Future Generation Computer Systems 141 (2023) 473–488

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

# End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum

Francesco Tusa [a,b,*], Stuart Clayman [b]

[a] School of Computer Science and Engineering, University of Westminster, London, UK
[b] Department of Electronic and Electrical Engineering, University College London, London, UK

ARTICLE INFO

*Article history:*
Received 16 August 2022
Received in revised form 27 October 2022
Accepted 26 November 2022
Available online 5 December 2022

*Keywords:*
Internet of things
End-to-end slice
Bare-metal cloud
Slice abstraction
Orchestration
Programmability
Multi-tenant

ABSTRACT

Fog computing, combined with traditional cloud computing, offers an inherently distributed infrastructure – referred to as the *cloud-to-edge continuum* – that can be used for the execution of low-latency and location-aware IoT services. The management of such an infrastructure is complex: resources in multiple domains need to be accessed by several tenants, while an adequate level of isolation and performance has to be guaranteed. This paper proposes the dynamic allocation of *end-to-end slice*s to perform the orchestration of resources and services in such a scenario. These *end-to-end slice*s require a unified resource management approach that encompasses both data centre and network resources. Currently, fog orchestration is mainly focused on the management of compute resources, likewise, the *slicing* domain is specifically centred solely on the creation of isolated network partitions. A unified resource orchestration strategy, able to integrate the selection, configuration and management of compute and network resources, as part of a single abstracted object, is missing. This work aims to minimise the silo-effect, and proposes *end-to-end slice*s as the foundation for the comprehensive orchestration of compute resources, network resources, and services in the *cloud-to-edge continuum*, as well acting as the basis for a system implementation. The concept of the *end-to-end slice* is formally described via a graph-based model that allows for dynamic resource discovery, selection and mapping via different algorithms and optimisation goals; and a working system is presented as the way to build slices across multiple domains dynamically, based on that model. These are independently accessible objects that abstract resources of various providers – traded via a *Marketplace* – with *compute slices*, allocated using the *bare-metal cloud* approach, being interconnected to each other via the connectivity of *network slices*. Experiments, carried out on a real testbed, demonstrate *three features* of the *end-to-end slices*: resources can be selected, allocated and controlled in a softwarised fashion; tenants can instantiate distributed IoT services on those resources transparently; the performance of a service is absolutely not affected by the status of other slices that share the same resource infrastructure.

## 1. Introduction

The continuous growth and evolution of pervasive services, ubiquitously accessible by their users over the Internet, has pushed the mobile network's capacity and capabilities to the limits. The Internet of Things (IoT) is playing a major role in this, as the number of IoT devices, and the amount of data they produce, have been increasing exponentially. It is expected there will be 36 billion IoT devices generating 79.4 ZB of data by 2025 [1]. Harnessing the above IoT data can help organisations develop new business models and streamline operational processes, thus creating more innovative products and services across various industries [2]. As this strategy is being pursued, mobility support, geo-distribution, as well as location awareness and low-latency have all become important requirements that the resource infrastructure should support.

IoT devices are usually resource-constrained and not able to perform data computation locally, hence, IoT data are commonly offloaded to an external processing layer. Traditional *cloud computing* would not be efficient in this context, as the large amount of data that need to be transferred, from the data producers to the centralised data centres, to perform the required computation, would incur in large round-trip delays that could ultimately affect the users' experience [3]. Since IoT devices are deployed at the *edge* of the network, the usage of resources located in proximity of those data sources is considered a more viable solution [4]. The data processing can happen primarily at the *edge*, while the *cloud*

* Corresponding author at: School of Computer Science and Engineering, University of Westminster, London, UK.
*E-mail address:* f.tusa@westminster.ac.uk (F. Tusa).

https://doi.org/10.1016/j.future.2022.11.026

would only be accessed when the *edge* resources do not suffice. *Fog computing* builds on this idea and extends it by combining and using the different layers of compute, storage and network resources distributed across the *edge* and the *cloud*, which are commonly referred to as the *cloud-to-edge continuum* [5].

Compared to the traditional sole management of either edge or cloud resources, this combined scenario requires the *seamless* interworking of various *heterogeneous* resource elements, belonging to the different layers of a distributed infrastructure. *Fog nodes*, such as resource-constrained devices, cloudlets and micro data centres, become all part of a resource continuum that extends up to the central cloud, whose elements are interconnected through various network resources. When a new IoT service instance is activated, relevant resources need to be selected from the cloud-to-edge continuum dynamically, in order to support the requested service functionality and to ensure the expected service performance.

