SEA: Towards Slicing Elasticity as a Service

André Luiz B. Rocha[1], Paulo Ditarso Maciel Jr.[2], Fabio Luciano Verdi[1]

[1]Computer Science Department
Postgraduate Program in Computer Science (PPGCC)
Federal University of São Carlos (UFSCar)

[2]Academic Unit of Information and Communication Technologies
Postgraduate Program in Information Technology (PPGTI)
Federal Institute of Paraíba (IFPB)

debeltrami@gmail.com, paulo.maciel@ifpb.edu.br, verdi@ufscar.br

Abstract. *In this paper, we address the challenge of supporting slice resource elasticity within an orchestrated closed control loop (CCL) to ensure Service Level Agreements (SLAs) based on pre-defined Key Performance Indicators (KPIs). We introduce the service model* Slicing Elasticity as a Service *(SlEaaS) and define the Slice Elasticity Architecture (SEA) to fulfill the gap of orchestration in the context of slices. The SEA offers a comprehensive CCL for orchestrating slice resources and enables a range of elasticity operations on different resource types. These operations are facilitated across diverse administrative and technological domains. This architecture allows tenants or slice providers to autonomously monitor slice resources and perform elasticity operations when KPI violations occur. The obtained results demonstrate the functionality of SEA through various scenarios of elasticity operations on network resources. We cover essential aspects such as component instantiation, slice resource monitoring, KPI definitions, and the execution of elasticity operations to ensure compliance with the tenant's SLA.*

1. Introduction

The concept of network slicing has been researched since 2009 [Galis et al. 2009], but it has recently received significant attention, mainly due to its incorporation into future mobile networks. Several definitions of network slicing can be found in the literature and standard bodies, such as Network Slicing, Cloud Network Slicing, and 5G Slicing. For the purpose of this paper, we assume the definition of Cloud Network Slicing (CNS), which refers to an end-to-end (E2E) infrastructure responsible for delivering services across multiple domains, encompassing cloud computing, networking, and storage resources [Clayman et al. 2021, Rocha et al. 2022].

The term *resource elasticity* has been in use for a long time, referring to the process of increasing or decreasing resources on physical or virtual servers. In the context of slices and CNS, the importance of this concept becomes even more evident [de Almeida et al. 2020, Gutierrez-Estevez et al. 2018]. A wide range of verticals benefits from resource elasticity, which is essential in order to adapt and fulfill services' requirements (e.g. load, cost, performance).

However, providing slice elasticity raises several challenges, unaddressed in the literature. The main challenges are fourfold: (i) supporting the heterogeneity of resources that comprise the slice, including cloud, networking, and storage; (ii) supporting different administrative and technological domains; (iii) guaranteeing elasticity operation with minimal service downtime; and (iv) providing slice resources orchestration through management and monitoring operations.

Add and remove resources in a computing environment has been present in our daily lives for decades and has recently been exacerbated by the growth of virtualization. Adding RAM in a computer, creating or deleting virtual machines in a server, and increasing the bandwidth or the number of queues in a router, are examples of elasticity operations. Currently, slice elasticity makes even more sense for applications such as AR/VR [Morín et al. 2022], V2X [Allouis et al. 2021] and many others. AR/VR applications are widely explored for various purposes and can benefit significantly from elasticity operations to ensure that the latency of an AR/VR game does not degrade even when the number of connected users increases. Similarly, for use cases related to V2X services, the ability to scale up or down the infrastructure of a slice is essential to ensure that the defined Key Performance Indicators (KPIs) are met, even with the increase or decrease in the number of users and vehicles connected to the 5G/6G network.

This paper introduces the Slice Elasticity Architecture (SEA) to support the new Slice Elasticity as a Service (SlEaaS) service model. It is noteworthy that our proposal differs from Slicing as a Service, as it focuses on the post-slice instantiation phase rather than the slice instantiation itself. The concept enables new players to independently implement platforms for offering slice elasticity operations using SEA components to interact with virtual resources instantiated on demand for that slice. While a slice provider may offer slice elasticity within their platform, there are many advantages in providing this service separately from slice infrastructure instantiation and negotiation. Therefore, we can highlight the following benefits of SlEaaS:

- **Independence:** decoupling SLEaaS from Slice as a Service empowers tenants to utilize this service independently of the slice provider.
- **Simplified management:** isolating from the slice provider simplifies the management of specific components responsible for delivering SLEaaS.
- **Specialized quality:** specializing the SLEaaS provider solely in the slice CCL, the service can be offered with higher quality and customized by each tenant.
- **Seamless utilization:** tenants and slice providers can seamlessly and automatically utilize SLEaaS to ensure the fulfillment of the required SLAs monitoring the KPIs and performing historical data analysis.

Tenants can delegate slice monitoring, management, and orchestration to a SlEaaS provider that implements the SEA. Whoever delegates the orchestration of their slice to the SlEaaS provider will have a range of elasticity operations available for their slices based on the different resources that compose it. This approach reduces the complexity of resource orchestration by entrusting the task to a specialized, customizable, and scalable service. The SEA can abstract slice resource orchestration, periodically check KPI violations, and execute elasticity operations to ensure tenants' SLAs. The main contributions of this work are as follows:

- Proposal of a new service model called **SlEaaS**, which automates the CCL orchestration of slices;

- Definition of **network elasticity operations** within the context of CNS;
- Implementation of **SEA's key functionalities** to validate slice elasticity operations;
- Demonstration of slice elasticity operations triggered by KPI violations to ensure service throughput and **meet tenant's SLAs**.

