$, when $|\mathbf{m}_i| > 1$, physical nodes may receive higher bids that are out of date. We discuss the conflict resolution of CAD in Section III-C.

Procedure 2 CAD `agreementPhase` for physical node i at iteration t

1: **Input:** $\mathbf{a}_i(t)$, $\mathbf{b}_i(t)$, $\mathbf{m}_i(t)$
2: **Output:** $\mathbf{a}_i(t)$, $\mathbf{b}_i(t)$, $\mathbf{m}_i(t)$
3: **for all** $k \in \mathcal{N}_i$ **do**
4: **for all** $j \in V_H$ **do**
5: **if** `IsUpdated`(b_{kj}) **then**
6: `update`$(\mathbf{b}_i(t), \mathbf{a}_i(t), \mathbf{m}_i(t))$
7: **end if**
8: **end for**
9: **end for**

Example 3: (SAD consensus.) We have assumed that hosting higher capacity virtual nodes bring higher revenue, and so continuing Example 1, after exchanging its bid vector with PN5, PN3 updates $\mathbf{b}_{PN3}$ from $(0, 20)$ to $(0, 40)$, and $\mathbf{x}_{PN3}$ from $(0, 1)$ to $(0, 0)$. Having lost the auction for node VN2 (the most profitable virtual node) to PN5, PN3 bids on VN1, and so updates again its bid vector from $\mathbf{b}_{PN3} = (0, 40)$ to $(20, 40)$, as all PN3's capacity can now be used for VN1 and PN5's bid on VN2 is recorded. PN3 also changes its allocation vector again from $\mathbf{x}_{PN3} = (0, 0)$ to $(1, 0)$. Eventually, all physical nodes agree that PN5's bid is the highest for the most profitable virtual node VN2, while PN4 wins VN1 as it has the highest residual capacity after VN2 assignment.

When instead physical nodes are allowed to bid on multiple virtual nodes in the same auction round ($|\mathbf{m}_i| > 1$) as in MAD, even if the received bid for a virtual node is higher than what is currently known, the information received may not be up-to-date. In other words, the standard max-consensus strategy may not work. Each physical node is required to evaluate the function `IsUpdated()`. In particular, `IsUpdated()` compares the time-stamps of the received bid vector, and updates the bundle, the bid and the assignment vector accordingly (Procedure 2, line 6.) Intuitively, a physical node loses its assignment on a virtual node j if it gets outbid by another physical node that has a more recent bid, or after realizing that its bid for j was subsequent to another previous bid that it had lost more recently.

More precisely, in CAD bids on a physical node for the same virtual node are required to be lower if more virtual nodes are previously allocated. This is obvious in our examples, as to bid, a physical node uses its residual capacity that decreases as more virtual nodes are added to the bundle — as we show later, this monotonically non-increasing condition must hold for any other utility function. This means that if a physical node i is outbid on a virtual node j, all the subsequent nodes $\mathbf{m}_{ij'}$ $\forall j' > j$ were computed using an invalid value and therefore need to be released, that is, $\mathbf{b}_{ij'} = 0$ $\forall j' > j$.

C. Conflicts resolution

When it receives a bid update, physical node i has three options: (i) `ignore` the received bid leaving its bid vector and its allocation vector as they are, (ii) `update` according to the information received, *i.e.* $w_{ij} = w_{kj}$ and $b_{ij} = b_{kj}$, or (iii) `reset`, *i.e.* $w_{ij} = \emptyset$ and $b_{ij} = 0$. When $|\mathbf{m}_i| > 1$, the bids alone are not enough to determine the auction winner as virtual nodes can be released, and a physical node i does not know if the bid received has been released or is outdated.

We conclude this subsection with two remarks that explore how such conflicts are resolved. In particular, we illustrate how bids should be ignored or reset if they are outdated, and how subsequent bids to a more recently lost bid should be released.

