Multi-PoP Network Slice Deployment: A Feasibility Study

Polychronis Valsamas*, Panagiotis Papadimitriou*, Ilias Sakellariou*, Sophia Petridou*, Lefteris Mamatas*, Stuart Clayman†, Francesco Tusa† and Alex Galis†

**Department of Applied Informatics*
University of Macedonia
Thessaloniki, Greece
{*xvalsama, papadimitriou, iliass, spetrido, emamatas*}*@uom.edu.gr*

†*Department of Electronic and Electrical Engineering*
University College London
London, UK
{*s.clayman, francesco.tusa, a.galis*}*@ucl.ac.uk*

***Abstract*—Network slicing is seen as a key enabler for meeting the diverse network service requirements, which stem from the transition to 5G. Furthermore, network slicing provides inherent support for multi-tenancy, enabling network providers to slice their infrastructure and resell it to a large number of tenants. Most existing work on slicing has been focused on certain mechanisms (*e.g.,* slice embedding) and architecture specifications. As such, the performance and scalability with network slice instantiation has not been studied in depth. These aspects are even more critical in the case of slice deployments across multiple Points-of-Presence (PoP), since the various slice components should be stitched together for the end-to-end slice instantiation. In this paper, we present the design and prototype implementation of a network slicing architecture, based on which we perform a feasibility study of network slicing using multiple experimental infrastructures. Our prototype implementation supports all the required functionality for slice instantiation, such as resource discovery, slice embedding, resource provisioning, link setup, and inter-PoP slice segment stitching. Our experimental results corroborate the feasibility of multi-PoP network slicing. We further gain useful insights on slice instantiation performance and scalability.**

I. Introduction

Network slicing has recently been promoted as a key enabler for leasing service-tailored bundles of computing, network, and storage resources, which are often termed as verticals [1]–[3]. This essentially fosters the integration of existing and novel network services into the infrastructure, as well as enabling the co-existence of multiple services with significantly different requirements in terms of bandwidth, delay, resilience and/or security. Network Function Virtualization (NFV) [4]–[6] and Software-Defined Networking (SDN) [7], [8] comprise some of the main enablers for network slicing.

Network slicing is usually not limited to only provisioning isolated resource bundles, but it also encompasses resource management and orchestration primitives on per-slice-basis. As such, tenants are allowed to exercise fine-grained control and management on their leased slices, with minimal provider interventions. This level of control can be attained, *e.g.,* by deploying a dedicated *Virtualized Infrastructure Manager* (VIM) per slice, as advocated by the NECOS project [9]. Alternatively, independent slice control could be enabled through a shim layer on top of a shared VIM. Since the latter introduces high complexity, we assume a *VIM-on-demand* slicing model in the rest of this paper [10].

In practice, the concurrent deployment and operation of network slices on top of shared infrastructures poses the need for diverse functionality spread across different layers of the network slicing architecture. For example, network slicing requires mechanisms to advertise, discover, select, and allocate resources for slice creation. In the case of slices spanning multiple Points-of-Presence (PoP) or different administrative domains, additional mechanisms are required for stitching together the different slice segments in order to instantiate the slice. While the efficiency of certain slicing mechanisms, *e.g.,* slice embedding [11], is well understood, the feasibility of a multi-PoP slice deployment has not been studied so far, to the best of our knowledge. Existing prototype implementations mainly pertain to network (function) virtualization [12], [13], and are typically limited to single-PoP deployments.

Along these lines, we conduct a feasibility study of multi-PoP network slice deployment to assess the timescales at which slices can be provisioned and further identify potential scalability bottlenecks. This study is carried out on top of geographically dispersed experimental infrastructures, which essentially offer a realistic setup for evaluation of slice provisioning performance. Our feasibility study is carried out using a prototype implementation of a network slicing architecture, which addresses the main needs of the network slice deployment, *i.e.*, resource discovery, slice embedding, resource provisioning, tunnel setup, and inter-PoP slice segment stitching. Additional support for service deployment and resource monitoring provides the necessary means for quantifying the network slicing gains on certain applications or services.