2. State of the art

Although literature discusses the elasticity of slices as a significant challenge, there is still a lack of definitions about which resources to increase or decrease and how to do it. Furthermore, there is a dearth of architectures focusing solely and exclusively on providing such elasticity operations efficiently, isolated and simplified for tenants. In this paper, we will present four of the most related works that we found and their main differences with the SEA.

The authors in [Abbas et al. 2021] introduce a solution that automates the configuration process and handles the management and orchestration of network slices. Their solution primarily aims to automate the network configuration of networking resources. In contrast, our paper specifically focuses on defining and implementing slice elasticity operations to enable the concept of Slice Elasticity as a Service (SlEaaS). Consequently, our solution is centered around post-slice instantiation, while it revolves around the management of the network slice lifecycle, which includes commissioning, activation, runtime monitoring, and decommissioning.

In [Kukliński et al. 2021], the authors present an architecture focused on 6G Network Slices, considering the 5G network standardization by 3GPP. Similar to our work, they highlight the critical challenges of slice management and orchestration. Their proposed architecture, also, is modular and encompasses essential characteristics inherent to slices. Additionally, the authors introduce the concept of Management as a Service (MaaS), which shares similarities with SlEaaS. They also discuss other potential elasticity operations within the slicing context. However, SEA goes a step further by implementing and evaluating the architecture on slices, taking into account the heterogeneity of slice resources. SEA also proposes the native support of a novel service model called SlEaaS. Another crucial aspect of SEA is the complete decoupling of the design from any research initiative or project, allowing for SlEaaS availability to both tenants and slice providers.

The ETSI OSM platform [Marsico 2021] serves as a Network Service Orchestrator responsible for controlling the lifecycle of Network Services and Network Slices offered on demand as a northbound service. OSM provides several functionalities and a well-defined user interface for resource management through interfaces that communicate with Virtual Infrastructure and WAN Infrastructure managers. However, the elasticity operations in OSM are limited to scaling Virtual Network Functions (VNFs). In contrast, our proposal defines different elasticity operations based on the type of slice resources, encompassing computing, networking, and storage. Tenants can utilize our well-defined interfaces to implement a complete closed loop to ensure the required SLAs for their slice, customizing KPIs to meet their specific needs. Our proposal can operate on top of infrastructures instantiated or managed by different slice provider proposals (e.g., OSM [Marsico 2021], NECOS [Clayman et al. 2021]).

D. Breitgand et al. [Breitgand et al. 2021] discuss the challenges of cross-domain slice scaling in the context of 5G network slices. The paper is part of the 5GZORRO project, a European initiative focused on providing an architecture and platform for cross-domain interactions between Communication Service Providers (CSPs). However, our work differs from theirs in terms of resources that are scaled to enhance service quality. Our proposal involves intermediate levels of elasticity, taking into account the vertical elasticities of already deployed slice parts. As a last resort, we renegotiate a new slice part (horizontal elasticity) with new infrastructure requirements defined by the tenant and orchestrated by our service. Furthermore, our approach provides elasticity operations based on SLAs defined by tenants, supports multiple slices from different tenants concurrently, and enables easy deployment, termination, or update of all architecture components due to its microservices approach.

3. Slicing Elasticity Operations of Network Elements

We assume that a cloud network slice consists of one or more *slice parts* (SPs), which may be provided by different providers. Furthermore, the SPs must fall into one of two categories: a datacenter slice part (DSP), a.k.a. cloud slice part, or a WAN slice part (WSP). The DSP is composed of cloud resources that are used to create a slice infrastructure, such as physical machines, virtual machines (VMs), or containers. On the other hand, the WSP resources connect DSPs, facilitating the establishment of end-to-end (E2E) slices. The WSP is made up of network elements (NEs) such as switches and routers.

The NEs of a WSP can be either physical or virtual, and may be shared or dedicated. Currently, these network elements can be deployed as white boxes running virtual switches and, when combined with software-defined networking, they offer several possibilities to explore. The virtual elements are provisioned on top of the physical network infrastructure, which is managed by a WAN slice provider. Each slice part operates within its own administrative and technological domains, which can significantly hinder elasticity operations.

In a previous work [Rocha et al. 2022], we defined the operations for computing resources. Due to the lack of a clear definition for elasticity operations of network resources, one of the contributions of this work is to define such elasticity operations and show how to materialize them in network elements. Table 1 shows the taxonomy. Elasticity operations can be *vertical* or *horizontal*. Vertical elasticity refers to the capability of adding (which we name *scale up*) or removing (which we name *scale down*) resources related to NEs in an existing slice part. Typically, it means adding/removing switches, buffer memory, routers, links and queues. Horizontal elasticity refers to adding (which we name *scale out*) or removing (which we name *scale in*) an entire slice part along with all the corresponding resources.

Specifically for this paper, the following network elasticity operations are used:

- **Vertical Scale Up**: add new NEs in an existent WSP;
- **Horizontal Scale Out**: add an entire WSP with its specific infrastructure, e.g., virtual network elements (switches and routers).

4. SEA Design and Implementation

Fig. 1 illustrates the architecture of the SEA, which comprises three layers: the APIs Layer (highlighted in orange), the Control Layer (highlighted in red), and the Per-Slice

Network Elasticity Operations		Examples
Vertical	Scale up	add switches, routers, links, buffer memory and queues
	Scale down	remove switches, routers, links, buffer memory and queues
Horizontal	Scale out	add an entire WSP
	Scale in	remove an entire WSP

Table 1. Slice elasticity operations for networking.

