Computer Networks 161 (2019) 45–60

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Secure Multi-Cloud Network Virtualization

Max Alaluna*, Eric Vial, Nuno Neves, Fernando M.V. Ramos

LASIGE, Faculdade de Ciências, Universidade de Lisboa, Departamento de informática, Edifício C6 Piso 3, Campo Grande, Lisboa 1749-016, Portugal

ARTICLE INFO

Article history:
Received 12 November 2018
Revised 18 May 2019
Accepted 5 June 2019
Available online 13 June 2019

Keywords:
Network virtualization
Network embedding
Multi-cloud platform

ABSTRACT

Existing network virtualization systems share a few characteristics, namely they target one data center of a single operator and only offer traditional networking services. As such, their support for critical applications that need to be deployed across multiple trust domains, while enforcing diverse security requirements, is limited. This paper enhances the state-of-the-art by presenting a multi-cloud network virtualization system, allowing the provision of virtual networks of containers. Our solution enables a provider to enrich its network substrate with public and private cloud-based resources, increasing flexibility and the range of supplied services. One challenging aspect that we tackle is the embedding of virtual network requests to the substrate infrastructure, as existing work is unfit to a modern data center context, scales poorly or does not consider the security of virtual resources. We propose a scalable heuristic that considers security as a first-class citizen and is specifically tailored to a hybrid multi-cloud domain. We evaluate our algorithm with large-scale simulations that consider realistic network topologies and our prototype in a substrate composed of one private data center and two public clouds. The system scales well for networks of thousands of switches employing diverse topologies and improves on the virtual network acceptance ratio, provider revenue, and embedding delays. Our results show that the acceptance ratios are less than 1% from the optimal and that the system can provision a 10 thousand container virtual network in approximately 2 minutes.

1. Introduction

Network virtualization gives an unprecedented level of flexibility to network operators. By shifting from traditional primitives (e.g., VLANs) and by embracing network-wide centralized control, modern virtualization systems [1–3] allow the complete decoupling of the (virtual) network from its physical realization. These production-level platforms facilitate provisioning, helping providers cope with the operational challenge of maintaining an efficient network while allowing users to migrate their workloads, unchanged, from their enterprises to the service providers facilities.

Existing solutions share a few characteristics. First, they target a single operator with full control over the infrastructure. For instance, NVP [1], AccelNet [2], and Andromeda [3], are single-operator solutions grounded on having entire control of the hypervisors at the edge, to enable network virtualization. The provider-centric nature of these solutions has drawbacks, however, as it limits the scale of the infrastructure to one cloud and introduces reliability and security concerns. A regional blackout affecting the operator (an event increasingly common [4,5]) would compromise a large number of applications. Security is also an increasing concern to users that consider migrating their workloads to the cloud. Recent studies report “security risks” to be a top barrier of adoption of cloud services [6]. Second, existing platforms restrict the services offered to traditional networking, namely L2 switching and L3 routing. With respect to security, for instance, the available services are limited to fine-grained forms of ACL filtering (e.g., micro-segmentation), and network availability is not considered. Commercial solutions (e.g., [7]) recently started including multi-cloud approaches and disaster recovery services, but their closed nature leaves it unclear how these services are materialized.

In this paper, we enhance the state-of-the-art by presenting our open-source multi-cloud network virtualization system: Sirius. The main idea is to enrich the substrate network with resources from public and private clouds in order to enhance the networking services of users, namely concerning security. Significant benefits from a multi-cloud approach include, first, the ability to scale out tenant workloads by outsourcing resources, enabling elasticity of the virtual infrastructure. This can be used not only to extend the virtual network across providers but also to save costs (e.g., by optimizing pricing schemes [8,9]). Second, it improves performance and dependability. If resources are spread (and replicated) across clouds it becomes possible to explore locality to decrease delays

* Corresponding author.
E-mail addresses: malaluna@alunos.fc.ul.pt (M. Alaluna), eevial@fc.ul.pt (E. Vial), nuno@di.fc.ul.pt (N. Neves), fvramos@ciencias.ulisboa.pt (F.M.V. Ramos).

https://doi.org/10.1016/j.comnet.2019.06.004
1389-1286/

and to tolerate operator-wide failures [4,5]. Third, it enhances the security options for the virtual networks. For instance, the user may consider a subset of her virtual network to be more sensitive, restricting it to a specific location considered more trustful (e.g., a cloud with stricter SLA). Or she may want to abide by the recently implemented General Data Protection Regulation (GPDR), by placing a virtual switch connecting databases that contain user data at a specific private location under the tenant's control, while the rest of the network is offloaded to public cloud infrastructures, to take advantage of their elasticity and flexibility. These advantages result in this type of deployments to be trending upwards, with 85% of enterprises reporting to already have a multi-cloud strategy for their business [10].

However, exploring these benefits is challenging. As the multi-cloud provider has no control over the public cloud resources (e.g., the VM hypervisor), it is necessary to employ some form of nested virtualization [11], and this may impact performance. Besides, the problem of embedding virtual resources in this setting is highly complicated. In particular, it is necessary to deal with a hybrid substrate, as private data center topologies (typically a Clos variant) differ significantly from the network offered by a public cloud (a full mesh or "big switch"). Since most network embedding algorithms [12] target wide-area networks and mesh topologies, they perform poorly when directly applied to this context. They are also unsuitable for any practical deployment, as they often recur to solving the Multi-Commodity Flow (MCF) problem for link mapping [13]. This approach is not only slow to be applied to large infrastructures (as we show in Section 5) – it is also impractical, as it would be necessary to adjust the resulting path splits to the granularity supported by the underlying cloud, a problem not addressed in the embedding literature. Aside from this, as our goal is to leverage from an enriched substrate to enhance the security and availability of virtual networks, it is necessary to extend the embedding algorithms with these new requirements further. So far, the security problem has received relatively little attention in the literature (we will detail this further in Section 6). We have explored this problem recently [14], by proposing a MILP solution for the secure VNE problem considering a multi-cloud network substrate. Unfortunately, that solution scales very poorly (see Section 5.2).

To address these challenges we propose a novel embedding algorithm, a core conceptual contribution of our work. Our solution achieves five goals – all five are necessary for a production-level deployment. First, it makes efficient use of the substrate resources, achieving a very high acceptance ratio of virtual network requests, consequently increasing provider profit. Second, it is topology-agnostic, allowing it to achieve good results for radically different topologies. Third, it allows users to specify the level of security (including availability) of each element of their virtual networks and guarantees the fulfillment of their requests. Forth, it improves application performance by significantly reducing the average path length. Finally, it scales well, making it practical for large-scale deployments.

This algorithm has been implemented in our Sirius system, supporting the management of multi-cloud substrate infrastructures, and the provisioning of user-defined virtual network requests. We analyze the behavior and performance of the algorithm with large-scale simulations that consider a private data center following Google's Jupiter topology design [15], extended with hundreds of cloud resources spread across two public clouds. Also, we evaluate our Sirius prototype in a substrate composed of three clouds: one private datacenter and two public clouds (Amazon EC2 and Google Cloud Platform). We demonstrate that our system allows virtual networks to extend across multiple clouds, without significant loss of performance compared to a non-virtualized substrate. When compared with the state-of-the-art approaches, our novel embedding algorithm enables multi-cloud providers to increase the virtual network acceptance ratio (with results less than 1% lower than the optimal), reduce path lengths, and grow provider revenue. Our evaluation also shows that virtual networks with several thousands of virtual hosts can be provisioned quickly, even if spread over several clouds.

In summary, the main contributions of our work are:

- The design and implementation of a multi-cloud network virtualization system that improves over the state-of-the-art by enabling users to define security requisites for all elements of the virtual networks;
- A network embedding algorithm that is demonstrated to be the best fit for our scenario, increasing the multi-cloud network acceptance ratio and quality of service, alongside scaling properties that guarantee its applicability in large-scale deployments;
- A detailed evaluation of our proposal under various scenarios.

We also made our system Sirius available open source[1], including all evaluation data (scripts and results).

2. Design

In this section, we present the design of Sirius. The initial inspiration for our solution draws from modern virtualization platforms [1,16]. Precisely, it departs from distributed control, leveraging the advantages of its logical centralization to orchestrate a distributed set of data plane elements.

Our solution allows users to define their virtual networks (VN) with arbitrary topologies and addressing schemes, and guarantee isolation of all tenants that share the substrate (see Fig. 1). On top of this foundation, we set a couple of ambitious goals for our platform, including the improvement of its scalability properties, and the enhancement of the networking services available to users. Towards these objectives, we introduce an additional set of requirements – the technical innovations that materialize these requirements are the core of this paper. These are: *Substrate scalability:* Allow the network substrate to scale out, by extending it with public cloud resources. *System scalability:* Handles on the order of the hundreds of requests per second. *Enhanced virtual network services:* Allow users to define the security and availability requirements of every element of their virtual networks, increasing their dependability. *Provider profit:* The mapping of virtual to substrate resources should maximize provider profit, through high acceptance ratios and efficient utilization of resources. *Fit for a hybrid multi-cloud:* The solution should perform well in a diverse substrate network with different topologies, including public (e.g., Amazon EC2), private (e.g., a modern data center), and hybrid clouds (e.g., a private DC extended with public cloud resources). *Practicality:* The constraints of the network substrate should be taken into account. *Performance:* The virtualization layer should not introduce significant overhead and application performance should not degrade.

These requirements are not met by any existing solution, which at most address a subset of these criteria. We further differentiate from related work in Section 6.

2.1. Virtual and Substrate Networks

Virtual networks. In our platform, the user can define her virtual network (Fig. 1(b), left) using a graphical user interface or through a configuration file. She can define any arbitrary topology composed of a set of *virtual hosts* and a group of *virtual switches*, interconnected by *virtual links*. The virtual switches can be of one

[1] https://github.com/netx-ulx/Sirius.

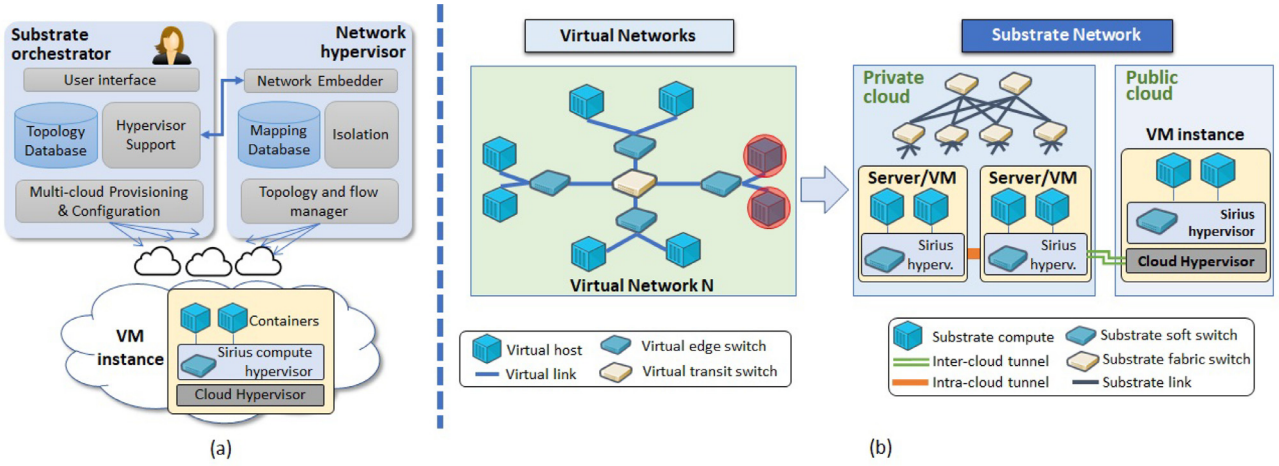


Fig. 1. (a) System architecture; (b) Virtual networks and substrate.

of two types: *virtual edge switches*, in case they have virtual hosts attached, or *virtual transit switches*, in case they do not. The virtual hosts can be configured with any addresses from the entire L2 and L3 address space.[2] The virtual links are configured with the bandwidths and latencies required.