In the following, Section II provides an overview of the NECOS network slicing architecture, on which our feasibility study relies upon. Section III discusses a proof-of-concept implementation for network slice instantiation with detailed descriptions of the individual steps taken for multi-PoP slice deployment. In Section IV, we present our experimental results, and finally in Section V, we highlight our conclusions and give future work directions.

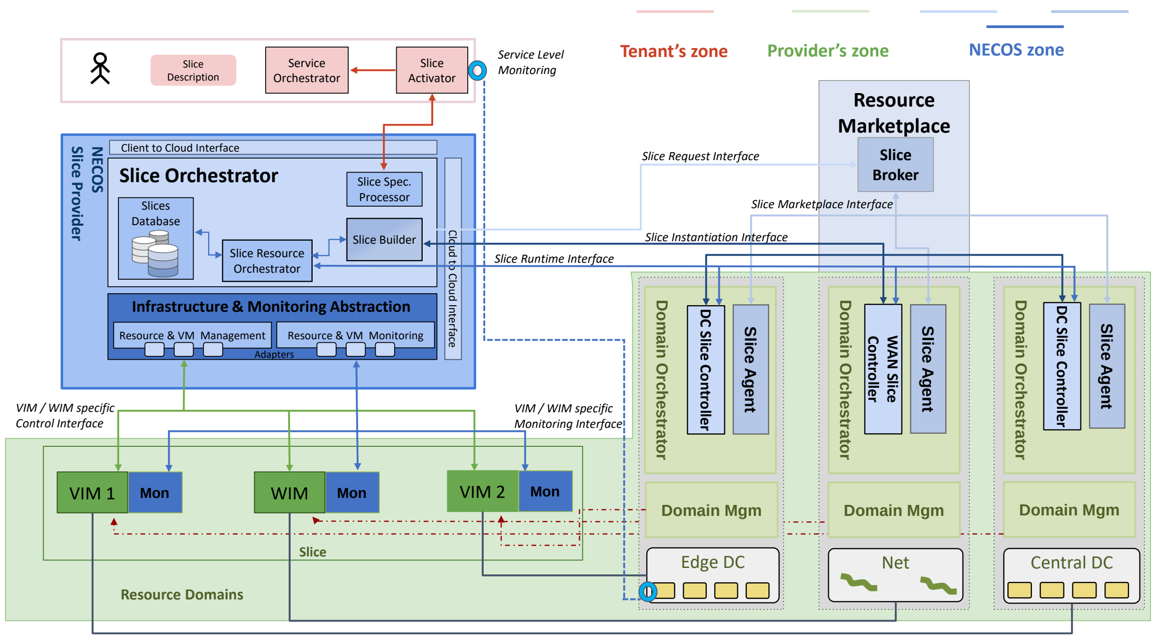


Fig. 1: NECOS slicing architecture.

II. Network Slicing Architecture

The proposed feasibility study relies on a prototype implementation of the network slicing architecture introduced by the NECOS project [2]. The NECOS architecture aims at creating slices by dynamically discovering and allocating resources that span over multiple PoP or administrative domains. The NECOS architecture consists of a number of functional components that inter-operate to deliver the end-to-end slice requested by the tenant [14], as depicted in Fig. 1. The scope of such a distributed design approach is twofold: it enables flexible adaptation of the system to the different needs of the participants to the NECOS ecosystem; it allows a clear separation of concerns during the system implementation phase. To achieve this, it primarily considers three roles:

- the *NECOS Slice Provider*, which is part of the NECOS ecosystem and is responsible to deploy and operate slices, based on the *Tenant*'s request;
- the *Slice Broker*, which is responsible for dynamic resource discovery and acts as the core of the Marketplace;
- the *DC and WAN Providers* that offer either Data Center (DC), or network (WAN) resources, which will become the *slice-segments* of the deployed end-to-end slices.

The architecture allows an administrative authority to fulfil more than one role, *e.g.,* an operator can offer a number of both DC and WAN resources and can act as the NECOS Slice Provider at the same time.