***Remark** 1: (bids may be ignored or reset.)* There are cases in which the bid values are not enough to resolve conflicts, and so the time-stamps at which the bid was generated are used to resolve conflicts. In particular, (1) if a sender physical node i thinks that a receiver k is the winner and k thinks the winner is $n \neq \{i, k\}$, or (2) when i thinks n is the winner and k thinks the winner is $m \neq \{n, i, k\}$, or when (3) both i and k think m is winning but with a different bid. In all these cases, knowing which bid is most recent allows k to either ignore or update its bid based on the bid from i. In other cases, even the time-stamps are not enough and i and k need to reset their bids. In particular, (4) when i thinks the winner is k and k thinks the winner is i. In this case, even if i's bid were more recently generated, it might have been generated before k's bid were *received* by i.

***Remark** 2: (releasing subsequent bids.)* Given PN3's bidding phase in Example 2, and computing PN5's vectors we have: $\mathbf{m}_{PN5} = (VN2, VN1, VN3, VN4)$, $\mathbf{b}_{PN5} = (31, 40, 25, 20)$ and $\mathbf{w}_{PN5} = (PN5, PN5, PN5, PN5)$. After receiving the bids from PN5, PN3 realizes that its first bundle's entry is outbid ($20 < 40$) and so it must release VN2. Therefore PN3 needs to also release the other subsequent node in its bundle VN1, as its bid value was a function of the bid on VN2, $i.e.$ the bid on VN1 assumed the residual capacity after VN2 is allocated on PN3.

Since CAD allows physical nodes to bid using their most updated residual capacity, releasing subsequent items from a bundle intuitively improves the sum of the utilities of the physical nodes and hence, when physical nodes cooperate, this improves the number of slices allocated. Moreover, as we show in Section IV-A, such residual capacity utility guarantees convergence to a slice embedding. Note also that, due to the slice topology constraints, a change of assignment of any virtual node not present in a bundle may invalidate all its bids. Assume, for example (Figure 1), that PN5 is winning VN2 when PN3 bids on VN1. The bid on VN1 may change if the connected VN2 is later hosted by another physical node, $e.g.$ PN4, as the residual physical link capacity to connect physical nodes PN3 and PN4 may be smaller than the residual capacity of the physical link connecting PN3 and PN5. In extreme cases, the residual capacity of the physical link PN3-PN4 can be even null, not allowing the embedding of the slice at all. To avoid storing bids computed with an out-of-date utility value, physical nodes simply reset their own bundle at the beginning of every bidding phase (procedure 1, line 3.)

D. Pseudo sub-modular utility functions

As we will see in Section IV, our CAD mechanism guarantees convergence allowing InPs to use their own bidding policies, as long as the function appears to be sub-modular to other bidders [14]. Sub-modularity is a well studied concept in mathematics [19], and applied to the distributed slice embedding problem, can be defined as follows:

Definition 2: (*sub-modular function.*) The marginal utility function $U(j, \mathbf{m})$ obtained by adding a virtual resource j to an existing bundle $\mathbf{m}$, is sub-modular if and only if

$$U(j, \mathbf{m}') \geq U(j, \mathbf{m}) \; \forall \; \mathbf{m}' \mid \mathbf{m}' \subset \mathbf{m}. \tag{4}$$

This means that if a physical node uses a sub-modular utility function, a value of a particular virtual resource j cannot increase because of the presence of other resources in the bundle.

Although having sub-modular utility functions may be realistic in many resource allocation problems [15], in the distributed slice embedding problem this assumption may be too restrictive, as the value of a virtual node may increase as new resources are added to the bundle, $e.g.$ the cost of mapping a virtual link between two virtual nodes decreases if a physical node hosts both virtual source and destination. To guarantee convergence without using a sub-modular score function, as in [14], we let each physical node communicate its bid on virtual node j obtained from a bid warping function:

$$\mathcal{W}_{ij}(U_{ij}, \mathbf{b}_i) = \min_{k \in \{1, \ldots, |\mathbf{b}_i|\}} \{U_{ij}, \mathcal{W}_{ik}\} \tag{5}$$

where $\mathcal{W}_{ik}$ is the value of the warping function for the k^{th} element of $\mathbf{b}_i$. Note how by definition, applying the function $\mathcal{W}$ to the bid before sending it is equivalent to communicating a bid that is never higher than any previously communicated bids. In other words, bids *appear* to other physical nodes to be obtained from a sub-modular utility function.