A core contribution of our solution arises from allowing users to further enhance their networks by setting the specific security requirements of each virtual host, switch, and link. For instance, specific virtual hosts may be considered sensitive, leading to restrictions about their location (e.g., to be hosted in a trusted facility) or of its type (e.g., to be hosted in a secure substrate, such as one offering threat prevention or encryption). For this purpose, our solution enables the network provider to define the security level of each cloud and each data plane element of the substrate, allowing users to specify the security requirements of all hosts, switches, and links of the virtual network. It is also possible to define the level of availability of virtual hosts. In this case, our system enforces these elements to be replicated, accordingly with their level. For instance, if the level of availability required is high, it may be replicated in a different cloud, to tolerate large-scale cloud outages.

Fig. 1 illustrates a virtual network in our system, with a simplified set of requirements. In this example, the nodes within a red circle represent sensitive elements and therefore need to be located in a trusted cloud.

Substrate network. Our system allows a network provider to extend its infrastructure by enriching its substrate with resources from public cloud providers, allowing it to be shared (transparently) by various users (Fig. 1(b), right). The resources are organized in such a way to create a single multi-cloud abstraction. Our substrate is composed of one or several private infrastructures, and one or several public clouds. In this infrastructure, the *substrate compute* elements run on top of the Sirius compute hypervisor, and are inter-connected by *substrate links* and *substrate switches*.

Every hypervisor runs one *substrate software switch* to allow communication between the substrate compute elements and between these and the outside world. We consider a second type of substrate switches in our model: the *substrate fabric switch*. This corresponds to a physical switch under Sirius's control – typically part of a private data center (DC) fabric. Note, however, that this does not mean the solution to require full control of all DC switches. If the provider does not have centralized control over the fabric (as it uses traditional L2/L3 networking), then the substrate topology will not include the fabric switches. In this case, the only switching elements are the software switches that run at the edge. The *substrate links* include tunnels (both intra-cloud and inter-cloud) and physical links. Again, physical links are only included in the substrate topology if Sirius has control over the physical switches to which they connect. Finally, every cloud includes one gateway that acts as an edge router. The inter-cloud tunnels are set up and maintained in this node (as such, this is the only element that requires a public IP address).

2.1.1. Architecture

Our system's architecture has two core components (Fig. 1(a)): a substrate orchestrator, that deals mainly with initializing, configuring, and updating the substrate resources; and the network hypervisor, chiefly responsible for embedding network requests into the substrate fulfilling the user's requirements, setting up and monitoring the network state, and guaranteeing isolation between virtual networks.

Substrate orchestrator. The orchestrator is composed of four modules that deal with user interaction and management of the substrate infrastructure. A web-based graphical interface allows administrators and users to define the substrate and virtual infrastructure topologies, respectively. The administrator and tenant interfaces are similar, but each includes its own user-specific parameters. The Multi-cloud Provisioning & Configuration module is responsible for the deployment of the required VM instances that form the substrate, by interacting with the various clouds. Afterwards, this module installs the compute hypervisor (that includes a software switch) and configures the required external tunnels.

Interaction with the network hypervisor is enabled by the Hypervisor Support module. First, it starts by forwarding virtual network requests to the network hypervisor. When it receives a response with the required mappings, it initializes the necessary substrate compute elements (containers) to map the virtual hosts. The orchestrator maintains information about the virtual and substrate topologies, along with its mappings, in a Topology Database.

Network hypervisor. The hypervisor initializes when the substrate network is set up. Its first task is to obtain information about the infrastructure from the orchestrator. This is enabled by interaction between the Hypervisor Support and Network Embedder modules. Afterward, the Topology and flow manager module contacts all data plane elements to populate its own internal data structures (e.g., get number and IDs of switch ports).

At this point, the hypervisor is ready to start receiving virtual network requests (VNR) forwarded by the orchestrator. When such request arrives to the Network Embedder, the hypervisor runs the

[2] With the restriction that they are not allowed to use the same address in different hosts of their network.

embedding algorithm to map the VNR into the substrate. As no existing solution was fit for our problem, this became the core algorithmic challenge to be solved. We thus leave the description of the algorithm we propose to the next section. After the embedding phase the results are communicated back to the Hypervisor Support module and the user requirements are enforced by the Multi-cloud Provisioning & Configuration module, that deploys the virtual hosts, virtual edge switches, and virtual links at, respectively, substrate compute elements, substrate software switches, and substrate links, that satisfy all requirements and have enough resources available to fulfill the specific demands. As the virtual transit switches do not have any virtual host attached, they can map to either a substrate software switch or a substrate fabric switch. Next, the Topology and flow manager sets all data paths in the data plane. Changes to the topology are also monitored, and data paths are dynamically teared up and down, as necessary.

The hypervisor employs three main techniques to ensure isolation. These are implemented in the Isolation module. First, every substrate compute element has a unique ID, the `hostID`, based on its location. This ID includes the substrate switch identifier and the port to which it connects. The hypervisor maintains a structure with all virtual-to-substrate mappings in the Mapping Database module, including information about the virtual network of each element. As different virtual networks can have elements with the same L2 and/or L3 addresses, the virtual addresses cannot be sent unmodified to the substrate. As such, the second technique we employ is address translation at the edge. We assign an ephemeral MAC (`eMAC`) to replace the virtual host MAC address (similar to the PMAC used in Portland [17]). The `eMAC` includes a compact version of the `hostID` and the user ID. Every time traffic originates from a virtual host, the MAC is translated, at that edge switch, in the eMAC. When traffic arrives at the destination, the inverse operation takes place. Third, as we want host applications to run unmodified in our system, Sirius intercepts all ARP messages and treats them in a way that is transparent to the hosts.

3. Network Embedding

The next section abstracts the elements of the infrastructure in a model that captures the fundamental characteristics of the substrate and user demands for the virtual networks. Sirius then uses the models to optimize the embedding of user requests in the clouds.

3.1. Network model

3.1.1. Substrate Network

The substrate network is modeled as a weighted undirected graph $G^S = (N^S, E^S, A_N^S, A_E^S)$, where N^S is a set of nodes, E^S is the group of links (or edges) and A_N^S / A_E^S are the attributes of the nodes and edges. A node n^S is a network element capable of forwarding packets. It can be either a software switch or a fabric switch, which is modeled by an attribute $type(n^S)$ with values 0 or 1, respectively.

A software switch connects several local compute elements (e.g., containers) to the infrastructure. All run in the same physical (or virtual) machine and share the local resources. Therefore, we employ attribute $cpu(n^S)$ to aggregate the total CPU capacity available for network tasks and processing of user applications. Similarly, these components have an equivalent set of protections and are located in the same cloud. We use attributes $sec(n^S) \geq 0$ and $cloud(n^S) \geq 0$ to represent the security level of the ensemble and the trust associated with the cloud, with higher values associated to stronger safeguards.

Fabric switches are internal routing devices, utilized for example in the access or aggregation layers of a data center. They are optional elements because our solution can enforce all necessary traffic forwarding decisions by configuring only the software switches at the edge. However, they allow for additional flexibility when computing the paths, often leading to more efficient embeddings. Since public cloud providers disallow the configuration of internal network devices, we restrict the modeling of fabric switches to private data centers. These switches have the same attributes as above, where the values for $sec(n^S)$ and $cloud(n^S)$ are dictated by the risk appetite of the owner organization. Overall, the attributes of a node are $A_N^S = \{\{type(n^S), cpu(n^S), \text{sec}(n^S), cloud(n^S)\} \mid n^S \in N^S\}$.

The edges are characterized by the total bandwidth capacity (attribute $bw(e^S) > 0$) and the average latency ($lat(e^S) > 0$). They also have an associated security level ($\text{sec}(e^S) > 0$), where for example inter-cloud links may be perceived less secure than edges of a private datacenter, as the first have to be routed over the Internet. Overall, the edge attributes are $A_E^S = \{\{bw(e^S), lat(e^S), \text{sec}(e^S)\} \mid e^S \in E^S\}$.

In Sirius, users have two means to specify the substrate network. We have a graphical tool to enable the drawing of the network topology and set the attributes. As alternative, a policy grammar can be employed to describe the substrate in a text file (details in [14]). In both cases, some aspects of our infrastructure can be explored to ease the specification. For example, default security levels can be inherited by all internal links and nodes of a cloud. In addition, instead of relying on the user to provide the links latencies, they can be calculated using measurements performed on the network (which are periodically updated).

3.1.2. Virtual Networks

VNs are also modeled as weighted undirected graphs $G^V = (N^V, E^V, A_N^V, A_E^V)$, where N^V is the set of virtual nodes, E^V is the set of virtual links (or edges), and A_N^V / A_E^V are the node and edge attributes. These attributes are similar to the substrate network attributes. For example, $type()$ classifies whether a node is a virtual edge or a virtual transit switch. In this case, and to reduce the specification effort, $type(n^V)$ is inferred by the system using a straightforward rule – *a switch with no virtual host attached is considered a virtual transit switch; otherwise, it is a virtual edge switch.*

A node n^V that corresponds to a virtual edge switch models the requirements of the switch and the locally connected hosts. Regarding demanded CPU, the attribute $cpu(n^V)$ is the sum of all requested processing capacity (for network tasks and applications). With the other attributes, we take the most strict requirement of all elements (e.g., if hosts need a security level of 4 but the switch only asks for 1, then $sec(n^V)$ is 4). For virtual transit switches, there is a direct relation between the requirements and the attributes of the matching node.

VNs only have an extra attribute in A_N^V to support enhanced availability. In many scenarios, hosts should be replicated so that backups can take over the computations after a failure. This means that during embedding additional substrate resources need to be allocated for the backups. The attribute $avail(n^V)$ can take three pre-defined values: $avail(n^V) = 0$ means no replication; $avail(n^V) = 1$ requests backups in the same cloud; for replication in another cloud then $avail(n^V) = 2$ (e.g., to survive from a cloud outage).

Overall, the two sets of attributes are: $A_N^V = \{\{type(n^V), cpu(n^V), \text{sec}(n^V), cloud(n^V), avail(n^V)\} \mid n^V \in N^V\}$ and $A_E^V = \{\{bw(e^V), lat(e^V), \text{sec}(e^V)\} \mid e^V \in E^V\}$.

3.2. Scalable SecVNE

Sirius instantiates the users' requests by mapping the associated VNs onto the substrate while respecting all declared attributes. Our solution handles the on-line VNE problem, where every time a VNR arrives there is an attempt to find appropriate provisioning

in the substrate. While deciding on the location of the resources, we perform a heuristic mapping with the purpose of increasing the overall acceptance ratio, reducing the usage of substrate resources, and fulfilling the security needs. If we find a solution, the residual capacity of the substrate resources decreases by the amount that is going to be consumed. Otherwise, we reject the request.

While experimenting with existing embedding algorithms, we observed some limitations when applied to our multi-cloud environment, namely related with the inability to address the security requirements (for instance, [13,18,19]), the incapacity to support large numbers of switches (for instance, [14,18,20,21]), and inefficiencies on the use of resources, often because the assumed network model is not a good match for our setting (for instance, [13,18–23]). Therefore, we designed a new algorithm built around the following ideas: (i) Optimal embedding solutions, for instance, based on linear program optimization do not scale to our envisioned scenarios, and therefore we employ a greedy approach based on two utility functions to guide the selection of the resources; (ii) The mapping of the virtual resources to the substrate is carried out in two phases: in the first the nodes are chosen, and then the links. While ensuring the security constraints, we give priority to the security level over the cloud trust; (iii) The backup resources necessary to fulfill availability requirements are only reserved after the primary resources have been completely mapped, giving precedence to the common case where no failures occur. The next subsections detail the various parts of our solution.