A tenant triggers a slice request in the *Slice Activator*, which could be a Graphical User Interface (GUI) designed to receive the slice specifications. This specification could be derived by the *service graph*, *i.e.,* a description of the service to be hosted by the slice, along with any geographic (*e.g.,* edge cloud location to host a service function) and resource specific constraints. This partial description of the slice is further refined by the *Slice Specification Processor*, which details the slice specification adding the *slice graph*, *i.e.,* slice-segments and their necessary connections, along with a mapping between the latter and the service components.

The *Slice Builder* receives this specification and is responsible for: (i) initiating the resource discovery process in the *Marketplace*, (ii) resource selection based on the available slice-segments received from the former, (iii) resource instantiation via contacting the *DC Controllers* of each slice-segment, and finally, (iv) communicating this information to the *Slice Resource Orchestrator* in order to finalise the end-to-end slice creation and proceed with the required activation.

The *Slice Broker* is the main Marketplace component. Its role consists of receiving a slice request in the form of a "partial" slice description, decomposing it into a number of single slice-segment resource requests addressed to the appropriate *DC* and *WAN Slice Agents*, and compiling a response back to the *Builder*. This response includes alternative proposals for each slice-segment from providers that hold the resources to accommodate them. Each proposal can potentially include information regarding cost or quality characteristics that will be used by the *Builder* to make the most appropriate (in terms of matching requirements) end-to-end slice configuration decision.

Two highly interconnected components reside on the resource providers' domain. The *Slice Agent*, responsible for replying to *Broker* requests by checking current resource availability, and the *DC* or *WAN Controller*. A controller is responsible for creating a *slice-segment*, *i.e.,* for managing the provider's resources that will be offered to the NECOS ecosystem; allocating resources on-demand as requested by the *Builder*; deploying on-demand VIMs and monitoring services for the slice-segment; and for any end-of-life operations, *e.g.,* slice-segment decommission.

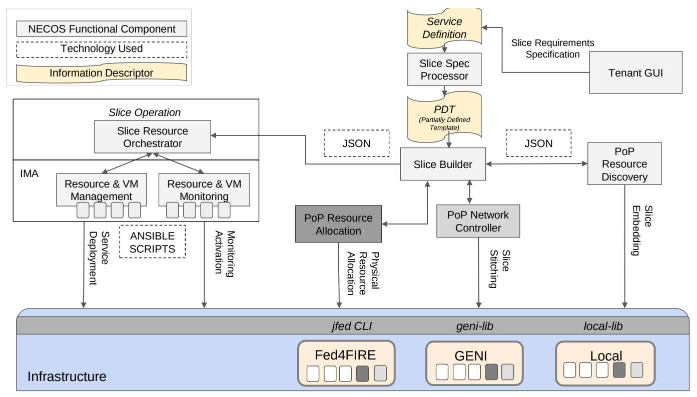


Fig. 2: Slice workflow overview.

The *Slice Resource Orchestrator (SRO)* manages the lifecycle of all the (stitched) slice-segments and orchestrates the service elements across the end-to-end slice, *i.e.,* it performs the placement of service components into the resource domains. These actions are performed according to the monitoring information provided by the *Infrastructure and Monitoring Abstraction* (*IMA*) component. The IMA is a uniform abstraction layer introduced in order to hide the specific technological implementation of the VIM / WAN Infrastructure Manager (WIM) and monitoring subsystems in each slice segment. A uniform northbound interface offered by IMA allows the SRO to perform his specific functions in an abstracted way: a *technology specific, pluggable adaptation layer* to different VIMs/WIMs and monitoring systems is in place at the southbound of the IMA to implement the required abstraction.

Finally, the *Slice Database* is the component specifically introduced for storing the information related to the different end-to-end slices.

III. Prototype Implementation

This section presents our prototype implementation for realising deployment of slices according to the NECOS approach. Compared to the NECOS architecture, we focus on a multi-PoP rather than on a multi-provider environment. This means that our implementation supports a simplified Marketplace operation (Fig. 2): the *Slice Broker* acts as a *Broker Agent* as well, being responsible to report on the resource availability in the different PoPs. Along these lines, we replaced the above two entities with a new component called *PoP Resource Discovery*. For the same reason, the *PoP Resource Allocation* and *PoP Network Controller* components implement the functionalities of the *DC Slice Controllers* and *WAN Slice Controllers*, respectively. Slice embedding over multiple Infrastructure and WAN providers is a complex enough operation to deserve an independent study [15].