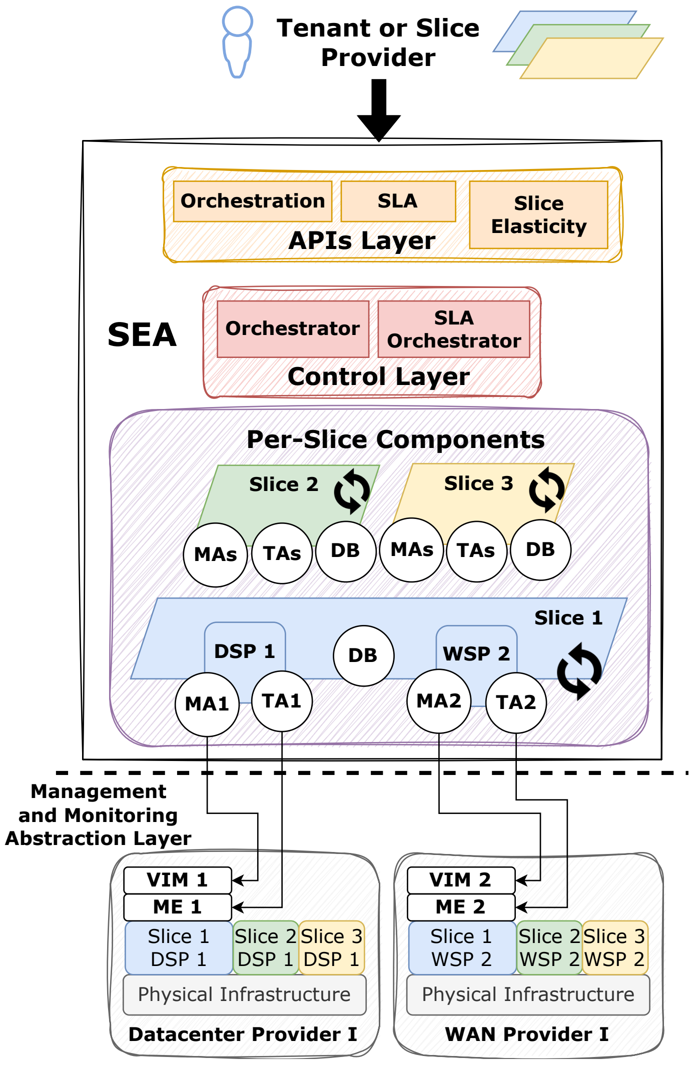


Figure 1. SEA design.

Components Layer (highlighted in purple, with each slice represented by a distinct color). The southbound integration is known as the Management and Monitoring Abstraction Layer, which necessitates some entities for each slice part. Firstly, a Virtual Infrastructure Manager (VIM) or WAN Infrastructure Manager (WIM) is required to oversee the management of infrastructure resources. Secondly, a Monitoring Entity (ME) is employed to monitor all the resources associated with the slice part. Tenants have the flexibility to select their preferred VIM/WIM and ME solutions, which are instantiated by the provider.

Regardless of the selected technology, the SEA can establish communication with the DSP and WSP via well-defined APIs in the form of adapters. This facilitates transparent interaction with diverse technologies while upholding the separation between providers and the SEA.

The architecture consists of shared components that are utilized by all clients, such as APIs, the SEA Orchestrator, and the SLA Orchestrator, as well as dynamic components that are instantiated for each slice. In Fig. 1, the white circles on the Per-Slice Components Layer represent these dynamic components, namely the Management Adapter (MA), Telemetry Adapter (TA), and Database (DB). The MAs are responsible for abstracting the interaction with different VIMs/WIMs to manage the physical or virtual resources provisioned for the corresponding SPs. Similarly, the TAs collect metrics from the MEs and store them in the slice database DB.

An important aspect of the SEA is the utilization of one adapter for each technology employed by the SPs. For instance, DSP 1 of Slice 1 (depicted in blue) may use OpenStack as the VIM and Prometheus as the ME, while DSP 1 of Slice 2 (depicted in green) may employ Kubernetes as the VIM and NetData as the ME. The same heterogeneity applies to the WSPs, where one may utilize an OpenFlow controller while another may rely on an MPLS network. The adapters' abstractions ensure seamless integration across different technologies. All the Per-Slice Components are instantiated as containers, providing manageability, rapid deployment, and other benefits associated with containerized applications.

The functionalities of the SEA can be categorized into four workflows, which are described as follows:

- **Workflow 1 (W1) - Start slice elasticity orchestration**
  The tenant uses the Orchestration API to request the creation of the per-slice components through a well-defined API. The SEA Orchestrator receives the request and creates the MAs, TAs, and DB to orchestrate that specific slice, isolating it from other slices. Each of them is instantiated inside an individual container that contains the code responsible for abstracting the technology used on each slice part, such as VIM/WIM and ME for MAs and TAs. For the DB component, the tenant can choose any database technology available from the provider that implements the SEA (NoSQL, SQL, or time-series database technologies) by informing it as an Orchestration API parameter. Fig. 2 illustrates a set of REST APIs defined on Orchestration API to support W1, they are POST APIs to start, stop and update SEA internal components.
  Each provider that implements the SEA must provide information about the supported technologies (MAs, TAs, and DB). For example, provider A can support only time-series databases, OpenStack as VIM, Ryu OpenFlow controller as WIM, and Ceilometer as ME; while provider B may offer different VIMs, WIMs and MEs technologies. More details about TAs and MEs can be found at [Rocha et al. 2020].
- **Workflow 2 (W2) - Create SLA**
  The CCL orchestration requires clients to use pre-defined methods and parameters of the SLA API to request CRUD (Create, Read, Update, and Delete) operations for SLAs and the respective KPIs. The SLA Orchestrator component establishes a