3.2.1. Utility Functions

The process for selecting the substrate nodes uses two utility functions to prioritize the nodes to pick earlier. The first function, *UResSec*(), utilizes only information about the current resource consumption and the node security characteristics. In particular, it values more the nodes that: (i) have the highest percentage of available resources, as this contributes to an increase in the VNR acceptance ratio; and (ii) provide the lowest security assurances (but still larger than the demanded ones), to reduce the embedding costs (highly secure cloud instances are typically substantially more expensive than regular ones[3]). The utility of a substrate node n^S is:

$$UResSec(n^S) = \frac{\%R_N(n^S) \times \sum_{\forall e^S \to n^S} \%R_E(e^S)}{sec(n^S) \times cloud(n^S)} \quad (1)$$

where the $\%R_N(n^S) = R_N(n^S)/cpu(n^S)$ is the percentage of residual CPU capacity (or available capacity) of a substrate node. The residual capacity is computed with Eq. (2), with $n^V \in N^V$ and $x \uparrow y$ denoting that the virtual node x is hosted on the substrate node y:

$$R_N(n^S) = cpu(n^S) \; - \sum_{\forall n^V \uparrow n^S} cpu(n^V) \quad (2)$$

and, where the sum element of Eq. (1) corresponds to the overall available bandwidth of the edges connected to n^S ($e^S \to n^S$ means n^S is an endpoint of e^S). The value of $\%R_E(e^S) = R_E(e^S)/bw(e^S)$ is the percentage of the residual capacity of a substrate link. The residual capacity is calculated with Eq. (3), with $e^V \in E^V$ and $x \uparrow y$ denoting that the flow of the virtual link x traverses the substrate link y:

$$R_E(e^S) = bw(e^S) \; - \sum_{\forall e^V \uparrow e^S} bw(e^V) \quad (3)$$

The second utility function, *UPath*(), contributes to decrease the distance (in a number of hops) among the substrate nodes of a VN embedding (for this reason we term this function *Path Contraction*), improving the QoS and decreasing further the provider costs. This allows coupling node to link mapping, which is key to improve network efficiency, as we show in Section 5.3. When a virtual node n^V is to be mapped, the utility of selecting a substrate node n^S is computed with two factors: (i) the *UResSec*() of n^S; and (ii) the average distance between n^S and the substrate nodes that have already been used to place the neighbors of n^V (given by function *avgDist2Neighbors*()). The substrate node utility with path contraction is:

$$UPath(n^S, n^V) = \frac{UResSec(n^S)}{avgDist2Neighbors(n^S, n^V)} \quad (4)$$

The intuition behind dividing the (initial) substrate node utility by the average distance is to diminish the utility of substrate nodes located further away from the nodes already mapped. The effect is a lower communication delay (average path length is shorter) and, simultaneously, a decrease in bandwidth costs.

3.2.2. SecVNE Algorithm

Our *Secure Virtual Network Embedding (SecVNE)* algorithm consists of three procedures: (i) receiving and processing the tenants VN request (main SecVNE procedure); (ii) primary node and edge mapping, employing the utility functions to steer the substrate resource selection; and (iii) backup mapping, allocating disjoint resources to avoid common mode failures.

SecVNE: VN requests are embedded into the substrate with Algorithm 1. The algorithm expects two kinds of inputs: (1) the VN graph (G^V) including all attributes; and (2) the substrate description, with the graph (G^S) and current nodes' and edges' residual capacities (R_N, R_E). At system initialization, the algorithm assigns the residual capacities with the values of the resource capacities in the substrate graph, but as VNRs are serviced, they are updated using Eqs. (2) and (3). Also, when a VNR execution ends, the residual capacities are increased to reflect the release of the associated resources.

The algorithm can potentially produce two mappings. One is for the network used in normal operation, called the *PMap* (from *pri-*

Algorithm 1: Main SecVNE Procedure.

Input: G^V, G^S, R_N, R_E
Output: *PMap* // mapping for primary network
Output: *BMap* // mapping for backup network

1 $PMap.N \leftarrow NodeMapping(G^V, G^S, R_N, R_E)$;
2 $PMap.L \leftarrow LinkMapping(G^V, G^S, R_N, R_E, PMap.N)$;
3 **if** $((PMap.N \neq \emptyset) \wedge (PMap.L \neq \emptyset))$ **then**
4 $Rtemp_N \leftarrow R_N$;
5 $Rtemp_E \leftarrow R_E$;
6 $UpdateResources(R_N, R_E, PMap)$;
7 **if** *(at least one virtual node needs backup)* **then**
8 $BMap.N \leftarrow BNodeMap(G^V, G^S, R_N, PMap)$;
9 $BMap.L \leftarrow BLinkMap(G^V, G^S, R_N, R_E, PMap)$;
10 **if** $((BMap.N \neq \emptyset) \wedge (BMap.L \neq \emptyset))$ **then**
11 $UpdateResources(R^S_N, R^S_E, BMap)$;
12 **return** (*PMap*, *BMap*);
13 **else**
14 $R_N \leftarrow Rtemp_N$;
15 $R_E \leftarrow Rtemp_E$;
16 **return** $(\emptyset, \emptyset)$;
17 **else**
18 **return** (*PMap*, $\emptyset$);
19 **else**
20 **return** $(\emptyset, \emptyset)$;

[3] For example, the cost of a Trend Micro Deep Security instance at aws.amazon.com/marketplace is five time higher than a normal instance.

Algorithm 2: NodeMapping().

Input: G^V, G^S, R_N, R_E
Output: nodeMap // mapping for the nodes

1 $nodeMap \leftarrow \emptyset$;
2 $scoreT \leftarrow getScore(G^V)$;
3 $utilT \leftarrow getUtil(G^S, R_N, R_E)$;
4 **forall the** $(n_i^V \in G^V)$ **do**
5 $virtualNodeMapped \leftarrow false$;
6 $candidT = getCandidates(n_i^V, G^V, G^S, utilT)$;
7 **forall the** $(n_j^S \in candidT)$ **do**
8 **if** $(cpu(n_i^V) \leqslant R_N(n_j^S))$ **then**
9 $nodeMap \leftarrow nodeMap \cup (n_i^V, n_j^S)$;
10 $delUtil(n_j^S, utilT)$;
11 $virtualNodeMapped \leftarrow true$;
12 *break*;
13 **if** $(virtualNodeMapped = false)$ **then**
14 **return** $\emptyset$;
15 **return** *nodeMap*;

mary mapping), and the other is for the backup nodes and edges to be employed in case of failure, called the *BMap* (from *backup mapping*). A mapping is a set of tuples, each with a virtual resource identifier and the corresponding substrate resource(s) where it will be placed (and some additional information).

The algorithm logic is relatively simple. It starts by seeking for a *PMap* (Lines 1–2). Then, it changes the residual capacities based on resource consumption after deployment (Line 6). If at least one of the virtual nodes requires availability support ($aval(n_i^V) > 0$), then a backup mapping is also obtained (Lines 8–9), and the capacities are updated (Line 11).

Node Mapping: Algorithm 2 implements the *NodeMapping*() procedure, whose goal is to find a valid embedding for the virtual nodes of the VN, taking into consideration all attribute requirements.

The procedure starts by creating a table where a score value orders the virtual nodes (Line 2, and top of Fig. 2). The virtual node score is calculated using:

$$NScore(n^V) = \frac{cpu(n^V) \times \sum_{\forall e^V \rightarrow n^V} bw(e^V)}{sec(n^V) \times cloud(n^V)} \quad (5)$$

where $e^V \rightarrow n^V$ means n^V is an endpoint of link e^V. In the figure, virtual node 1 has a score of 2, and virtual node 3 has a score of 9. As processing of virtual nodes is performed in increasing score order, in this example these virtual nodes are thus the first and the last, respectively, to be processed.

Next, we calculate the *UResSec* utility for all substrate nodes (Line 3; middle of the figure). For instance, the *UResSec* score of substrate node *a* is equal to 15, whereas the score of *c* is 72 and of *e* is 70. These scores are stored in *utilT*, a hashMap indexed by the security level and cloud trust of the node, to optimize accesses by security demand. The scores of substrate nodes *c* and *e* referred above, for instance, are stored in the table line that corresponds to a security level of 2, and a cloud trust of 2.

Virtual nodes n_i^V are then processed one at a time. For each, we select all acceptable candidate substrate nodes, i.e., the switches n_j^S that provide security assurances of at least the same level requested ($sec(n_j^S) \geq \text{sec}(n_i^V)$ and $cloud(n_j^S) \geq cloud(n_i^V)$). As an example, if the virtual node requires a security level and cloud trust of (at least) 2, the substrate nodes included in last *utilT* line are not considered, as those have a cloud trust and security level of only 1. In addition, candidates are chosen based on the type of virtual node: if n_i^V is a virtual edge switch ($type(n_i^V) = 0$) then only substrate software switches are acceptable ($type(n_j^S) = 0$); otherwise, for virtual transit switches ($type(n_i^V) = 1$), we allow either software or fabric substrate switches. The algorithm places these candidates in a structure called *candidT*, ordered by decreasing *UPath* utility value (i.e., not ordered using the *UResSec* score stored in the *utilT* table) – observe Line 6 of the algorithm, and bottom of the figure. For instance, node *e*, with a *UResSec* score equal to 70, has a *UPath* score of 20. Next, we search *candidT* for the first node that has enough residual CPU capacity (Line 8), and use the first to be found (Line 9). We also remove this node from *utilT* to prevent further mappings to this substrate node from this VNR (i.e., from this tenant), thus avoiding situations where a single failure would compromise a significant part of the primary virtual network of a single tenant (Line 10).

We should note that the ordering of the search has a substantial impact on performance, namely concerning request acceptance and costs – we recall that the processing of *scoreT* is in increasing score order, while *candidT* is in decreasing utility order. The intuition, which was confirmed by our simulations, is that this leads to embeddings where: (i) virtual nodes with modest security demands are mapped to substrate nodes that give fewer assurances ("freeing" nodes with higher security level to serve more demanding requests); (ii) nodes end up being physically located near to each other; (iii) there is a more even distribution of the residual capacities.

Link Mapping: Algorithm 3 finds a mapping between the virtual edges and the substrate network. Each edge is processed individu-

Algorithm 3: LinkMapping().

Input: G^V, G^S, R_E, *nodeMap*
Output: *linkMap* // link mappings
1 $linkMap \leftarrow \emptyset$;
2 $Rtemp_E \leftarrow R_E$;
3 **forall the** $(e_i^V \in E^V)$ **do**
4 $totalBw \leftarrow 0$;
5 $Rloop_E \leftarrow R_E$;
6 **forall the** $(e_j^S \in E^S)$ **do**
7 **if** $(sec(e_i^V) > \text{sec}(e_j^S))$ **then**
8 $Rloop_E(e_j^S) \leftarrow 0$;
9 $Paths \leftarrow getPaths(e_i^V, G^V, G^S, Rloop_E, nodeMap)$;
10 **foreach** $(p \in Paths)$ **do**
11 **if** $(lat(e_i^V) \geq getLatency(p, G^S))$ **then**
12 $bwp \leftarrow getMinBandwidth(p, Rloop_E)$;
13 $totalBw \leftarrow totalBw + bwp$;
14 $candP \leftarrow candP \cup (e_i^V, bwp, p)$;
15 **if** $(|candP| = MaxPaths)$ **then**
16 *break*;
17 **if** $(totalBw \geq bw(e_i^V))$ **then**
18 **forall the** $(mp \in candP)$ **do**
19 $bw(mp) \leftarrow \lceil (bw(mp)/totalBw) * bw(e_i^V) \rceil$;
20 $UpdateLinkResources(R_E, candP)$;
21 $linkMap \leftarrow linkMap \cup candP$;
22 **else**
23 $R_E \leftarrow Rtemp_E$;
24 **return** $\emptyset$;
25 **return** *linkMap*;

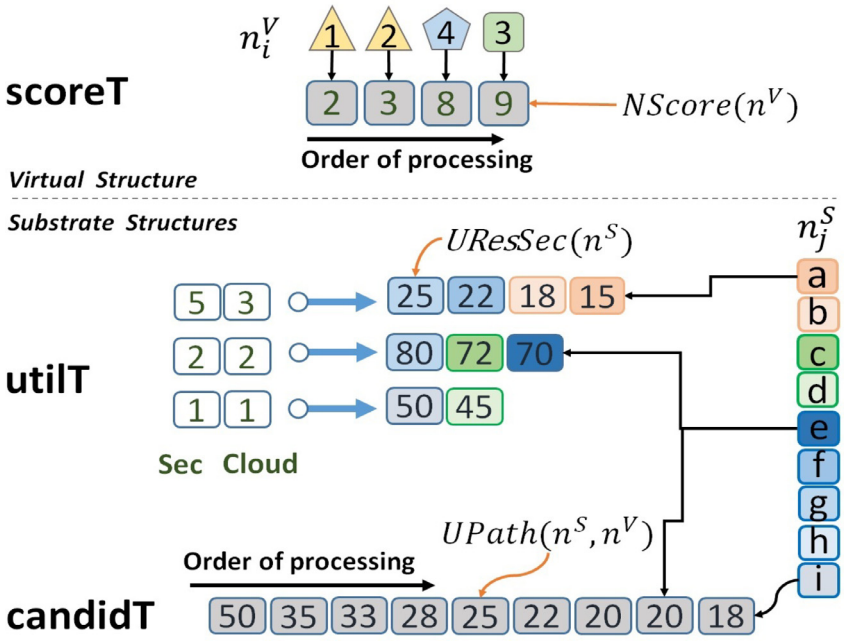


Fig. 2. Data structures used in node mapping.

ally, searching for a suitable network connection between the two substrate nodes where its virtual endpoints will be embedded. The approach is flexible, allowing the use of single or multiple paths.