We start by discussing the prototype with the functional description of the required steps for the slice deployment and then give an overview of the deployment workflow along with basic implementation details of the associated components.

The slice deployment operation involves the following steps:

- **Slice Requirements Specification:** The *Tenant* defines the slice requirements that include: (i) general slice parameters, including geographic constraints, cost model to use, monitoring options and slice time-frame (*e.g.,* duration); (ii) a service graph consisting of service elements and links with particular demands for physical and/or virtual resources (*e.g.,* VIM type and configuration); (iii) slice stitching requirements, such as bandwidth demands and resource reservation or tunnelling protocol.
- **Slice Embedding:** In this slice deployment step, our facility relies on the Marketplace, which makes initial decisions for the slice and dynamically discovers physical resources that match the expressed demands. Practically, it defines the number of slice-segments and how to distribute the service elements among them, collects the resource offers from different *Infrastructure* and *WAN Providers*, and determines which of them to accept.
- **Physical Resource Allocation:** This step involves the allocation and booting up of physical servers and network devices in the different slice-segments. This is followed by the deployment of the required Operating Systems and VIMs, and completed by booting up all the physical resources. We emulate the edge routers with diverse communication capabilities using physical machines that support tunnelling protocols and bandwidth throttling.
- **Slice Stitching:** The facility stitches the slice-segments together by establishing the WAN connection between them and the required intra-domain network configurations. Furthermore, it employs the VLAN or VXLAN protocols for slice isolation. Finally, all slice servers are connected to each other on a private IP network.
- **Service Deployment:** This step involves the service deployment, which includes the transfer of the VM images to specific servers, the creation of the VMs, and their boot up. The service activation usually terminates with an additional service configuration process.
- **Monitoring Activation:** The last step includes the deployment and configuration of the requested monitoring capabilities from the *Tenant*. This is associated with the activation of probes for the requested Key Performance Indicators (KPIs) via the deployment of a particular monitoring tool, *e.g.,* Lattice [16], Prometheus [17] and Collectd [18]).

We now present an overview of the slice deployment workflow (*i.e.,* as shown in Fig. 2) and the basic implementation details of the prototype components. The *Tenant* realizes the **Slice Requirements Specification** through a GUI, which produces the *Service Definition* schema elaborated in [15]. The *Slice Specifications Processor*, the *Slice Builder* and the *PoP Resource Discovery* components jointly implement the **Slice Embedding** step. The *PoP Resource Allocation* allocates on-demand the physical machines and deploys the requested VIM. The last two components act as wrappers of the novel FED4FIRE [19] and GENI [20] testbed tools (*i.e.,* jfed CLI and geni-lib), in a similar approach to [21]. Furthermore, the same components access a similar testbed control abstraction handling our own UOM testbed. After

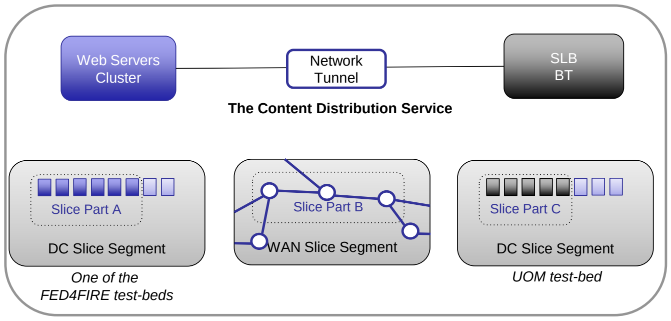


Fig. 3: Content distribution service slice.

the allocation of physical machines deployed at the different PoPs (*i.e.,* slice-segments), the *Slice Builder* communicates with the *Slice Resource Orchestrator (SRO)*, which in turn realizes the **Service Deployment** and **Monitoring Activation** through Ansible scripts. Lastly, the SRO oversees the **Slice Operation**; however, this is not part of the current work. The *PoP Resource Allocation*, *PoP Resource Discovery* and *PoP Network Controller* have been implemented in Python, whereas the other NECOS components as Node-RED nodes[1]. All of them exchange information descriptors specified as JSON messages, in accordance with the NECOS information model [15].