mechanism for checking the KPIs, which can be triggered either by alerts directly created in the database or through a polling scheme that periodically checks the metrics for each slice in the database and verifies metric violations.
The SLA creation process relies on six essential groups of parameters. These are as follows: (1) slice id and slice part id; (2) KPIs to be monitored; (3) threshold used to trigger an operation; (4) KPIs violation check interval; (5) elasticity operation to be executed after a trigger, as defined in Table 1; (6) parameters specific to the elasticity operation, such as new network elements (NEs), new queues, and slice part configuration. These parameters can be sent to the SLA API in a well-defined descriptive file format, such as JSON, YAML, or XML. Fig. 2 contains the most basic REST APIs defined to CRUD the SLAs.

- **Workflow 3 (W3) - Perform slice elasticity**
  After W1, the tenant has the option to initiate an elasticity operation through the Slice Elasticity API in addition to the CCL. The SEA Orchestrator is responsible for executing the elasticity operation, whether it is triggered manually or automatically in response to a KPI violation detected by the KPI checking mechanism. In either case, the MA is utilized to create or delete new resources in the event of vertical elasticity requirements. If horizontal elasticity is needed, the slice provider is called upon to instantiate a new SP (scaling out) or decommission an existing one (scaling in).
  It is essential to emphasize that the SEA is implemented by a slice elasticity provider, not by the slice provider. Thus, the creation of new SPs must be carried out by slice providers. On the other hand, all vertical elasticity operations are handled by the SEA implementation, as it involves the addition or removal of resources from already instantiated and configured SPs. Fig. 2 presents four Slice Elasticity REST APIs responsible for triggering on the Orchestrator the slice elasticity operations.
- **Workflow 4 (W4) - Adapt SEA per slice components**
  Following a horizontal elasticity operation, the slice provider is required to utilize the Orchestration API to adjust the adapters and reflect the infrastructure changes resulting from the addition or removal of a new SP. This operation affects the per-slice components of the SEA, as there is a direct relationship between MAs/TAs and WIMs/MEs. Consequently, the Orchestration API must be invoked to instantiate both MA/TA components in the case of scaling out or remove existing adapters in the case of scaling in.
  The communication between the Orchestration API and the SEA Orchestrator is responsible for the instantiation of the new MA and TA. The API utilized for this purpose is a component of the Orchestration APIs, referred to as *scaleSEA* (refer to Fig. 2). Horizontal elasticity is the only operation that requires these adaptations since new adapters need to be created and configured to monitor and manage the resources (such as VMs, NEs, and containers) associated with a new SP. Conversely, when vertical elasticity occurs, the per-slice components remain unchanged, as the configuration has already been completed during W1.

5. Evaluation

This section aims to demonstrate the two elasticity operations defined in Section 3. In addition, experimental scenarios were elaborated, ranging from the instantiation of com-

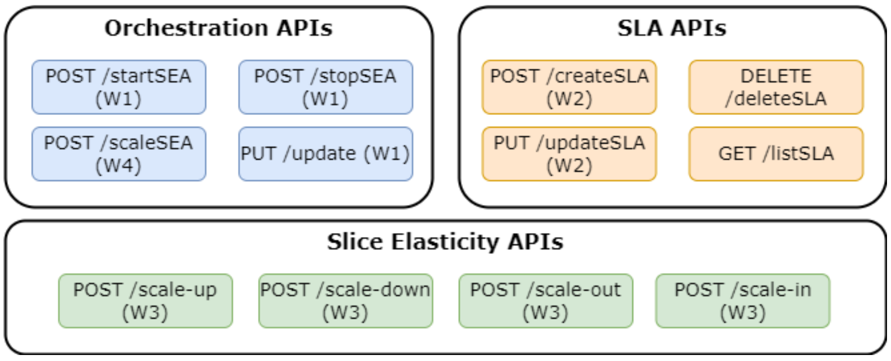


Figure 2. SEA APIs summary.

ponents necessary to orchestrate the slice to the creation of KPIs and, finally, the execution of the elasticity operation. Fig. 3 illustrates the evaluation scenarios to cover two different vertical elasticities scale up (adding new queues and NEs, respectively) and one horizontal elasticity (adding a new WSP 2). At first, Fig. 3 (a) shows the Slice infrastructure in its initial state (before the KPIs violations and slice elasticity operation), which consists of two DSPs connected via one WSP. Two applications run on VMs in each DSP, representing an E2E communication service. App 1 is running through VM1 (DSP 1) and VM3 (DSP 2) in blue, while App 2 is running through VM2 (DSP 1) and VM4 (DSP 2) in green. The WSP 1 topology provided only for this slice contains three switches (S1, S2, and S3), managed by an OpenFlow Controller as WIM. The WSP 1 was configured with bandwidth limited to 20 Mbps.

In all scenarios, the objective is to improve the throughput of App 2 (highlighted with a yellow star), through the execution of elasticity operations after a KPI is violated. App 1 is only used as background traffic to assist in the evaluation. Fig. 3 (b) represents Scenario I, where the *Vertical Elasticity Scale Up* is executed by adding priority queues to existing switches. Scenario II is shown in Fig. 3 (c), corresponding to the *Vertical Elasticity Scale Up* by adding one new virtual switch (S4) to provide a new disjoint path between VM 2 and VM 4. Lastly, Fig. 3 (d) illustrates the *Horizontal Elasticity Scale Out* on Scenario III by adding a new WSP (WSP 2) but with a bandwidth of 40 Mbps (two times greater than WSP 1). All elements highlighted in purple inside Figure 2 indicate the resources added due to the elasticity operations.