The algorithm consists of the following steps. First, the substrate edges that do not provide the necessary security guarantees are excluded (Lines 6–9). For this purpose, we set to null the residual bandwidth capacity of those edges on an auxiliary variable $Rloop_E$ (Line 9), thus preventing their selection at later steps.

Second, we obtain a set of paths that could be employed to connect the two substrate nodes where the virtual edge endpoints will be embedded (Line 10). In our implementation, we resort to the K-edge disjoint shortest path algorithm to find these paths, using as edge weights the inverse of the residual bandwidths. This ensures that when "distance" is minimized, the algorithm picks the paths that have the most available bandwidth.

Third, the substrate paths to be used are chosen (Lines 11–17). Prospect paths p are an ordered sequence of substrate edges ($p = (e_1^S, e_2^S, \ldots)$), which have a certain latency (equal to the sum of the link latencies, and calculated by *getLatency*()) and a maximum bandwidth (given by the smallest residual bandwidth of all the edges, and computed by *getMinBandwidth*()). Eligible paths need to have latency less than the requested (Line 12). We store these paths in a candidate set *candP* together with the corresponding virtual edge and available bandwidth (Line 15). The set will have at most *MaxPaths*, a constant that defines the degree of multipathing (when set to 1, we use a single path) (Lines 16–17). This constant can be used to prevent an excessive level of traffic fragmentation, which is important when managing the number of entries in the packet forwarding tables of the switches. On the other hand, it improves dependability because localized link failures can be automatically tolerated with multi-path data forwarding (if enough residual capacity exists in the surviving paths).

The last step is to define how much traffic goes through each path, ensuring that together they provide the requested edge bandwidth (Lines 18–22). An edge can only be mapped if enough bandwidth is available in the paths (Line 18). In this case, we update the bandwidth in every path to an amount proportional to their maximum capacity, therefore distributing the load (Lines 19–20). Then, the residual capacities of the substrate edges are updated, accordingly to this embedding procedure (Line 21), and the set of paths is saved (Line 22).

Backup Mapping: To improve the availability of their virtual networks, tenants can specify replication of specific virtual nodes. For this purpose, the algorithm performs a backup embedding that satisfies the same attribute requirements as the primary mapping. The backup functions, *BNodeMap*() and *BLinkMap*() called in Algorithm 1, operate similarly as their normal counterparts, with the exception that we exclude all resources used in the primary mappings from the embedding. We note that there are a few cases, however, where it may be impossible to enforce this objective. For instance, inside the same rack typically there is only one ToR switch. So, in this specific case, there is some level of sharing.

4. Implementation

We implemented the Sirius network hypervisor as an SDN application on top of the Floodlight controller. The orchestrator is deployed on an Apache Tomcat server. For the compute hypervisor, we have resorted to Docker with Open vSwitch [24] as the software switch. As such, substrate compute elements are Docker containers. Users interact with the system through a GUI based on a browser-based visualization library [25].

Our system was deployed in a substrate composed of two public clouds (Amazon EC2 and Google Compute Engine) and one private infrastructure (our own data center). The public clouds are managed with Apache jclouds. We acquire VMs in the public cloud and then configure Docker to support the automatic provisioning of containers. In the private cloud, we resort to VMs running in a rack server. The substrate fabric switches of our data center are Pica8 P-3297, operating at 1 Gbps. The intra-cloud edges are created using GRE, and the clouds interconections are set up with OpenVPN tunnels.

5. Evaluation

The evaluation aims to answer several questions. First, we want to determine if our solution is efficient in using the substrate resources. Namely, regarding the acceptance ratio of virtual network requests, which will translate into profit for the multi-cloud

Table 1
VNR configurations that were evaluated.

Notation	Description
NS+NA	No security or availability demands on the VNRs
10S+NA	VNRs with 10% of resources (nodes and links) with security demands (excluding availability)
20S+NA	Like *10S+NA*, but with security demands for 20% of the resources
NS+10A	VNRs with no security demands, except for 10% of the nodes requesting replication
NS+20A	Like *NS+10A*, but for 20% of the nodes
20S+20A	20% of the resources (nodes and links) with security demands and 20% of the nodes with replication

provider. Second, we want to understand how the system scales, both concerning the enrichment of the substrate with cloud resources and the rate of arrival of VNRs. Additionally, we need to find out if Sirius handles well different kinds of topologies, including private, public, and hybrid clouds. Finally, we would like to measure the overhead introduced by the virtualization layer, and how it affects application performance.

We evaluate Sirius using large-scale simulations, comparing it with the two most commonly used heuristics: `D-ViNE`, which uses a relaxation of a MILP solution for node mapping and MCF for link mapping [18,26]; and the heuristics proposed by Yu et al. [13] that follow a greedy approach for node mapping and use MCF for link mapping (we label this solution `FG+MCF`). As MCF has scalability limitations, for this second approach we have also used the shortest path algorithm (`FG+SP`).

Besides, we evaluate the performance of our prototype over a multi-cloud substrate composed of a private data center and two public clouds (Amazon and Google), measuring the elapsed time to create various networks.

5.1. Testing environment

We extended an existing simulator [27] to collect various metrics about the embedding when a VNR workload arrives at the system.

Substrate networks. We employed two types of substrate network models: for public clouds, we utilized Waxman, where pairs of nodes are connected with a probability of 50% [28] (using the GT-ITM tool [29]); for the private data center, we created networks following the Google's Jupiter topology design [15]. For comparison with the optimal solution (Section 5.2), we considered a small scale substrate of 25 nodes. The reason is that the optimal solution we proposed in [14] does not scale to large networks. For the large-scale simulations (Section 5.3) we considered three substrate networks: `pub_substrate` - 100 nodes spread evenly in three clouds; `pvt_substrate` - 1900 nodes in one private data center; and `multi_substrate` - 2500 nodes spread in three clouds and a private data center. The CPU and bandwidth (cpu^S and bw^S) of nodes and links is uniformly distributed between 50 and 100 and 500 and 1000, respectively. Latencies inside a data center were set small ($lat^S \in \{1.0\}$), and between clouds were set larger ($lat^S \in \{50.0\}$), following empirical evidence we collected. These resources are also uniformly associated with one of three levels of security and trust ($sec^S \wedge cloud^S \in \{1.0, 1.2, 5.0\}$) in the public clouds, and one level ($sec^S \wedge cloud^S \in \{6.0\}$) in the private cloud. These values were chosen to achieve a good balance between the diversity of security levels and their monetary cost. The rationale for this choice was our analysis of the cost of Amazon EC2 instances with normal and secure VM configurations. These costs assume a wide range of values, related to the implemented defenses. For example, while an EC2 instance with content protection is around 20% more expensive than a normal instance (hence our choice of 1.2 for the intermediate level of security), the cost of instances with more sophisticated defenses are at least five times greater (our choice for the highest level of security).

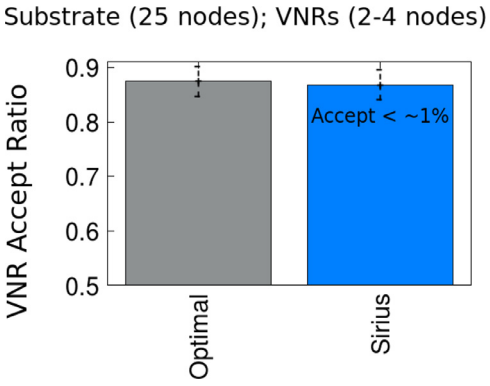


Fig. 3. VNR acceptance ratio: Sirius vs optimal.

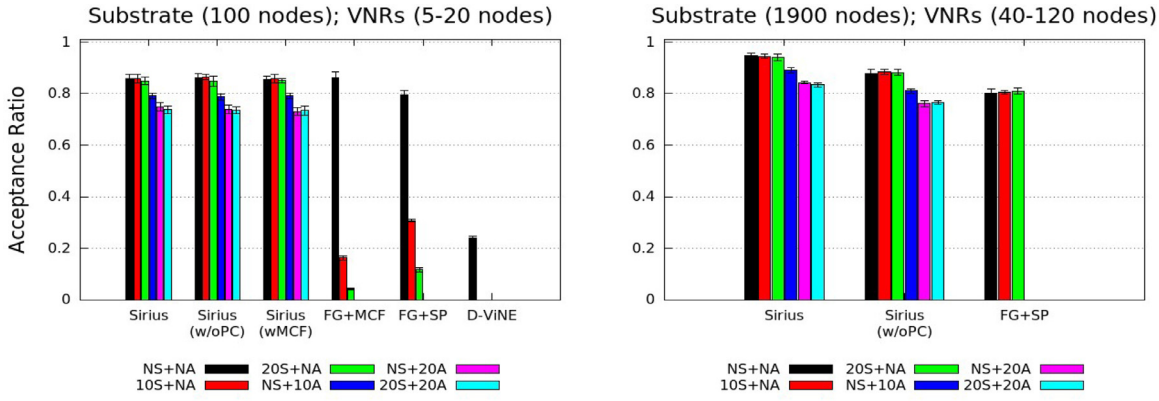
Virtual networks. VNRs have a number of virtual nodes uniformly distributed between 5 and 20 for the smaller scale setups, and between 40 and 120 for the larger-scale ones. Pairs of virtual nodes are connected with a Waxman topology with probability 50%. The CPU and bandwidth of the virtual nodes and links are uniformly distributed between 10 and 20, and 100 and 200, respectively. Several alternative security and availability requirements are evaluated, as shown in Table 1. We model VNRs arrivals as a Poisson process with an average rate of 4 VNRs per 100-time units for the first setup and 8 VNRs for the others. Each VNR has an exponentially distributed lifetime with an average of 1000 time units.

5.2. Evaluation against optimal solution

The goal of an heuristic is to compute good solutions fast, in order to obtain results that are close to the optimal, while enabling it to scale to very large networks. We thus start by asking how far is our heuristic from the optimal solution proposed in previous work [14]. For this purpose, we have considered a small scale substrate that, as explained above, consists of 25 nodes (the optimal solution [14] does not scale to larger values). We simulated 2000 VNRs with a number of virtual nodes that is uniformly distributed between 2 and 4 nodes. We compared the acceptance ratio of both solutions considering the VNR configuration NA+NS (recall Table 1). The results are presented in Fig. 3. The Sirius heuristic presents results that are very close to the optimal, with an acceptance ratio that is less than 1% lower than the optimal.