E. Phase 3: Virtual Link Embedding

Similar to the bidding and agreement phases for virtual nodes, in the virtual link embedding phase, our CAD mechanisms allow applications and provider's goals to tune the slice embedding protocol behavior through policy instantiation.

This last phase is based on the observation that all virtual link embedding schemes have two commonalities: information known at each physical node about physical paths, and the algorithm for determining the best physical path(s) to allocate a virtual link. We hence define three CAD policies for virtual link embedding: (i) the type of information known at each physical node, for example the routing table or the available paths for any source-destination, (ii) the update frequency of such information, for example every hour or every time a new slice is requested, and (iii) the selection of physical path(s) over which a virtual link is mapped. One example of such

virtual link embedding scheme is a simple SP assisted auction, where, similarly to [25] and [13], an SP elicits bids from each InP, computes the "cheapest" loop-free physical path according to the bids, and then allocates the virtual link on that path. As shown in [24], another effective example is a k-shortest path algorithm with path splitting [7].

In our experiments we let physical nodes know the routing table, computed only once at the beginning of our experiments using Dijkstra's algorithm, and we also use the k-shortest (hop distance) path algorithm with $k = 3$. This virtual link (path) embedding policy has the limitation of forcing intermediate physical nodes on a path to accept the allocation of a virtual link if they have capacity. We leave for future work the exploration of other strategies (for example path bidding [13].)

IV. Convergence and Performance Guarantees

In this section we show results on the convergence properties of CAD. By convergence we mean that a valid mapping (Section III) is found in a finite number of steps (Definition 1.) Moreover, leveraging well-known results on sub-modular functions [9], [19], we show that under the assumption of pseudo sub-modularity (Section III-D) of the utility function, CAD guarantees a $(1 - \frac{1}{e})$ optimal approximation. [7]

A. Convergence Analysis

All physical nodes need to be aware of the mapping, by exchanging their bids with only their first-hop neighbors, therefore a change of bid information needs to traverse all the physical network, which we assume has diameter D. The following proposition (Proposistion 4.1) states that a propagation time of D hops is also a necessary and sufficient condition to reach max-consensus on a single virtual node allocation. Another interesting observation that follows from the result is that the number of steps for CAD to converge on the embedding of a slice of $|V_H|$ virtual nodes is always $D \cdot |V_H|$ in the worst case, regardless of the size of the bundle vector. This means that the same worst-case convergence bound is achieved if CAD runs on a single or on multiple virtual nodes simultaneously. These claims are a corollary of Theorem 1 in [4], which deals with a distributed task allocation problem for a fleet of robots.

Let the tasks allocated by a robot represent the virtual nodes to be hosted by a physical node. Therefore, by induction on the size of the bundle the following result holds as a corollary of Theorem 1 in Choi *et al.* [4]:

***Proposition** 4.1:* (*Convergence of CAD.*) Given a virtual network H with $|V_H|$ virtual nodes to be embedded on a physical network with diameter D, the utility function of each physical node is pseudo sub-modular, and the communications occur over reliable channels, then the CAD mechanism converges in a number of iterations bounded above by $D \cdot |V_H|$.