For the experimental setup, we used Mininet[1] for emulating open vSwitches with support to the protocol OpenFlow v1.4. Moreover, the OpenFlow Ryu Controller was chosen for the control plane of WSPs. We assumed a total bandwidth limited to 20 Mbps (for WSP1) and used the Iperf3 tool[2] to generate traffic for both applications. Regarding the traffic composition, App 1 comprises 9 TCP flows and App 2 of 1 TCP flow, totaling 10 flows competing for the bandwidth capacity of 20 Mbps. No limitations were imposed for any TCP flow. The elasticity operations aim to improve the throughput of App 2 compared with the throughput before the elasticity operation is triggered.

The infrastructure used as a testbed is composed of two VMs to represent each WSP instantiated over the same physical computer with 32 GB of RAM DDR4, an In-

[1]http://mininet.org/.
[2]https://iperf.fr/

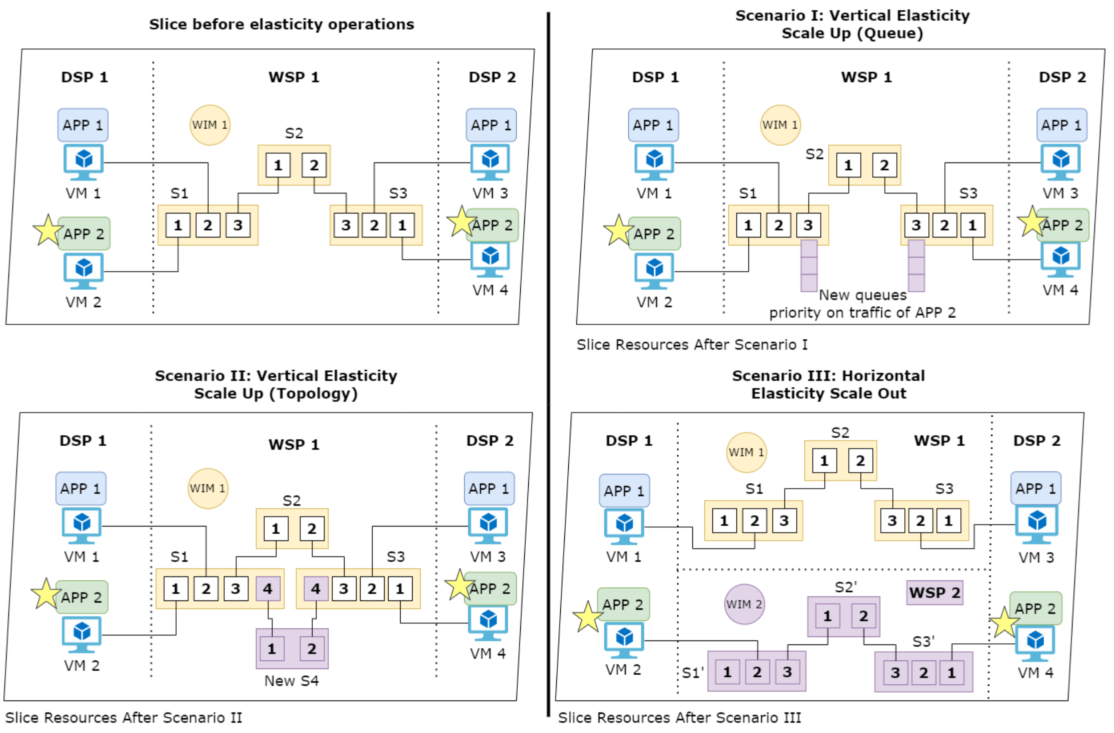

Slice before elasticity operations
DSP 1
WSP 1
DSP 2
APP 1
APP 2
VM 1
VM 2
VM 3
VM 4
WIM 1
S1
S2
S3
Scenario I: Vertical Elasticity
Scale Up (Queue)
New queues
priority on traffic of APP 2
Slice Resources After Scenario I
Scenario II: Vertical Elasticity
Scale Up (Topology)
New S4
Slice Resources After Scenario II
Scenario III: Horizontal
Elasticity Scale Out
WSP 2
WIM 2
S1'
S2'
S3'
Slice Resources After Scenario III

Figure 3. SEA scenarios evaluation.

tel(R) Core(TM) i5-9600K CPU, running Windows 11 as the SO. The VMs were virtualized using the Hyper-V Manager and were instantiated two VMs with 8 GB of RAM, 2 vCPUs, and running the Ubuntu 20.04 LTS. Each of those VMs represents one WSP with the Mininet installed inside. The SEA was running inside a third VM with the same hardware configuration but with the SEA installed and the APIs running correctly. It is important to highlight that the SEA instantiated the per-slice components in the same third VM. The VMs containing the applications were Mininet hosts with Iperf3 installed.

During each experiment, the four-phase workflow (Section 4) is executed to accomplish the entire CCL. As we focus on the network slice part, only the actions on related elements are described next. Therefore, in W1, the dynamic per-slice components were created. In addition, the MA adapter interfaces the Ryu Controller inside the WSP, and the TA adapter also collects monitoring metrics from the Ryu Controller using the OpenFlow protocol. So, in this case, the Ryu Controller also acts as the ME. In W2, the configured KPI defines that when a given switch port reaches 18 Mbps of received rate, an elasticity operation must be triggered[3]. The KPI checking interval is 5 seconds. In W3, the elasticity operation is executed in each scenario. Finally, when needed (Scenario III), W4 is executed to instantiate the new MA and TA elements. For the sake of reproducibility, all the code developed and examples of APIs can be found at [4].