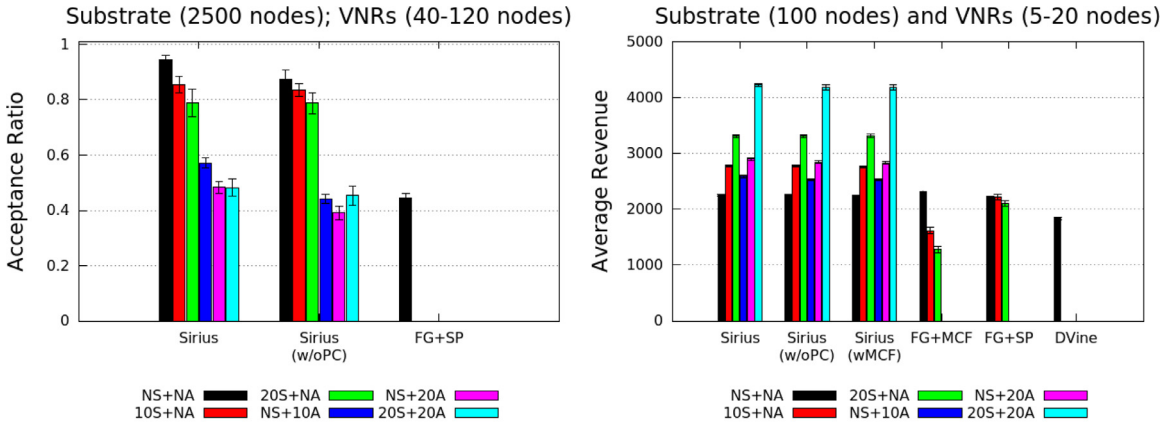
5.3. Large-scale simulations

We performed an extensive set of simulations to compare Sirius with the state-of-the-art VNE solutions. Figs. 4a,b, and 5a present the results for the Acceptance Ratio (AR) of the three scenarios under evaluation, respectively. We consider two variants of our approach to better assess the advantages of the mechanisms that form the overall design: `Sirius(w/oPC)` only employs the *UResSec*() utility function, and therefore does not take into consideration the length of the paths offered by *UPath*() (i.e., it is



(a) Public cloud (100 nodes). (b) Private DC (1900 nodes).

Fig. 4. Acceptance ratio: ratio of successful VNRs.



(a) AR in multi-cloud scenario (2500 nodes). (b) Average revenue (100 nodes).

Fig. 5. Acceptance ratio (multi-cloud scenario) and provider revenue.

the version *without* Path Contraction – recall Section 3.2.1); and `Sirius(wMCF)` using MCF for link mapping.

Acceptance ratio. In the case of VNRs with no security demands (`NS+NA`) in the smaller network (Fig. 4a), Sirius approaches behave similarly to `FG+MCF`, but increase the AR by 8% over `FG+SP` and present a significant 3-fold improvement over `D-ViNE`. The poor performance of `D-ViNE` is a result of its underlying model not fitting our specific multi-cloud environment. For instance, this solution considers geographical distance, which is not as relevant in a virtualized environment. Notice however that the results for D-Vine represent its best configuration with respect to geographical location – we have tested D-Vine with the entire range of options for this parameter. As the first conclusion, in the network topology offered by a public cloud (a full mesh), both `FG+MCF` and `Sirius` achieve good results.

As we introduce security demands, Sirius acceptance ratio decreases but only slightly. For instance, when 20% of all virtual elements request a level of security above the baseline, the reduction of the acceptance ratio is of only 1%. The same is true with requests that include availability, although the decrease is more pronounced (up to 13%). We expected this result, as replication needs not only to double node resources but also leads to an increase in the number of substrate paths (to maintain replica connectivity). The alternative solutions perform poorly with security requirements, as they do not consider these additional services. Therefore, most of the produced embeddings were rejected because they would violate at least one of the demands. We show no results in the case of `D-VINE` because, after more than one week of running this experiment, the algorithm had not yet finished. We also do not include results for the tests for availability because the algorithms we compare against do not consider the possibility of replication.

When observing Figs. 4b and 5a, the advantage of our approach is made clear. No results are included for algorithms with MCF for link mapping, as they take an extremely long time to complete. `Sirius` has an acceptance ratio 18% above `FG+SP` in the `pvt_substrate` (Fig. 4b), and of over 210% in the `multi_substrate` (Fig. 5a). These results demonstrate the effectiveness of our solution in improving the acceptance ratio over the alternatives for both virtualized datacenters and, even more strikingly, for a multi-cloud scenario. The main reason is our more detailed model, which incorporates different types of nodes (software and fabric switches), increasing the options available to map virtual nodes. The conclusions with respect to security are similar to above. One note, however, to explain why the results do not degrade with security in the `pvt_substrate` case, compared to the others. The reason is that in this experiment all nodes are considered of the highest security level, as they are inside the private data center. Another observation is that in some cases the average acceptance ratio is higher (despite still inside the confidence interval) with security demands, which can be counter-intuitive. The reason is that in some cases fulfilling security requirements tends to slightly better balance the substrate load.

Provider revenue. Next, we focus on the economical advantage for the multi-cloud provider. As in most previous work [13,18], we

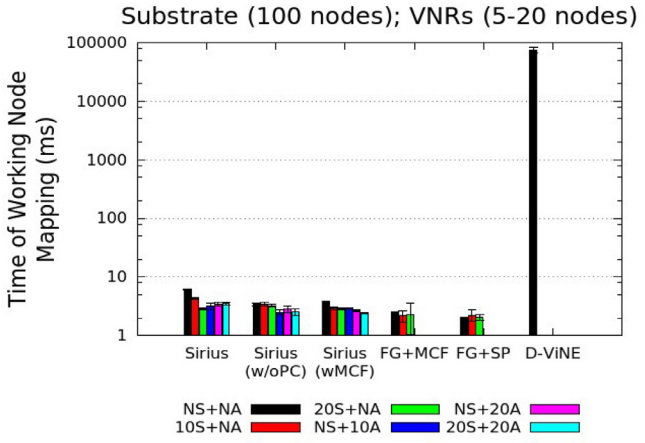


Fig. 6. Embedding time for node mapping.

assume the revenue of accepting a VNR is proportional to the acquired resources. However, in our case, we assume that security is charged at a higher premium value (in line with public cloud services). We calculate the revenue per VN as:

$$\mathbb{R}(\text{VNR}) = \lambda_1 \sum_{i \in N^V} [1 + \varphi_1(i)]\ cpu^V(i)\ \ sec^V(i)\ \ cloud^V(i) + \lambda_2 \sum_{(i,j) \in E^V} [1 + \varphi_2(i,j)]\ bw^V(i,j)\ \ sec^V(i,j),$$

where λ_1 and λ_2 are scaling coefficients that denote the relative proportion of each revenue component to the total revenue. These parameters offer providers the flexibility required to price differently the resources. Variables φ account for the need to have backups, either in the nodes $\varphi_1(i)$ or in the edges $\varphi_2(i, j)$.[4] In the experiments, we set $\lambda_1 = \lambda_2 = 1$.

Fig. 5b presents the average revenue generated by embedding VNRs in `pub_substrate`. The main conclusion is that Sirius generally improves the profit of the multi-cloud provider. First, revenue is enhanced when we include security, which gives incentives for providers to offer value-added services. Second, availability can even have a stronger impact because more resources are used to satisfy VNRs

Scalability. We now turn our attention to system scalability, with a focus on embedding latency, as this metric translates into the attainable service rate for virtual network requests. The measurements use code that is equivalent to the one used in our network hypervisor (for our approach, it is the same java implementation). Figs. 6 and 7 present the time to map nodes and links, respectively. As can be seen, `D-VINE` scales very poorly in both phases, while `Sirius`, `Sirius(w/oPC)`, and `FG+SP` behave better. With mappings taking in the order of tens of ms, these solutions enable embedding hundreds to thousands of virtual elements per second. The time to embed backup elements is of the same order of magnitude (we omit the graph for space reasons). Finally, as the substrate and the size of the virtual networks grow, the embedding latency increases accordingly. Fig. 7b displays the worst case of our experiments: the time for link mapping in the `multi_substrate`. For such large-scale network, embedding increases to around 60 seconds per virtual network. In summary, `Sirius`, `Sirius (w/oPC)`, and `FG+SP` are the only embedding solutions that scale to reasonable numbers in the context of a realistic network virtualization system.

[4] $\varphi_1(i) = 1$ if a backup is required, or 0 otherwise; $\varphi_1(i, j) = 1$, in case at least one node needs a backup, or 0 otherwise.

To conclude, we look into the benefits brought by the Path Contraction (PC) function *UPath*(). By "contracting" path lengths, Sirius requires fewer substrate links. This can be confirmed in Fig. 8a, which shows the total number of substrate links utilized in the private data center topology (similar results were obtained for the other scenarios). Even when comparing with `FG+SP`, a scheme that resorts to shortest path, Sirius ends up performing better. The reason is that our heuristic couples the two phases of embedding by bringing neighboring nodes closer to each other. As a consequence, there was the expectation of increasing embedding efficiency and improving application performance by reducing network latency. Indeed, the PC mechanism enhances the embedding acceptance ratio (see Figs. 4b and 5a). Moreover, *UPath*() decreases the distance between virtual nodes, measured as the number of hops between their corresponding substrate nodes ("path length"). Fig. 8b illustrates, for one representative simulation run, how path lengths are significantly decreased. On average, we observed a 26% reduction on this metric.

5.4. Prototype experiments

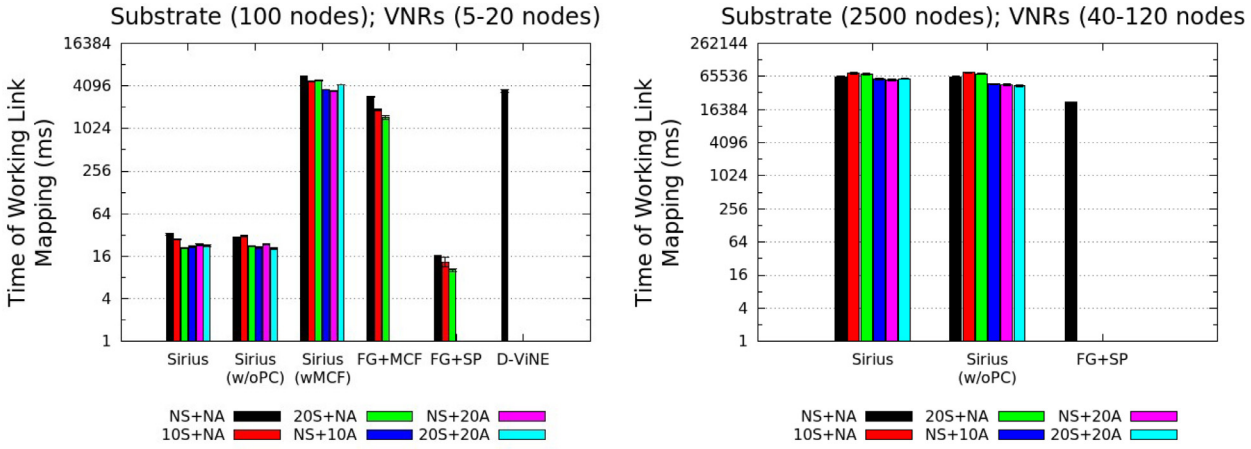
We now turn to evaluate Sirius by running real experiments with our prototype. We have set up a multi-cloud substrate, composed of 2 public clouds (Google and Amazon), and a private data center (our cluster in Lisbon). In Section 5.4.1, we test the performance of the prototype, including the time to set up the substrate, the time to provision virtual networks, and data plane performance (regarding latency and throughput). Then, in Section 5.4.2, we compare Sirius running our embedding algorithm against the alternative of integrating other state-of-the-art algorithms into Sirius.

5.4.1. Prototype performance

Substrate setup time. Fig. 9 shows the time to set up a substrate with VMs distributed through the three clouds. Since we perform most operations in parallel, it is possible to observe only a small increase in time when the number of VMs more than doubles, from 10 to 25. The slowest operation is VM configuration, which includes the time for software installation (e.g., Docker) and getting a basic container image. The second most relevant delay is VM provisioning by the cloud provider. Overall, the added cost of our solution is small, as it involves only the setup of the required tunnels.