Proof: (*Sketch*). We use $\mathcal{W}_{ij}(U_{ij}, \mathbf{b}_i)$ as a bid function (sub-modular by definition). From [4] we know that a consensus-based auction run by a fleet of N_u agents, each assigned at most L_t tasks, so as to allocate N_t tasks, converges in at most $N_{min} \cdot D$ where $N_{min} = \min\{N_t, N_u \cdot L_t\}$. Note that the proof of Theorem 1 in [4] is independent of the utility function used by the agents as long as they are sub-modular, and of the constraints that need to be enforced on the tasks. Since for CAD to converge, every virtual node needs to be assigned, in the distributed slice embedding problem, N_{min} is always equal to $N_t \equiv |V_H|$, and therefore we prove the claim. ■

B. Performance Guarantees

We assume that each physical node i does not bid on a virtual node j unless it brings a positive utility, therefore U_{ij} and so $\mathcal{W}_{ij}$ are positive. Moreover, if we append the bundle $\mathbf{m}_i$ to bid on an additional set of virtual nodes $\mathbf{v}$ resulting in bid vector $\mathbf{b}'_i$, we have:

$$\mathcal{W}_{ij}(U_{ij}, \mathbf{b}'_i) \leq \mathcal{W}_{ij}(U_{ij}, \mathbf{m}_i) \; \forall \mathbf{v} \neq \emptyset \tag{6}$$

which means that $\mathcal{W}_{ij}$ is monotonically non-increasing.

Since the sum of the utilities of each single physical node, and since the bid warping function $\mathcal{W}_{ij}(U_{ij}, \mathbf{b}_i)$ of CAD is a positive, monotone (non-increasing) and sub-modular function, all the axioms of Theorem 3.1 in Nemhauser *et al.* [19] on sub-modular functions are satisfied. We hence obtain the following result:

***Proposition** 4.2:* (*CAD Approximation.*) The CAD node consensus strategy yields an $(1 - \frac{1}{e})$-approximation w.r.t. the optimal node assignment solution.

V. Performance Evaluation

To test the proposed distributed auction algorithms, we developed our own trace-driven simulator, whose code is publicly available at [1].

Physical Network Model: Using the BRITE topology generator [18], we obtain a physical topology. We use the generation model of BRITE to build a flat topology using either the Waxman model, or the Barabasi-Albert model with incremental growth and preferential connectivity. We tested our algorithms with physical network sizes varying n physical nodes with about $5n$ physical links (as in [24]). Our simulations do not consider delay constraints, while link capacity constraints are discussed later in this section. The results are similar regardless of the topology generation model and the physical network size. In this paper we only show the results obtained for $n = 50$ and a Barabasi-Albert physical topology.

Virtual Network Model: we use a real dataset of 8 years of Emulab [23] slice requests [22]. For each simulation run we process 61968 requests; the average size of a request is 14 with standard deviation of 36 virtual nodes; 99% of the requests have less than 100 virtual nodes, and 85% have at most 20 virtual nodes. Excluding the 10% long-lived requests that cause the standard deviation of slice lifetime to exceed 4-million seconds, the duration of the requests

[7] Note that in this paper we use utility functions that optimize the allocation of virtual nodes and their first-hop links, but not virtual path allocations.