IV. Experimental Evaluation

A. Evaluation Environment

In this section, we experimentally validate the aforementioned slice deployment steps, namely the *Slice Embedding (SEmb)*, *Physical Resource Allocation (PRA)*, *Slice Stitching (SS)*, *Service Deployment (SDepl)* and *Monitoring Activation (MAct)* processes, through our prototype implementation of the NECOS architecture. We consider a distributed content service that geographically spans over Europe and USA. We assume that a *Tenant* requests a slice consisting of the following service functions: (i) a cluster of Web servers; (ii) a service load balancer (SLB); and (iii) benchmarking tools (BT). We further consider that the *Tenant* specifies each service function to be allocated in a particular geographic location, and thus, the service functions span two different DC slice segments, as depicted in Fig. 3. In particular, the left DC slice segment contains resources from the FED4FIRE testbeds federation, whereas the right DC slice segment consists of physical resources located at our own UOM testbed. To emulate the WAN slice segment that stitches the DC slice segments, we deploy an additional physical machine at each side, acting as an edge router.

To proceed with the slice deployment, we define the service and slice requirements as a generic YAML slice-information input that includes the three slice segments. In more detail, the DC slice segment at the UOM testbed consists of six physical nodes hosting: (i) a service load balancer which distributes (*i.e.,* in a round robin fashion) the Web traffic of a number of clients to the Web servers located at the left-side DC slice segment; and (ii) the benchmarking tools emulating the clients' behaviour. The DC slice segment with the Web servers' cluster is physically located in the USA (*i.e.,* the CloudLab Utah testbed) and is accessed through the FED4FIRE facility. The number of physical machines in this segment is expressed by the parameter *Nodes* of our experiment, which is in the range of $[5 \ldots 30]$. We choose two classes of *Nodes'* hardware type, *i.e.,* the *pc3000* class with 3.0 GHz processor, 2 GB DDR2 RAM and 300 GB storage, and the *d430* class with two 2.4 GHz $8-$core processors, 64 GB DDR4 RAM and 2.2 TB storage (a detailed description of hardware specifications can be found at https://wiki.emulab.net/wiki/UtahHardware). The first class could serve as edge cloud and the latter as core cloud. In this particular DC slice segment, we consider the deployment of a Web server per physical node, define the virtualization technology (*i.e.,* ClickOS), and further specify the service resource flavour (*i.e.,* CPU, RAM utilization and storage usage) for each Web server. Furthermore, we designate the traffic policy (*i.e.,* equally, randomly) for the service. The third slice segment (WAN) is responsible for configuring the inter/intra-domain connectivity of the DC slice segments. We boot an extra physical machine in each DC slice side, which acts as an edge router, and, for simplicity, we use GRE tunnels to setup the connectivity between them.

To secure intra-connectivity, we configure each physical node's routing table to allow the communication with the edge router of the other side. For inter-domain connectivity, we assume a star topology where each physical node is connected to the edge router (central node) and through the edge router to the remote DC slice segment (physical remote nodes).

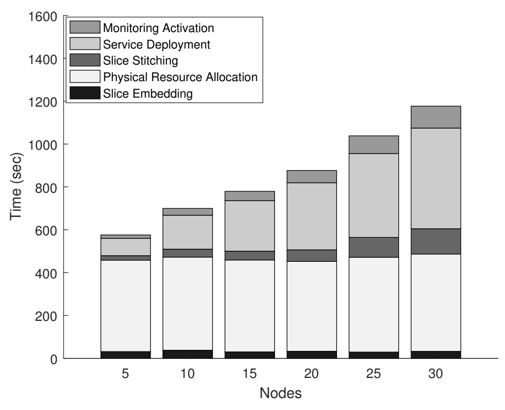
For the *Monitoring Activation* step of slice deployment, we perform the configuration of the CollectD open-source monitoring tool. For the allocated physical resources, we enable the following KPIs: (i) CPU and RAM usage; and (ii) incoming/outgoing traffic. The tool collects the KPI metrics every 20 sec (time interval).