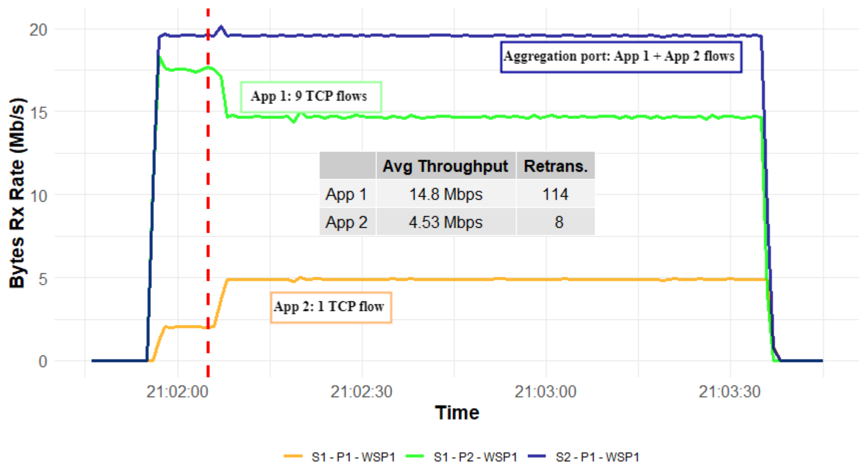
Table 2 summarizes all the scenarios presented, highlighting what are the slice resources added to each scenario and the expected throughput of App 2. Below we detailed all scenario results.

[3]Applications' metrics can also be used, such as frames per second, requests per second, etc.
[4]https://github.com/dcomp-leris/SEA/tree/main/yamls.

Scenarios	New slice resources	Expected Throughput App 2
Scenario I Vertical Scale Up (Queue)	⇒ New priority queues	≈ 17 Mbps
Scenario II Vertical Scale Up (Topology)	⇒ New switch (S4) ⇒ New ports on switches (S1 and S3)	≈ 17 Mbps
Scenario III Horizontal Scale Out	⇒ New WIM with bandwidth of 40 Mbps (WSP 2) ⇒ New virtual resources (S1', S2' and S3')	≈ 40 Mbps

Table 2. Summary of the scenarios evaluated.

5.1. Scenario I: Vertical elasticity queue scale up



	Avg Throughput	Retrans.
App 1	14.8 Mbps	114
App 2	4.53 Mbps	8

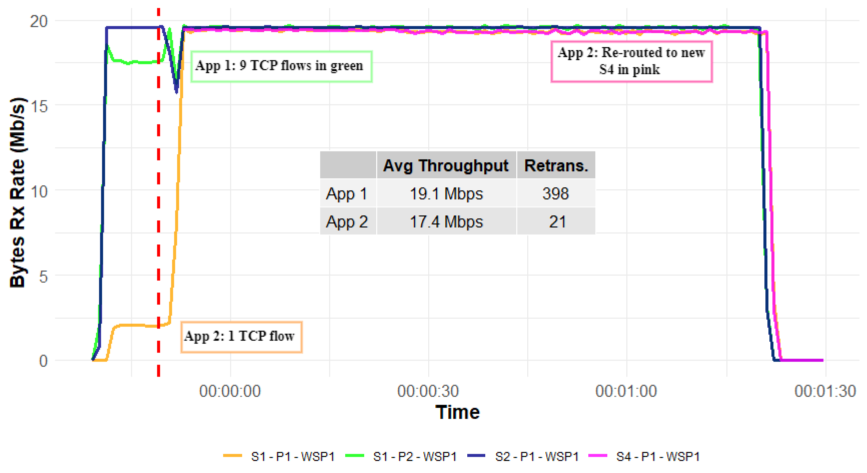
Figure 4. Scenario I: Vertical Scale Up (Queue).

- **Slice elasticity action:** After detecting the KPI violation, this elasticity operation triggers the creation of two queues with a ceiling throughput of 5 Mbps on port 3 of both S1 and S3 (purple queues on Fig. 3(b)). As defined by W3 in Section 4, the tenant provides the extra information necessary for the execution of this operation. One example of the data used can be seen at [5].
- **Results:** Fig. 4 shows the traffic (bytes received rate) being monitored on specific ports of the topology and the exact moment when the elasticity operation is triggered, marked as a dashed red line. The aggregated traffic (dark-blue line on top) comes from the flows of App 1 (green line) and App 2 (orange line) to port 2 of S1. As a result of the KPI violation on that port, the created queues enforce the pretended improvement in App 2 throughput to an average of 4.53 Mbps (instead of the ≈2.5 Mbps before the elasticity), limiting App 1 flows to the remaining bandwidth. It is noteworthy that the applications' flows share the available bandwidth (20 Mbps) inside the slice, with App 1 flows being even more "greedy" before the elasticity operation.

[5]https://github.com/dcomp-leris/SEA/blob/main/yamls/create-pol.yaml.

- **Discussion:** The table inside Fig. 4 highlights the intended behavior by showing the applications' average throughput and the number of retransmissions during the test. After the elasticity operation, the App 2 transfer rate increases from $\approx$2.5 Mbps to $\approx$5 Mbps, consequently decreasing App 1 throughput. In addition, we can notice a continuous transmission during the experiment. Some packets were retransmitted (*114* on App 1 and *8* on App 2) due to the new OpenFlow rules that prioritized the queue to App 2 traffic. However, even with these retransmissions, there was no service interruption, packet loss, or negative impact on the application performance. Finally, we conclude that the tenant was able to improve the App 2 throughput without setting any strict policy on App 1, which is expected to have its throughput decreased in this scenario.