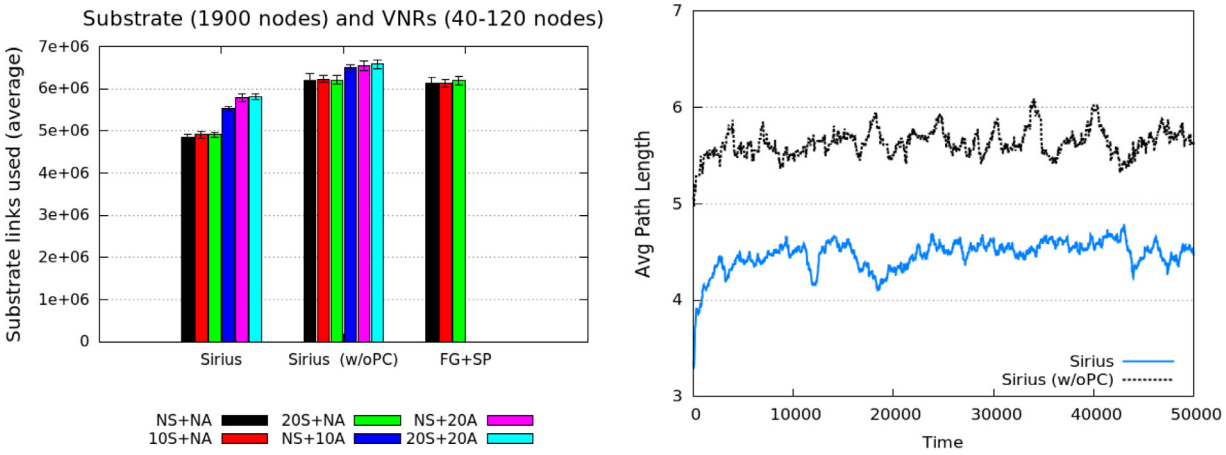
Virtual network provisioning. Substrate provisioning represents a one-off cost in terms of setup time, and therefore it does not have an impact on user experience. By contrast, the time to provision virtual networks represents the operating run-time cost that directly impacts the user, and so it requires particular care. In Fig. 10b we present the provisioning time for VNs of different sizes in number of virtual hosts (containers), on substrates with distinct numbers of substrate compute nodes (VMs). As the reader can observe, we present two versions in the figure: our first implementation ("baseline") and the final, optimized version. In both cases, the rapid increase in the elapsed time from 1k to 4k containers, and from 10k to 25k containers, is due to the rise in the number of containers deployed per VM (from 100 to 400 in the former, and from 400 to 1000 in the latter). In all cases, our embedding procedure represents a relatively insignificant fraction of the overall provisioning time (less than 3%).

The superlinear increase in provisioning time for large VNs has motivated us to investigate performance optimizations. In our baseline implementation, container configuration represented by far the largest fraction of this time (over 75%). We thus made a closer inspection of this provisioning stage, shown in Fig. 10a. Container configuration consists of running a customized version of the *ovs-docker* script — the Open vSwitch utility that enables



(a) Public cloud (100 nodes). (b) Multi-cloud (2500 nodes).

Fig. 7. Embedding time for link mapping.



(a) Number of embedded links (1900 nodes). (b) Average path length.

Fig. 8. The effect of coupling node and link mapping with Path Contraction.

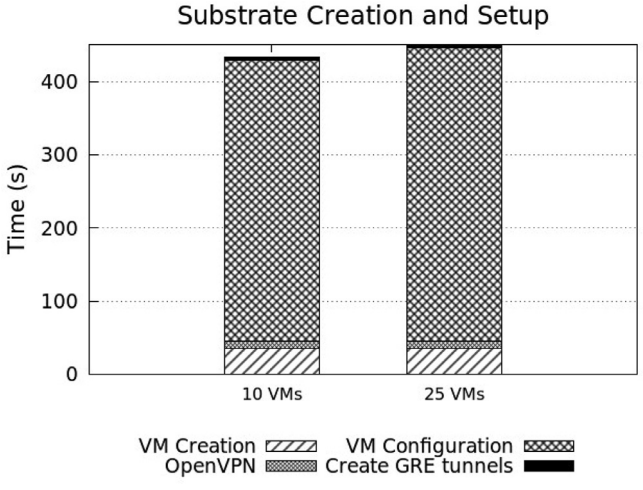
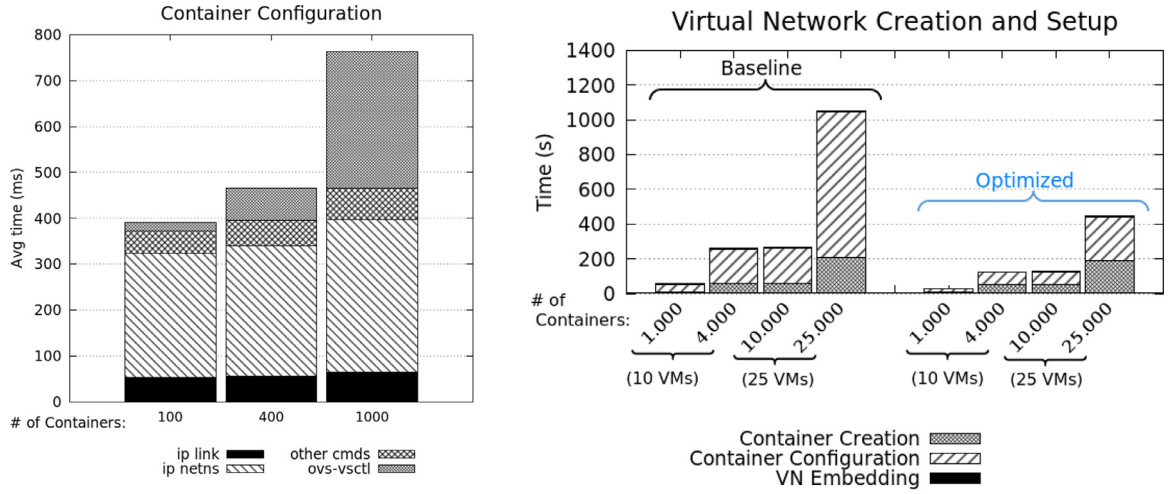


Fig. 9. Substrate setup time.

Docker networking. Most of the processing is spent on three tasks: creating virtual container interfaces with *ip link*, setting up the networking name space with *ip netns*, and link the interfaces with a specific OVS switch port with *ovs-vsctl add-port*. Fig. 10a shows that the *ovs-vsctl* command, in particular, scales poorly, with script execution time exhibiting a growth rate that explains the superlinear increase in configuration time reffered to above. We thus realized the bottleneck of the baseline version to be the use of sequential calls to the script within a single process, a problem that becomes more severe with larger networks. As an optimization of the configuration process, we execute several instances of the *ovs-docker* script in parallel. To avoid connection issues (related to the maximum number of open sockets), we bound the number of concurrent calls to 10 in our implementation.

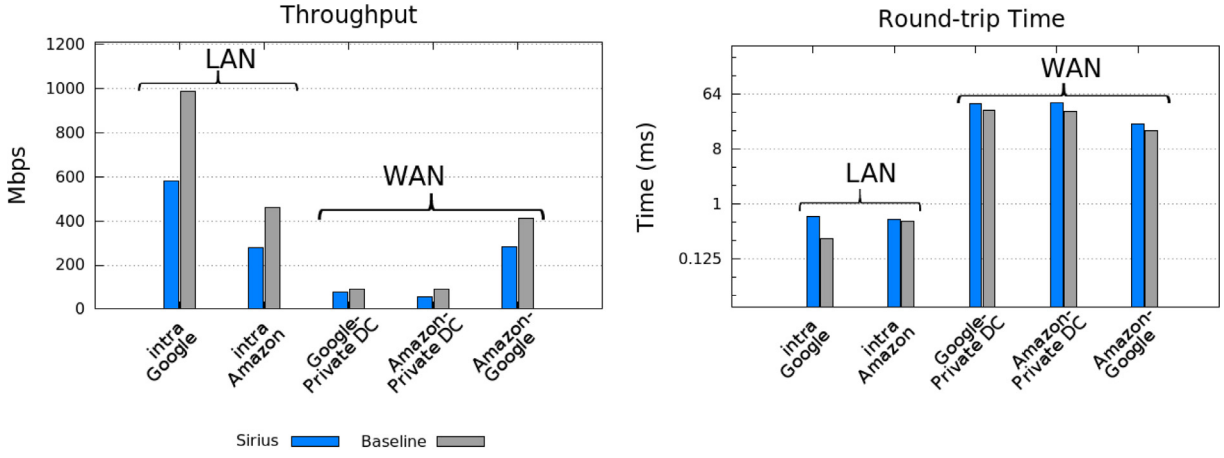
This optimization of the configuration procedure resulted in a significant gain of between 2x and 2.5x in virtual network provisioning time. As shown in Fig. 10b, the optimized version of Sirius is able to provision a virtual network of 10k containers in less than 2 minutes, more than halving the result from the baseline. As the scale of the VN increases so does the gain, which is close to 2.5x for the larger, 25-thousand hosts virtual network.

Throughput and latency. Fig. 11 presents the cost of virtualization with respect to latency and throughput, using as baseline a VM configuration that accesses the network directly. Inter-cloud RTTs raise by around 30% (i.e., between 5 and 10ms), and intra-cloud RTTs increase by less than 400us. As inter-cloud applications typically assume latencies of this magnitude in their design, and



(a) Container configuration in-depth. (b) Virtual network provisioning.

Fig. 10. Virtual network provisioning time.



(a) Throughput. (b) Latency.

Fig. 11. Prototype measurements: intra- and inter-cloud throughput and latencies.

the added intra-cloud cost is small, this overhead is arguably acceptable. Throughput decreases further, with larger costs when the baseline is high, as expected [1]. We are currently investigating networking-enhanced VM instances to reduce this overhead.

5.4.2. Comparison against alternative embedding algorithms

In this section, we compare the embedding algorithm employed in Sirius against two alternatives: the optimal solution presented in [14], and the commonly-used greedy approach using MCF for path selection, proposed in [13]. For this purpose, we have implemented these two embedding algorithms and integrated them into Sirius.

The goal of this experiment is to measure the time it takes for the embedding algorithm to report a solution. In addition, we also want to measure its success rate, as in some cases the algorithm may fail to find a solution. We have considered the multi-cloud substrate composed of three clouds explained before, with the following setup (presented at the bottom of Fig. 12). We have set up 9 VM instances at Google as *substrate compute* elements (recall Fig. 1), each with the following characteristics: 16Gb RAM, four vCPUs, and all running Ubuntu Server 16.04 LTS. In Sirius, one of these instances is the gateway, which we set with the role of *fabric switch* (node G in the figure), to which all other VMs are connected. We recall that in our model the fabric switch cannot have any *virtual host* attached, and so is able only to function as *virtual transit switch*. The other VMs ($b \sim i$) run a *software switch*, with several containers running on top. At Amazon we run 10 VM instances, with the same characteristics as above. Again, one (node A) is set with the role of fabric switch, while the others ($j \sim r$) run a software switch and several containers. Our private data center in Lisbon includes one OpenFlow switch (fabric switch, node L) and one bare metal physical server (node a) with 32 Gb RAM, 8 CPUs, running Ubuntu Server 14.04 LTS. The server hosts a single VM as substrate compute element, running a software switch, with several containers connected. The fabric switches are all interconnected. In all clouds, we model each software switch with 100 CPU units.