B. Experimental Results

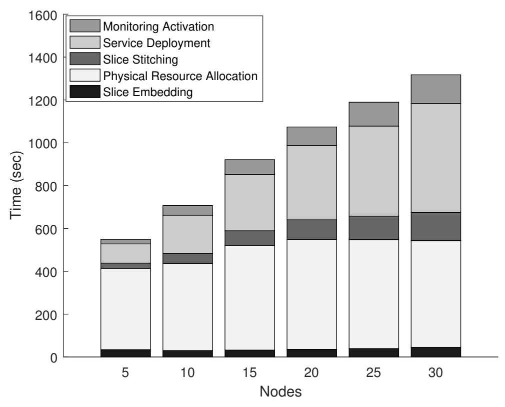
Our evaluation results show the slice deployment time when either core or edge cloud nodes are employed at the DC slice segment accommodating the Web servers' cluster. Fig. 4 provides both a general view of the total delay incurred for slice instantiation, as well as the time spent for each individual step, *i.e., Slice Embedding*, *Physical Resource Allocation*, *Slice Stitching*, *Service Deployment* and *Monitoring Activation*. Delay is expressed as a function of the number of the physical *Nodes* deployed at the CloudLab Utah testbed. We report the results across five runs.

Fig. 4(a) and 4(b) indicate that the less time-consuming steps are *Slice Embedding*, *Slice Stitching* and *Monitoring Activation*. *Slice Embedding*, in particular, is almost fixed at around 30 secs in the case of core cloud nodes, and at around $35-40$ secs in edge clouds. The exact delays along with the standard deviations are also reported in Table I. The *Slice Stitching* step ranges from almost $20-120$ secs and $25-130$ secs, for core and edge cloud nodes, respectively.

[1]https://nodered.org/

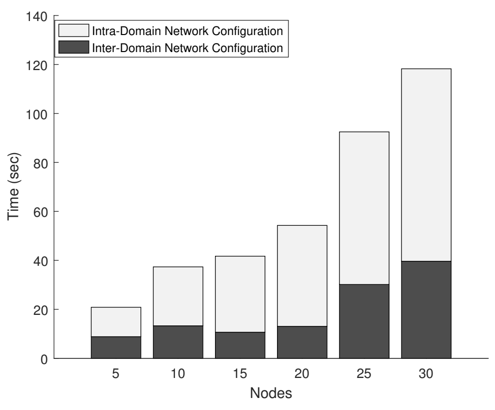


(a) Core cloud nodes

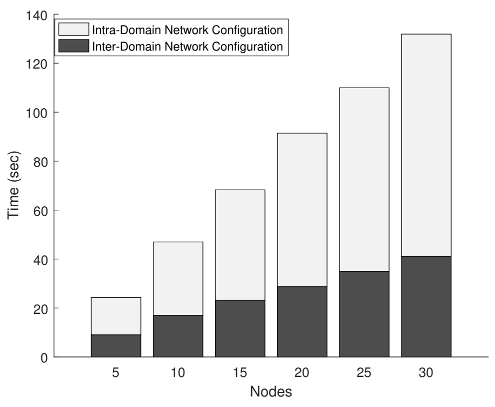


(b) Edge cloud nodes

Fig. 4: Slice deployment time.



(a) Core cloud nodes



(b) Edge cloud nodes

Fig. 5: Slice stitching time.

This step along with the monitoring increase linearly with the number of nodes. The *Monitoring Activation* starts at 16 sec and reaches almost 100 secs (Fig. 4(a)), whilst it ranges from 21 to 133 secs in Fig. 4(b). These results show slightly lower delays when edge cloud nodes are allocated for the DC slice segment. Especially, the *Slice Embedding* step does not yield any scalability limitation, at least in the scale of our experimental setup.