5.2. Scenario II: Vertical elasticity topology scale up



	Avg Throughput	Retrans.
App 1	19.1 Mbps	398
App 2	17.4 Mbps	21

Figure 5. Scenario II: Vertical Scale Up (Topology).

- **Slice Elasticity Action:** After detecting a KPI violation, this operation creates the new NEs and re-routes the App 2 flow. The extra information needed for this scenario consists of the new topology configuration, with a newly created switch S4 connecting S1 and S3 (purple elements on Fig. 3(c)). S4's ceiling transmission is restricted to 20 Mbps.
- **Results:** Fig. 5 shows the bytes received rate on specific ports of the topology and a dashed red line indicates the moment the elasticity operation is triggered. As soon as this operation is completed, the newly created port 1 in S4 receives the re-routed traffic of App 2 (in pink), bounded at $\approx$20 Mbps as expected.
- **Discussion:** The table inside Fig. 5 indicates how the App 2 throughput improved from $\approx$2.2 Mbps to $\approx$20 Mbps after the elasticity operation is completed. The following steps were performed: the creation of topology elements (switch, ports, and links), re-routing of App 2 traffic (without interruption), and the monitoring of new NEs immediately after the SEA instantiates them. The elasticity operation creates the OpenFlow rules to ensure that App 1 traffic goes through the S2 and App 2 traffic through the new S4. Similar to the previous scenario, the re-routing of flows after carrying out the elasticity operation resulted in significant retransmissions.

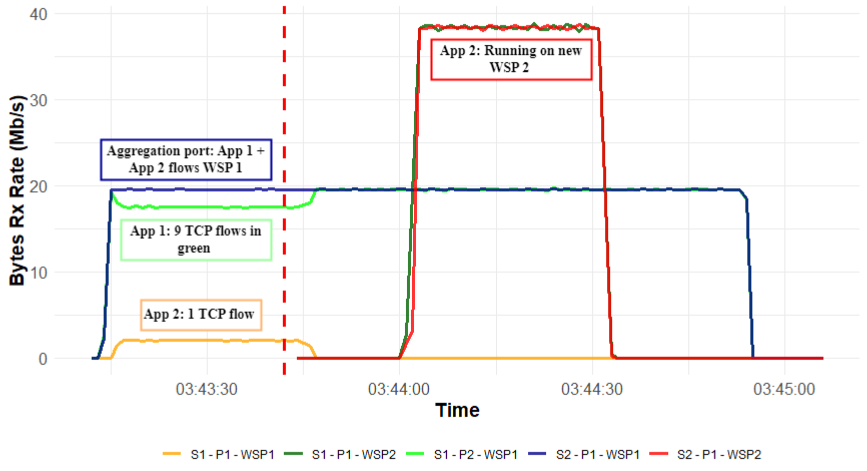


Figure 6. Scenario III: Horizontal Scale Out.

5.3. Scenario III: WAN Horizontal elasticity scale out

- **Foreword:** horizontal elasticity is the capability to add or remove an entire slice part. Regardless of how it is triggered, if manually by the tenant or automatically by the SEA, the SLA Orchestrator delegates the elasticity to the Slice Provider so that it executes the operation as stated in W3. In case of a slice part addition, the Slice Provider must invoke the SEA API to instantiate the adapters according to pre-defined parameters of the new WSP, as part of W4. Then, SEA reclaims control and assumes the next steps of the CCL.
- **Slice Elasticity Action:** the Slice Provider must invoke SEA as defined in W3, with the following extra information needed to instantiate the adapters: IP address, port, and credentials of the WIM and ME[6]. The tenant pre-defined attributes to the new WSP include the WIM controller (Ryu Controller), the ME (OpenFlow protocol v1.4), the metric to be monitored (rate of received bytes), the polling interval (1s), the new topology requirement in terms of which network elements (clone of WSP 1), and the bandwidth limit of network elements (40 Mbps).
- **Results:** Fig. 6 also illustrates the bytes received rates of specific ports in this scenario, related to WSP 1 and WSP 2. We configured the App 2 flow as a 30-second stream before the elasticity operation and after that, we created a new App 2 flow configured to use WSP 2. The figure shows that, right after the first flow of App 2 finishes (in orange), App 1 flows reach the max bandwidth capacity of port 1 in S2 (20 Mbps). It is also noted that after the elasticity operation trigger indicated by the dashed red line, the WSP 2 is instantiated and the App 2 flow is started throughout it, eventually reaching its max bandwidth capacity of $\approx$40 Mbps, shown in the red line. We experimented this way to emphasize that it is not SEA's responsibility to re-route any application flow after a horizontal elasticity happens. After W3 and W4, the tenant can interact with the MA to manage the new WSP infrastructure and must reconfigure the applications to use it or not. For the sake of simplicity, we emulate this behavior by executing App 2 twice in different instants, before and after the elasticity.

[6]More details about this operation can be found in [Rocha et al. 2022].