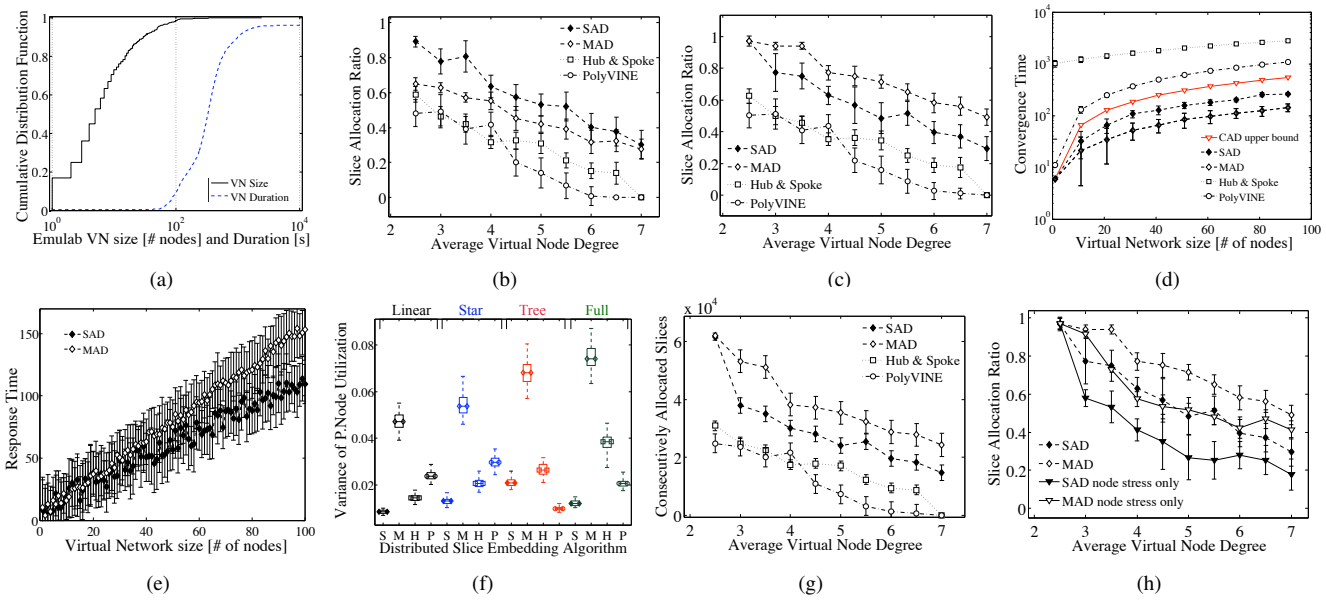

Fig. 2. (a) CDF of the size and lifetime of 8 years of Emulab slice requests. (b) SAD allocates more slices when a single shortest path is available. (c) MAD allocates more slices when a k-shortest path link allocation policy (where $k = 3$) is used. (d) MAD has shorter convergence time. (e) SAD has shorter response time. (f) SAD better balances the load on physical nodes ($k = 3$). S, M, H and P indicate SAD, MAD, Hub and Spoke and PolyViNE, respectively. (g) MAD allocates more slices consecutively ($k = 3$.) (h) Considering simultaneously node and link stress in the utility improves the slice allocation ratio.

is on average 561 with 414 seconds of standard deviation (Figure 2a.) As the dataset does not contain neither the number of virtual links nor the virtual network topology, we connect each pair of virtual nodes at random with different average node degree (Figures 2b, c, g, and h.) Moreover, we extend our evaluation comparing linear, star, tree, and fully connected virtual topologies (Figure 2f). All our simulation results show 95% confidence intervals; the randomness comes from both the virtual network topology to be embedded, and the virtual constraints, that is, virtual node and link capacity requirements. Similarly to previous work [24], we randomly assign physical link capacities between 1 and 100, then we assign the physical node capacity to be the sum of its outgoing physical link capacities. Then we assume the virtual link capacity to be randomly chosen between $1/\mathcal{R}$ and $100/\mathcal{R}$, where $\mathcal{R} = \{50, 100, 500\}$, and the virtual node capacity is then assigned to be the sum of its outgoing virtual links. The results are similar and we only show plots for $\mathcal{R} = 100$.

Comparison Method: we compare our CAD mechanism, instantiated with the SAD and MAD configuration, with another policy based distributed virtual network embedding algorithm, PolyViNE [5], and with the first published distributed virtual network embedding algorithm [12], that we call Hub and Spoke due to the adopted heuristic.

Evaluation metrics: our evaluation results quantify the benefits of our approach along two metrics: embedding efficiency and time to find a solution. In particular, we evaluate the response time — number of steps measured in one-hop communications needed to realize a VN can or cannot be embedded — and the convergence time — number of steps until a *valid* embedding is found. The efficiency of an embedding is evaluated with the VN allocation ratio — ratio between the number of virtual networks successfully embedded and requested, and with the resource utilization — physical node and link capacity utilized to embed the VN requests, as well as with the *endurance* of the algorithm, *i.e.* the number of successfully allocated requests before the first VN request is rejected. We also evaluate the effect of different utility functions.