On the other hand, the *Physical Resource Allocation* and the *Service Deployment* steps are the most time-consuming. Resource allocation involves the servers' boot-up time, which entails the prolongation of the total deployment time. However, in case of the core cloud nodes, the resource allocation requires as much as 74% of deployment time when $Nodes = 5$, which significantly decreases at 30% when $Nodes = 30$. The corresponding percentages range from $69 - 37\%$ in case of edge cloud nodes. This decrease is due to the fact that the time remains almost stable with the increase of the nodes. In contrast to this observation, the delay incurred for *Service Deployment* increases with the number of nodes. As a result, service deployment attributes $14 - 40\%$ and $16 - 38\%$ (in respect to the number of nodes) of the total slice deployment time when core and edge cloud nodes are employed, respectively.

In Figs. 5(a) and 5(b), we further elaborate on the time required for *Slice Stitching*. As described in the experimental environment section, the WAN slice segment configures connectivity both inside the DC slice segments, and between the two geographically remote segments, as well. In these figures, the light dark and grey areas correspond to the intra- and inter-domain network configuration, respectively. According to these plots, intra-domain network setup incurs longer delays compared to the inter-domain network configuration, irrespective of the type of cloud nodes. This stems from the fact that in slice stitching, physical node configuration takes place sequentially, whereas in the intra-domain case configurations are applied to a larger number of nodes (compared to inter-domain).

V. Conclusions and Future Work

In this paper, we studied the feasibility of multi-PoP slice deployment, motivated by the increasing interest in network slicing, as means for multi-service and multi-tenancy. We relied on a prototype implementation which provides support for slice and service deployment, based on the slice instantiation workflow exemplified in the NECOS architecture. Our results and micro-benchmarks across a diverse range of network slice sizes corroborate that slice instantiation delay scales linearly with the slice size. The dominant factor is

TABLE I: Standard deviation of deployment time for core and edge cloud nodes

Step	SD (σ) in sec	Core cloud nodes						Edge cloud nodes					
		5	10	15	20	25	30	5	10	15	20	25	30
MAct	Time	16.07	31.83	43.25	57.48	82.44	102.34	21.57	44.91	69.43	86.74	111.19	133.74
	SD	1.51	2.87	0.06	0.11	5.92	1.02	1.9	2.43	2.22	2.37	1.98	5.84
SDepl	Time	80.56	158.19	235.86	312.97	391.61	469.95	89.41	178.49	262.38	346.66	420.19	508.21
	SD	0.24	0.93	1.19	1.65	1.33	0.76	3.18	4.92	10.77	11.12	3.01	7.81
Intra-domain SS	Time	12.01	24.06	31.04	41.22	62.32	78.59	15.24	29.91	45.04	62.72	74.99	90.85
	SD	1.54	3.18	0.09	0.04	6.06	0.97	2.63	0.66	0.35	2.36	1.17	2.44
Inter-domain SS	Time	8.84	13.28	10.67	13.05	30.15	39.64	9.07	17.07	23.23	28.72	34.96	41.03
	SD	2.75	4.54	0.14	0.43	8.34	0.24	2.81	0.59	0.72	0.85	0.22	0.55
PRA	Time	426.91	434.49	428.50	419.09	442.61	453.93	379.73	406.53	488.59	512.8	508.44	497.95
	SD	5.04	10.88	0.52	28.84	4.44	17.18	26.83	35.22	29.65	10.99	1.36	30.13
SEmd	Time	31.31	37.79	30.02	32.83	29.17	32.56	34.36	30.33	32.4	36.19	39.54	45.44
	SD	3.06	10.08	2.28	3.90	1.51	1.96	5.05	4.54	4.66	10.36	8.44	19.66

physical resource allocation, which involves the booting of servers. In contrast to resource allocation (which is out of our control in the remote experimental infrastructures), both slice embedding and stitching incur low delays, and certainly do not introduce any scalability limitation, at least at the scale of our experimental setup.