- **Discussion:** Firstly, the App 1 and App 2 flows compete for the available bandwidth at WSP 1. After the elasticity operation was completed, App 1 continued using only the WSP 1 infrastructure. On the other hand, App 2 was manually configured to use only the newly instantiated WSP 2 (shown in the red line). As a result, both applications could reach the maximum throughput of corresponding WSPs (20 Mbps for WSP 1 and 40 Mbps for WSP 2). The WSPs elements are exclusively dedicated and instantiated to each slice, meaning isolated traffic inside each slice. Therefore, the WSP 2 does not have any initial traffic on it. In addition, no retransmissions occurred in this scenario since the applications were reconfigured at the service level.

6. Conclusion

In this paper, we introduce a new architecture, SEA, which enables slicing elasticity as a service. Our architecture supports the SlEaaS service model, which aims to abstract the heterogeneity of slice resources and infrastructure providers. We specifically focus on the elasticity of network elements, which has not been well-explored in the literature. Our evaluation demonstrates the main concepts behind the provision of this closed control loop, including Vertical Scale Up and Horizontal Scale Out. Based on our results, we conclude that the SEA significantly improves the service performance of a specific application in response to elasticity operations in two different scenarios. We were also able to validate the main functionalities and components of the architecture.

The limitations of the SEA are currently related to the scalability of the slice elasticity provider in transparently supporting a large number of MAs and TAs. Since these components are instantiated as containers within the infrastructure of the slice elasticity provider, each of them consumes resources, which may require scaling the number of servers responsible for providing elasticity as a service to monitor and orchestrate a large number of slices. In future work, we intend to evaluate this resource consumption and measure the number of MAs and TAs that can be supported on servers with different configurations.

The future work of SEA hinges on the integration of machine learning algorithms to enhance the lives of tenants by enabling them to request slice elasticity services. By leveraging machine learning algorithms, the slice provider can identify the CRUD parameters of SLAs based on phrases provided by the tenant, eliminating the need for explicitly defining each parameter through the SLA API. Additionally, historical data on slice elasticity operations and KPIs can be utilized to forecast when to initiate a slice elasticity operation, preempting the violation of thresholds. Moreover, it can optimize the utilization of slice resources even in the absence of any explicitly defined KPIs.

Acknowledgment

This research was partially supported by the H2020 4th EU-BR Collaborative Call, under the grant agreement no. 777067 (NECOS – *Novel Enablers for Cloud Slicing*). This work has been supported by the following Brazilian research agencies: CAPES and CNPq under grant agreement no. 001.

References

[Abbas et al. 2021] Abbas, K., Khan, T. A., Afaq, M., and Song, W.-C. (2021). Network slice lifecycle management for 5g mobile networks: An intent-based networking approach. *IEEE Access*, 9:80128–80146.

[Allouis et al. 2021] Allouis, A., Dayoub, I., and Cherkaoui, S. (2021). On 5g-v2x use cases and enabling technologies: A comprehensive survey. *IEEE Access*, 9:107710 – 107737.

[Breitgand et al. 2021] Breitgand, D., Lekidis, A., Behravesh, R., Weit, A., Giardina, P., Theodorou, V., Costa, C. E., and Barabash, K. (2021). Dynamic slice scaling mechanisms for 5g multi-domain environments. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 56–62.

[Clayman et al. 2021] Clayman, S., Neto, A., Verdi, F., Correa, S., Sampaio, S., Sakelariou, I., Mamatas, L., Pasquini, R., Cardoso, K., Tusa, F., Rothenberg, C., and Serrat, J. (2021). The necos approach to end-to-end cloud-network slicing as a service. *IEEE Communications Magazine*, 59(3):91–97.

[de Almeida et al. 2020] de Almeida, L. C., Jr., P. D. M., and Verdi, F. L. (2020). Cloud network slicing: A systematic mapping study from scientific publications. *CoRR*, abs/2004.13675.

[Galis et al. 2009] Galis, A., Abramowicz, H., Brunner, M., Raz, D., Chemouil, P., Butler, J., Polychronopoulos, C., Clayman, S., Meer, H., Coupaye, T., Pras, A., Sabnani, K., Massonet, P., and Naqvi, S. (2009). Management and service-aware networking architectures (mana) for future internet — position paper: System functions, capabilities and requirements. pages 1 – 13.

[Gutierrez-Estevez et al. 2018] Gutierrez-Estevez, D. M., Gramaglia, M., de Domenico, A., di Pietro, N., Khatibi, S., Shah, K., Tsolkas, D., Arnold, P., and Serrano, P. (2018). The path towards resource elasticity for 5g network architecture. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 214–219.

[Kukliński et al. 2021] Kukliński, S., Tomaszewski, L., Kołakowski, R., and Chemouil, P. (2021). 6g-lego: A framework for 6g network slices. *Journal of Communications and Networks*, 23(6):442–453.

[Marsico 2021] Marsico, A. (2021). Osm in action. Technical report, OSM ETSI, https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_in_Action.pdf.

[Morín et al. 2022] Morín, D. G., Pérez, P., and Armada, A. G. (2022). Toward the distributed implementation of immersive augmented reality architectures on 5g networks. *IEEE Communications Magazine*, 60(2):46–52.

[Rocha et al. 2020] Rocha, A., Maciel, P. D., Tusa, F., Cesila, C., Rothenberg, C., Pasquini, R., and Verdi, F. L. (2020). Design and implementation of an elastic monitoring architecture for cloud network slices. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7.

[Rocha et al. 2022] Rocha, A. L. B., Cesila, C. H., Maciel, P. D., Correa, S. L., Rubio-Loyola, J., Rothenberg, C. E., and Verdi, F. L. (2022). Cns-aom: Design, implementation and integration of an architecture for orchestration and management of cloud-network slices. *J. Netw. Syst. Manage.*, 30(2).