A. Simulation results

We present here our trace-driven simulation results summarizing the key observations.

(1) MAD leads to larger VN allocation ratio, as long as multiple physical paths are available for each virtual link. When the virtual link allocation policy allows a virtual link to be allocated only on a single physical shortest path, SAD has a higher VN allocation ratio (Figure 2b.) This is because SAD, allowing a single virtual node allocation for each auction round, balances the load over physical resources more efficiently. When instead a physical node i is allowed to simultaneously win a bundle of virtual nodes $\mathbf{m}_i$ as in MAD, the physical links adjacent to i quickly exhaust their capacity due to the VN topology; all the outgoing virtual links adjacent to the virtual nodes in $\mathbf{m}_i$ that are not mapped on i are in fact mapped onto a small set of physical paths starting from physical node i. However, if the virtual link embedding policy uses a k-shortest path (with $k \geq 3$), MAD is able to allocate more VNs (Figure 2c.) From this result we conclude that when fewer physical paths are available, InPs should consider (switching to) a SAD setting, otherwise MAD is more efficient. In the considered physical topologies, there are no more than 3 physical paths between any pair of physical nodes,

and the confidence intervals overlap for SAD and MAD with $k = 2$.

(2) MAD has faster convergence time. Although we showed that MAD has the same worst-case convergence bound as SAD, simulation results show how MAD can in practice be faster (Figure 2d). In the best case, a single physical node has highest bids for all virtual nodes, and all the other bidders will converge on a VN allocation in a single auction round.

(3) SAD has faster response time. Due to the VN partitioning policy, that is, due to the fact that the SP releases only two virtual nodes at a time, SAD has a quicker response time as physical nodes immediately know if a virtual node or a link (and so the entire VN) cannot be allocated (Figure 2e.) We do not show the response time for the other algorithms in Figure 2e as they are similar to their convergence time.

(4) SAD better balances the load independent of the VN topology. To verify our findings, we average over time the variance of the utilization across all nodes with 25% and 75% percentiles for each of the algorithms, and we repeat the experiment for linear, star, tree, and full virtual network topologies (Figure 2f). Note how SAD better balances the load, independent of the VN topology. One exception is PolyViNE, that has lowest load variance for tree topologies, but at the expense of lowest VN allocation ratio.

(5) SAD allocates more VNs before the first one is rejected. As a direct consequence of a better VN allocation ratio, we verify that SAD yields a larger number of VNs allocated before the first one gets rejected in case the virtual link allocation policy allows only a single physical shortest path, while MAD allocates more requests if multiple physical loop-free paths are available (Figure 2g).

(6) Considering link stress in the utility function improves the VN allocation ratio. In this last experiment we show how different utility functions may lead to different VN allocation efficiency. In particular, by comparing two different utilities, $i.e.\ U'_{ij} = (\mathbb{T}_i - S'_{ij})$ where S' is only the stress on the physical nodes, and U_{ij} where the stress also includes adjacent physical links, we confirm the premise that considering nodes and links simultaneously in the slice embedding problem leads to higher VN allocation rate (Figure 2h). We leave the investigation of the best utility function given the goals of providers as an interesting research direction.

VI. Conclusions and future work

In this work we proposed CAD, a general distributed approach to solve the slice embedding problem, consisting of three tightly coupled phases — discovery, virtual network mapping and allocation [8]. By leveraging the distributed task assignment literature, and well-known results on sub-modular function properties, we show how CAD has bounds on both convergence and performance. Using extensive trace-driven simulations, we compare the performance of two existing distributed solutions with our mechanism, instantiated with two different sets of policies, following different providers' goals. We plan to further investigate CAD by considering larger InP topologies, and by prototyping it within Quantum [21], the networking component of the OpenStack initiative that allows building and configuring virtual network connectivity and resource allocation policies in real cloud settings.