In principle, there is room for optimizations (especially within the slice provider's domain) in order to reduce the slice instantiation delay. For example, slice instantiation tasks, executed across servers (*e.g.,* virtual machine setup and configuration) and switches, can run in parallel to substantially speed up resource provisioning. We further plan to employ more advanced slice embedding mechanisms (*e.g.,* [11]) and investigate the scalability of multi-domain slice instantiation with near-optimal slice embeddings.

Acknowledgments

This work has received funding from the EU's Horizon 2020 research and innovation programme through the 4th open call scheme of the FED4FIRE+ (grant agr. nº 732638), the EU-BRA Horizon 2020 NECOS Project (grant agr. nº 777067), and the MESON project, co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: T1EDK-02947).

References

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[2] F. S. D. Silva, M. O. O. Lemos, A. Medeiros, A. V. Neto *et al.*, "NECOS project: Towards lightweight slicing of cloud federated infrastructures," in *4th IEEE Conf. on Network Softwarization and Workshops*, June 2018, pp. 406–414.

[3] H. Zhang, N. Liu, X. Chu, K. Long *et al.*, "Network slicing based 5g and future mobile networks: Mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, Aug 2017.

[4] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1," Oct. 2012.

[5] M.-A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris *et al.*, "T-NOVA: An open-source MANO stack for NFV infrastructures," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 586–602, Sept 2017.

[6] B. Sousa, L. Cordeiro, P. Simoes, A. Edmonds *et al.*, "Toward a fully cloudified mobile network infrastructure," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 547–563, Sep 2016.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[8] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[9] S. Clayman, F. Tusa, and A. Galis, "Extending Slices into Data Centers: the VIM on-demand model," in *IEEE 9th International Conf. on Network of the Future - NoF*, Pozna, Poland., 19-21 November 2018.

[10] L. A. Freitas, V. G. Braga, S. L. Correa, L. Mamatas *et al.*, "Slicing and allocation of transformable resources for the deployment of multiple virtualized infrastructure managers (VIMs)," in *4th IEEE Conf. on Network Softwarization and Workshops*, June 2018, pp. 424–432.

[11] D. Dietrich, A. Rizk, and P. Papadimitriou, "Multi-provider virtual network embedding with limited information disclosure," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 2, pp. 188–201, June 2015.

[12] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann *et al.*, "Network virtualization architecture: Proposal and initial prototype," in *Proc. of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, ser. VISA '09, 2009, pp. 63–72.

[13] P. Papadimitriou, I. Houidi, W. Louati, D. Zeghlache *et al.*, "Towards large-scale network virtualization," in *Wired/Wireless Internet Commun.*, 2012, pp. 13–25.

[14] S. Clayman and A. Galis, "D3.2: NECOS System Architecture and Platform Specification.V2," 4 2019. [Online]. Available: http://www.h2020-necos.eu/documents/deliverables/

[15] P. D. Maciel, F. L. Verdi, P. Valsamas, I. Sakellariou *et al.*, "A marketplace-based approach to cloud network slice composition across multiple domains, submitted," in *4th IEEE Conf. on Network Softwarization and Workshops*, June 2019.

[16] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice," in *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*. IEEE, 2010, pp. 239–246.

[17] "Prometheus, Monitoring system and time-series database," https://prometheus.io, 2016, [Online; accessed Apr-2016.]].

[18] "Collectd - The system statistics collection daemon," https://collectd.org/, 2015, [Online; accessed Dec-2015].

[19] T. Wauters, B. Vermeulen, W. Vandenberghe, P. Demeester *et al.*, "Federation of internet experimentation facilities: architecture and implementation," in *European Conf. on Networks and Commun.*, Jun. 2014, pp. 1–5.

[20] M. Berman, J. S. Chase, L. Landweber, A. Nakao *et al.*, "Geni: A federated testbed for innovative network experiments," *Computer Networks, Elsevier*, vol. 61, pp. 5–23, Mar. 2014.

[21] P. Valsamas, I. Sakellariou, S. Petridou, and L. Mamatas, "A multi-domain experimentation environment for 5g media verticals," in *IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds 2019 (CNERT)*, April 2019.