Our experiment consists in emulating nine sequential VNRs, made by different tenants. We consider three virtual topologies, and three requests for each topology, done in sequence. Namely, the first request is for a big switch topology; the second, a Virtual cluster; and the third is for an inter-network (top of Fig. 12). We repeat this sequence three times. We have chosen these topologies as they are ubiquitous and representative of different applications. The first two, the big switch and the virtual cluster, are taken from [30]. The first is a single switch to which several hosts connect, in a star topology. The second topology consists of

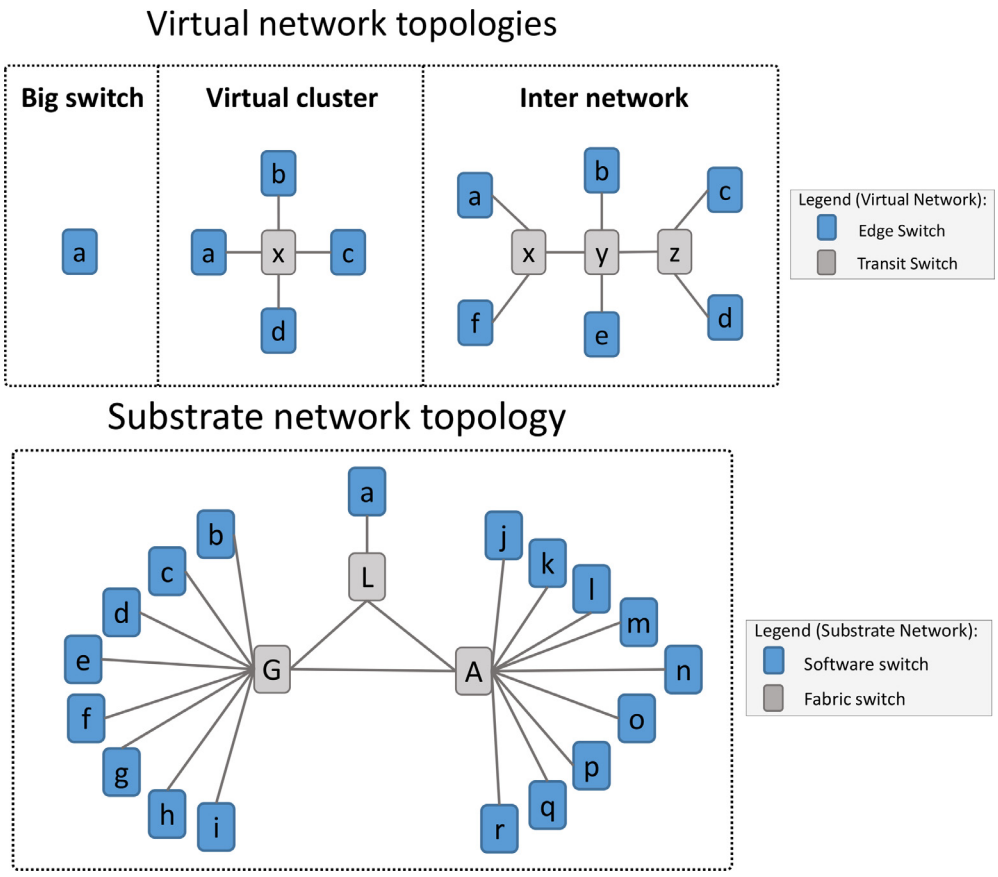


Fig. 12. Virtual (top) and Substrate (bottom) topologies for experiments considering three embedding algorithms: Sirius, full-greedy [13], and the optimal solution [14].

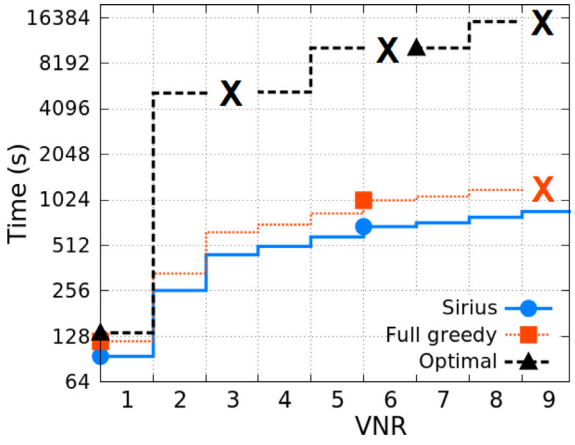


Fig. 13. Embedding time for 9 sequential VNRs.

multiple virtual clusters connected to a central switch, over links that are typically oversubscribed. The third topology represents a typical three-tier topology commonly used in web workloads, where a layer of load balancers connects to a layer of application servers, which then connect to another layer of database servers. In all the topologies considered, each *edge switch* runs a set of containers. In each VNR, one virtual switch running several virtual hosts has a CPU requirement of 50 units.

Fig. 13 presents the results of our experiment. The plot shows the embedding time of the three algorithms considered. The symbol **X** marks the cases when the algorithm failed to return a solution. As expected, the optimal solution is very slow. This fact is apparent in all cases, with each embedding taking many seconds to return a result, but is made particularly evident in the inter-network case. The three VNRs requesting this topology have failed due to a timeout.[5] Sirius is also faster than the full greedy approach, due to the latter using MCF for mapping the virtual links. It is also possible to observe that full greedy has failed to find a solution to the last request, whereas Sirius mapped all requests. This result illustrates the advantage of our enhanced model that distinguishes between fabric and software switches. Our solution makes better use of the substrate resources, as it can map virtual transit switches to fabric switches. As the full greedy approach from [13] does not make this distinction, it needs to map all switches (edge and transit) to software switches. As a result, it does not have enough resources to fulfill the last request.

6. Related Work

Cloud networking. The performance of a cloud-based application critically depends on the network that inter-connects its computational elements. The enhancement of cloud services with network guarantees has therefore been an active area of research in the past decade. One of the first attempts was SecondNet [31], a Virtual Data Center abstraction that provides cloud tenants with bandwidth guarantees for pairs of VMs. The authors of Oktopus [30] went further, proposing two new abstractions: the virtual cluster – a virtual switch interconnecting multiple VMs with bandwidth guarantees for all virtual links, and

[5] We set the timeout to 1-hour.

the virtual oversubscribed cluster – an oversubscribed two-tier cluster that suits applications featuring local communication patterns. Faircloud [32] and EyeQ [33] improved by considering work-conserving bandwidth allocations, and Jang et al. [34] by providing bounded packet delay. Follow-up work extended these models with dynamic traffic patterns [35], improved embedding algorithms [36], and better scaling properties [37,38]. These works extend the cloud model with network guarantees, but none offers network virtualization. In particular, they do not allow for arbitrary virtual network topologies.

Network virtualization. The growth of cloud computing and the large scale of the data center networks that enabled it has left cloud operators with a difficult problem: to choose between networking flexibility (Ethernet) or scalability and efficiency (IP). SEATTLE [39] was among the first solutions to marry the plug-and-play functionality and ease of management of Ethernet networks, by employing flat addressing, with the high scalability and efficiency of shortest-path IP-based routing (enabled by hash-based resolution of host information). VL2 [40] and PortLand [17] have improved over this initial work by exploring scalable data plane primitives, such as multi-path, and by addressing some of the scaling limitations of SEATLLE (namely by avoiding the use of broadcasts). While these works significantly improved the status quo, they did not provide *complete* network virtualization – the ability to fully decouple the virtual network from the underlying substrate. It was only with the emergence of Software-Defined Networking (SDN) [41] that became feasible to provide this form of network virtualization. Modern network virtualization platforms, including VMware NVP [1], Microsoft AccelNet [2], and Google Andromeda [3], are edge solutions based on SDN to offer complete network virtualization to its tenant applications. Contrary to our work, these modern platforms do not consider a multi-cloud substrate. In addition, the papers do not explain how the embedding problem is addressed and do not describe the incorporation of security or availability services.

Virtual network embedding. While efficient greedy heuristics already existed for the node mapping phase of the Virtual Network Embedding (VNE) problem, Yu et al. [13] where the first to solve the link mapping problem efficiently. The authors assumed a substrate that is path splitting-capable, enabling solving the problem as a multicommodity flow (MCF). MCF improved the situation but, as we have argued, is not a good fit for production-quality systems. In [18], Chowdhury et al. proposed an approach for VNE that introduced coordination between the node and link mapping phases to improve the acceptance ratio. As we show in Section 5.3, this solution scales very poorly. Besides, none of these works considers security or availability.

Survivable VNE. More recently, Yu et al. [21] have proposed to improve the availability of VNE by extending existing algorithms with node redundancy. Others [19,20] have proposed VNEs that allow recovering from substrate link failures. Rahman et al. [20] have considered single link failures, whereas Shahriar et al. [19] went further to consider the presence of multiple substrate link failures. Contrary to our work, these set of proposals target only availability.

Security in VNE. A relatively less well-explored perspective on the VNE problem is providing security guarantees. Fischer et al. [42] have introduced this problem, considering the inclusion of constraints in the VNE formulation that consider security. The authors proposed the assignment of security levels to every physical resource, allowing virtual network requests to include security demands. This position paper has only enunciated the problem at a high level and has not proposed any solution or algorithm. Bays et al. [22] and Liu et al. [23,43] have afterward proposed VNE algorithms based on this idea. However, the authors of [22] consider only link protection. Liu et al. go further, by also considering node security. However, they also do not consider availability nor a multi-cloud setting with different trust domains. Also, their model makes a few assumptions that make it unfeasible for a realistic network virtualization platform. First, it requires virtual nodes to fulfill a specific security level, demanded by the physical host. In practice, a virtualization platform cannot assume such level of trust about its guests. Second, it assumes that the duration of a virtual network request is known beforehand. This limitation reduces its applicability in a traditional pay-as-you-go model, as a cloud tenant typically does not know in advance the duration of its requests. We make none of these assumptions in our work. In [14] we have proposed a MILP solution for this problem, considering a multi-cloud network substrate. As we explained in Section 5.2, that solution scales very poorly.

Multi-provider VNE. Most VNE work considers a single (intra-domain) substrate infrastructure. One of the early exceptions was PolyVINE [44], a policy-based inter-domain VN embedding framework that embeds end-to-end VNs. However, the goal of this work is to coordinate policies among inter-domains, a different problem from ours. Other examples include the works by Houidi et al. [45] and Dietrich et al. [46]. These solutions considered a multiple substrate in the embedding, and address some important albeit orthogonal problems that could be incorporated into our Sirius (e.g., [46] addresses the multi-domain VN embedding problem with limited information disclosure, which is of high relevance to VN providers). None of these works consider node and link security, nor availability. In addition, they follow the "traditional" network model used in most embedding work, considering only one type of node. Our multi-cloud approach follows a virtualized datacenter model that differentiates between two types of virtual nodes: virtual hosts and virtual switches. As we explain in Section 5.3, this is key to achieve high efficiency and higher acceptance ratios over the traditional models. One common limitation of all the VNE work we presented in this section is that it focuses solely on the embedding problem and/or presents generic frameworks – they do not build a concrete system.

Multi-cloud systems. The multi-cloud model was successfully applied in the context of computation [47] and storage [48]. One common goal is to improve system dependability. Examples include MapReduce [49], coordination [50], and file systems [48]. To the best of our knowledge, Sirius is the first system to apply this model to network virtualization.

7. Conclusions

In this paper, we presented the design and implementation of Sirius, a multi-cloud network virtualization platform. Our solution improves the state-of-the-art by extending the substrate network with cloud services and enhancing virtual networks with security and dependability.

Evaluations of our prototype in large-scale simulations reveal that, compared with the state-of-the-art alternatives, our solution scales well, increases the acceptance ratio and the provider profit for diverse topologies, maintaining short path lengths to guarantee application performance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the EC through project SUPERCLOUD (H2020-643964), and by national funds of

FCT with ref. UID/CEC/00408/2019 (LASIGE) and ref. PTDC/CCI-INF/30340/2017 (uPVN project).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2019.06.004.

References

[1] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, R. Zhang, Network Virtualization in Multi-Tenant Datacenters, in: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 203–216. http://dl.acm.org/citation.cfm?id=2616448.2616468.

[2] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H.K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D.A. Maltz, A. Greenberg, Azure Accelerated Networking: Smartnics in the Public Cloud, 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018.