Acknowledgment

This work is supported in part by the National Science Foundation under grant CNS-0963974.

References

[1] CAD source code. http://csr.bu.edu/cad.
[2] The GENI initiative http://www.geni.net.
[3] D. P. Bertsekas. Auction Algorithms. In *Encyclopedia of Optimization*, Dec 2001.
[4] H.-L. Choi, L. Brunet, and J. P. How. Consensus-based Decentralized Auctions for Robust Task Allocation. *IEEE Trans. of Rob.*, 25(4), 2009.
[5] M. Chowdhury, F. Samuel, and R. Boutaba. PolyViNE: Policy-Based Virtual Network Embedding Across Multiple Domains. In *Proc. of ACM SIGCOMM workshop on Virtualized Infrastructure Systems and Arch.*, VISA '10, pages 49–56, New York, NY, USA, 2010. ACM.
[6] N. R. Devanur and S. M. Kakade. The Price of Truthfulness for Pay-Per-Click Auctions. In *Proceedings of the 10th ACM conference on Electronic commerce*, EC, pages 99–106, New York, NY, USA, 2009.
[7] D. Eppstein. Finding the k Shortest Paths. *SIAM Journal of Computing*, 28(2):652–673, 1999.
[8] F. Esposito, I. Matta, and V. Ishakian. Slice Embedding Solutions for Distributed Service Architectures. *ACM Computing Surveys (to appear)*, Accepted on November 2012.
[9] U. Feige. A Threshold of ln n for Approximating Set Cover. *J. ACM*, 45(4):634–652, July 1998.
[10] GENI Planning Group. GENI Facility Design Document, http://www.geni.net/GDD/GDD-07-44.pdf, March 2007.
[11] I. Houidi, W. Louati, W. Ben Ameur, and D. Zeghlache. Virtual Network Provisioning across Multiple Substrate Networks. *Computer Networks*, 55(4):1011–1023, Mar. 2011.
[12] I. Houidi, W. Louati, and D. Zeghlache. A Distributed Virtual Network Mapping Algorithm. In *ICC '08*.
[13] N. Immorlica, D. Karger, E. Nikolova, and R. Sami. First-price Path Auctions. In *Proceedings of the 6th ACM conference on Electronic commerce*, EC '05, pages 203–212, New York, NY, USA, 2005. ACM.
[14] L. B. Johnson, H.-L. Choi, S. S. Ponda, and J. P. How. Allowing Non-Submodular Score Functions in Distributed Task Allocation. In *IEEE Conference on Decision and Control (CDC)*, 2012.
[15] A. Kulik, H. Shachnai, and T. Tamir. Maximizing Submodular Set Functions Subject to Multiple Linear Constraints. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 545–554, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
[16] R. P. Leme, V. Syrgkanis, and E. Tardos. Sequential Auctions and Externalities. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 869–886, 2012.
[17] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1st edition, Mar. 1996.
[18] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '01, pages 346–, Washington, DC, USA, 2001. IEEE Computer Society.
[19] G. Nemhauser, L. Wolsey, and M. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions. *Math. Prog.*, 1978.
[20] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: a Platform for High-Performance Internet Applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, Aug. 2010.
[21] Quantum. http://wiki.openstack.org/quantum.
[22] R.Ricci. Personal communication., 2011.
[23] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Operating System Review '02*.
[24] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, 2008.
[25] F. Zaheer, J. Xiao, and R. Boutaba. Multi-provider Service Negotiation and Contracting in Network Virtualization. In *IEEE Network Oper. and Management Symposium (NOMS), 2010*, pages 471 –478, April 2010.
[26] Y. Zhu, R. Zhang-Shen, S. Rangarajan, and J. Rexford. Cabernet: Connectivity Architecture for Better Network Services. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT, 2008.