[3] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E.C. Zermeno, E. Rubow, J.A. Docauer, J. Alpert, J. Ai, J. Olson, K. DeCabooter, M. de Kruijf, N. Hua, N. Lewis, N. Kasinadhuni, R. Crepaldi, S. Krishnan, S. Venkata, Y. Richter, U. Naik, A. Vahdat, Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization, 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018.

[4] U. Today, Massive Amazon cloud service outage disrupts sites, 2017. https://www.usatoday.com/story/tech/news/2017/02/28/amazons-cloud-service-goes-down-sites-scramble/98530914/ (February).

[5] G.C. Platform, Google compute engine incident 16007, 2016. https://status.cloud.google.com/incident/compute/16007 (April).

[6] C. Insiders, Cloud security report, 2018.

[7] VMWare, nsx data center (2018).

[8] L. Zheng, C. Joe-Wong, C.W. Tan, M. Chiang, X. Wang, How to Bid the Cloud, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, ACM, New York, NY, USA, 2015, pp. 71–84, doi:10.1145/2785956.2787473.

[9] P. Sharma, D. Irwin, P. Shenoy, How Not to Bid the Cloud, in: Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, HotCloud'16, USENIX Association, Berkeley, CA, USA, 2016, pp. 1–6. http://dl.acm.org/citation.cfm?id=3027041.3027042.

[10] Rightscale, state of the cloud report, 2017. February.

[11] M. Ben-Yehuda, M.D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, B.A. Yassour, The Turtles Project: Design and Implementation of Nested Virtualization, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 423–436. http://dl.acm.org/citation.cfm?id=1924943.1924973.

[12] A. Fischer, J.F. Botero, M.T. Beck, H. de Meer, X. Hesselbach, Virtual network embedding: a survey, IEEE Commun. Surv. Tutor. 15 (4) (2013) 1888–1906, doi:10.1109/SURV.2013.013013.00155.

[13] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 17–29, doi:10.1145/1355734.1355737.

[14] M. Alaluna, L. Ferrolho, J.R. Figueira, N. Neves, F.M.V. Ramos, Secure multi-cloud virtual network embedding, CoRR abs/1703.01313. http://arxiv.org/abs/1703.01313.

[15] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, A. Vahdat, Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, ACM, New York, NY, USA, 2015, pp. 183–197, doi:10.1145/2785956.2787508.

[16] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, B. Snow, Openvirtex: Make Your Virtual Sdns Programmable, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 25–30, doi:10.1145/2620728.2620741.

[17] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM '09, ACM, New York, NY, USA, 2009, pp. 39–50, doi:10.1145/1592568.1592575.

[18] M. Chowdhury, M.R. Rahman, R. Boutaba, Vineyard: virtual network embedding algorithms with coordinated node and link mapping, IEEE/ACM Trans. Netw. 20 (1) (2012) 206–219, doi:10.1109/TNET.2011.2159308.

[19] N. Shahriar, R. Ahmed, S.R. Chowdhury, M.M.A. Khan, R. Boutaba, J. Mitra, F. Zeng, Connectivity-Aware Virtual Network Embedding, in: 22016 IFIP Networking Conference (IFIP Networking) and Workshops, 2016, pp. 46–54, doi:10.1109/IFIPNetworking.2016.7497249.

[20] M.R. Rahman, R. Boutaba, Svne: survivable virtual network embedding algorithms for network virtualization, IEEE Trans. Netw. Serv. Manage. 10 (2) (2013) 105–118, doi:10.1109/TNSM.2013.013013.110202.

[21] H. Yu, V. Anand, C. Qiao, G. Sun, Cost Efficient Design of Survivable Virtual Infrastructure to Recover from Facility Node Failures, in: 2011 IEEE International Conference on Communications (ICC), 2011, pp. 1–6, doi:10.1109/icc.2011.5962604.

[22] L.R. Bays, R.R. Oliveira, L.S. Buriol, M.P. Barcellos, L.P. Gaspary, Security-aware Optimal Resource Allocation for Virtual Network Embedding, in: Proceedings of the 8th International Conference on Network and Service Management, CNSM '12, International Federation for Information Processing, Laxenburg, Austria, Austria, 2013, pp. 378–384. http://dl.acm.org/citation.cfm?id=2499406.2499466.

[23] S. Liu, Z. Cai, H. Xu, M. Xu, Security-Aware Virtual Network Embedding, in: 2014 IEEE International Conference on Communications (ICC), 2014, pp. 834–840, doi:10.1109/ICC.2014.6883423.

[24] B. Pfaff, J. Pettit, T. Koponen, E.J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The Design and Implementation of Open Vswitch, in: Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, USENIX, Association, Berkeley, CA, USA, 2015, pp. 117–130. http://dl.acm.org/citation.cfm?id=2789770.2789779.

[25] VIS.JS, 2017. http://visjs.org/, accessed: 2017-10-25.

[26] N.M.M.K. Chowdhury, M.R. Rahman, R. Boutaba, Virtual Network Embedding with Coordinated Node and Link Mapping, in: IEEE INFOCOM 2009, 2009, pp. 783–791, doi:10.1109/INFCOM.2009.5061987.

[27] ViNE-yard, http://www.mosharaf.com/ViNE-Yard.tar.gz.

[28] M. Naldi, Connectivity of waxman topology models, Comput. Commun. 29 (1) (2005) 24–31, doi:10.1016/j.comcom.2005.01.017.

[29] E.W. Zegura, K.L. Calvert, S. Bhattacharjee, How to Model an Internetwork, in: Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies Conference on The Conference on Computer Communications - Volume 2, INFOCOM'96, IEEE Computer Society, Washington, DC, USA, 1996, pp. 594–602. http://dl.acm.org/citation.cfm?id=1895868.1895900.

[30] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, Towards Predictable Datacenter Networks, in: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, ACM, New York, NY, USA, 2011, pp. 242–253, doi:10.1145/2018436.2018465.

[31] C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, Secondnet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees, in: Proceedings of the 6th International COnference, Co-NEXT '10, ACM, New York, NY, USA, 2010, pp. 15:1–15:12, doi:10.1145/1921168.1921188.

[32] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, I. Stoica, Faircloud: Sharing the Network in Cloud Computing, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, 2012.

[33] V. Jeyakumar, M. Alizadeh, D. Maziéres, B. Prabhakar, A. Greenberg, C. Kim, Eyeq: Practical Network Performance Isolation at the Edge, Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013.

[34] K. Jang, J. Sherry, H. Ballani, T. Moncaster, Silo: Predictable Message Latency in the Cloud, in: SIGCOMM '15,

[35] D. Xie, N. Ding, Y.C. Hu, R. Kompella, The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, ACM, New York, NY, USA, 2012, pp. 199–210, doi:10.1145/2342356.2342397.

[36] M. Rost, C. Fuerst, S. Schmid, Beyond the stars: revisiting virtual cluster embeddings, SIGCOMM Comput. Commun. Rev. 45 (3) (2015) 12–18, doi:10.1145/2805789.2805792.

[37] C. Fuerst, S. Schmid, L. Suresh, P. Costa, Kraken: Online and Elastic Resource Reservations for Multi-tenant Datacenters, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9, doi:10.1109/INFOCOM.2016.7524466.

[38] L. Yu, Z. Cai, Dynamic Scaling of Virtual Clusters with Bandwidth Guarantee in Cloud Datacenters, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9, doi:10.1109/INFOCOM.2016.7524355.

[39] C. Kim, M. Caesar, J. Rexford, Floodless in Seattle: A Scalable Ethernet Architecture for Large Enterprises, in: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM '08, ACM, New York, NY, USA, 2008, pp. 3–14, doi:10.1145/1402958.1402961.

[40] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, Vl2: A Scalable and Flexible Data Center Network, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM '09, ACM, New York, NY, USA, 2009, pp. 51–62, doi:10.1145/1592568.1592576.

[41] D. Kreutz, F.M.V. Ramos, P.E. Verí-ssimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: a comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76, doi:10.1109/JPROC.2014.2371999.

[42] A. Fischer, et al., Position Paper: Secure Virtual Network Embedding, in: Praxis der Informationsverarbeitung und Kommunikation.

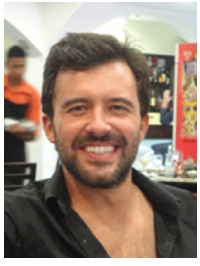
[43] S. Liu, Z. Cai, H. Xu, M. Xu, Towards security-aware virtual network embedding, Comput. Netw. 91 (C) (2015) 151–163, doi:10.1016/j.comnet.2015.08.014.

[44] M. Chowdhury, F. Samuel, R. Boutaba, Polyvine: Policy-based Virtual Network Embedding across Multiple Domains, in: Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, VISA '10, ACM, New York, NY, USA, 2010, pp. 49–56, doi:10.1145/1851399.1851408.
[45] I. Houidi, W. Louati, W.B. Ameur, D. Zeghlache, Virtual network provisioning across multiple substrate networks, Comput. Netw. 55 (4) (2011) 1011–1023, doi:10.1016/j.comnet.2010.12.011.
[46] D. Dietrich, A. Rizk, P. Papadimitriou, Multi-provider virtual network embedding with limited information disclosure, IEEE Trans. Netw. Serv. Manage. 12 (2) (2015) 188–201, doi:10.1109/TNSM.2015.2417652.
[47] D. Williams, H. Jamjoom, H. Weatherspoon, The Xen-blanket: Virtualize Once, Run Everywhere, in: Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys '12, ACM, New York, NY, USA, 2012, pp. 113–126, doi:10.1145/2168836.2168849.
[48] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, P. Verissimo, Scfs: A Shared Cloud-backed File System, in: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, USENIX ATC'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 169–180. http://dl.acm.org/citation.cfm?id=2643634.2643652.
[49] P.A.R.S. Costa, F.M.V. Ramos, M. Correia, Chrysaor: Fine-Grained, Fault-Tolerant Cloud-of-Clouds Mapreduce, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2017, pp. 421–430, doi:10.1109/CCGRID.2017.89.
[50] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa, Depsky: dependable and secure storage in a cloud-of-clouds, Trans. Storage 9 (4) (2013) 12:1–12:33, doi:10.1145/2535929.

Max Alaluna is a Ph.D. student in the Dept of Computer Science, Faculty of Sciences of the University of Lisboa. He received the B.Sc. degree in computer engineering and M.Sc. degree in computing and systems from Military Institute of Engineering, Rio de Janeiro, Brazil in 2004 and 2011, respectively. He is a member of the Navigators research group (www.navigators.di.fc.ul.pt) and the LASIGE research unit (www.lasige.di.fc.ul.pt). He participated in the H2020 SUPERCLOUD project, where he developed a multi-cloud network virtualization platform. His main research interests are network virtualization, security and dependability.

Eric Vial is an integration engineer at the Faculty of Sciences, University of Lisbon. He is a member of the Navigators research group and the LASIGE research unit. After a degree in telecommunications from the ENST-Bretagne school and a 10-year experience as a network project manager, he's been working on the design and implementation of several testbed platforms at the university.

Nuno Neves is Professor at the Department of Computer Science, Faculty of Sciences, University of Lisbon. His research interests are in security and dependability aspects of distributed systems and networks. He is co-principal investigator of two research projects, one on vulnerability discovery on healthcare systems (SEAL) and another on programmable networks (uPVN). He also coordinates a cooperation project on secure smart grids. His work was recognized in several occasions, e.g., with the IBM Scientific Prize and the William C. Carter award at IEEE DSN. He currently has more than 130 publications in journals and conferences. (Home page: http://www.di.fc.ul.pt/~nuno).



Fernando M.V. Ramos is an assistant professor in the Department of CSE at the Faculty of Sciences, University of Lisbon. He is a member of the Navigators research group and the LASIGE research unit. Prior to that he was with the University of Cambridge, ISEL, Telefonica Research, University of Aveiro, and PT Innovation. He has a PhD from the University of Cambridge. Fernando is currently Principal Investigator of the uPVN project. He is a senior member of the IEEE, and member of the ACM and the MEF research council. His research interests lie at the intersection of networking, systems, and security.