The intrinsic heterogeneity of the cloud-to-edge continuum is exacerbated by its potential extension through multiple administrative domains [6]. Services can be built by *tenants* on a dispersed set of resources offered by multiple infrastructure providers. This raises additional challenges, as federation agreements need to be established between providers, and inter-domain interactions may be required after a service has been activated [7]. Moreover, various tenants can request the deployment of diverse types of services, with potentially orthogonal Quality of Service (QoS) requirements, on this shared federated infrastructure. Some IoT use cases, such as Industrial Control, Real-time Analytics, and Artificial Intelligence (AI), certainly have more demanding performance needs compared to other use cases. This may introduce additional complexity, as they would benefit from service execution within *fully-isolated* and *highly-secure* compute and networking environments, in order to achieve the best performance and minimise the potential interference with other running services [8].

From all of the above considerations, it can be observed that automating the deployment of IoT services in the cloud-to-edge continuum is a non-trivial task. It requires the *orchestration* of heterogeneous resources, from potentially different administrative domains, to support the instantiation of the required service components. Then, the service components need to be activated on those resources, and to be configured as part of a single workflow. Finally, multiple service instances, belonging to different tenants, may co-exist on the same resource infrastructure and their expected performance has to be guaranteed, regardless of the status of the other running services. These are all different aspects of a broad problem, currently being investigated by many research studies [6], and it is also the **main research question** considered in this paper.

The 5G was conceived to support *multi-tenant* infrastructures, i.e., to enable the execution of services with demanding and diverse requirements – such as ultra-low latency, massive data rate, and high-reliability – on a shared pool of resources [9]. *Network slicing* is a 5G key technology that builds on both *Network Function Virtualisation (NFV)* and *Software Defined Networks (SDNs)* to provide isolated service execution, and to enable the co-existence of use cases with diverse demanding requirements on a shared physical infrastructure [10]. Many network slicing solutions are specifically focused on the network management and do not define *cloud slicing* mechanisms [7]. When a new *network slice* is created for a tenant, the network infrastructure is dynamically partitioned and assigned exclusively to that tenant, however, the management of the compute resources does not follow a similar pattern. The Virtual Machines (VMs) belonging to all the different services and tenants often share the same physical compute elements. This poses security concerns and can lead to the problem of services impacting each other's performance, regardless of being mapped and executed on separate slices [11].

Whilst those may not be issues for all the tenants, many of the IoT services mentioned earlier may perform data-intensive processing tasks [3]. These may not run efficiently within virtualised computing environments, and their execution may also affect the resource utilisation of other services running on the same physical host. *Bare-metal clouds* can mitigate those issues by providing single-tenant computing systems, provisioned on-demand and billed via a pay-per-use approach, which may become a viable way to create *compute slices*. Customers can benefit from stronger isolation than multi-tenant clouds, and could also have the freedom to deploy their preferred (virtual) resource management solution, allowing a trade-off between resource manageability and performance. For instance, the overhead of the virtualisation layer can be minimised for critical workloads, by either using more lightweight containerisation techniques or accessing the bare-metal resources directly [12].

A careful analysis of the state-of-the-art on orchestration in the cloud-to-edge continuum, network slicing, and bare-metal clouds, highlights that relevant advance was made in those areas by separate research communities. New solutions have been proposed to automate the allocation of resources across the cloud and the edge [6], and to allocate isolated groups of either network [13] or compute (and storage) resources [14]. However, to the best of our knowledge, a comprehensive unified resource management approach that combines the benefits of all these technologies is currently missing. In an attempt to bridge the above gap, this paper proposes the usage of *end-to-end slices* as the basis of a unified, multi-tenant strategy for the orchestration of resources across multiple providers, and the automated activation of services, in the cloud-to-edge continuum.

In the context of network slicing, the term *end-to-end* refers to the orchestration of the different segments of the network infrastructure (e.g., access network, transport network, core network, etc.) [15]. In this paper, it is used in a broader sense to describe a unified approach for the orchestration of heterogeneous compute, storage and network resources, specifically selected from the cloud-to-edge continuum and allocated for a tenant *on-demand*, according to their specific service requirements. This ensures the adequate level of isolation and security requested by that tenant, and can prevent the likelihood of service performance degradation.

This *end-to-end slice* concept allows a separation of concerns between the resource orchestration and the service orchestration. The former usually refers to the process of managing and coordinating the physical computational resources provided by the underlying infrastructure to serve the applications/services [16]; service orchestration indicates the management of the life-cycle of one or more distributed components that together deliver a service or functionality [17]. In this work, each slice is built as a fully-isolated set of resources, and provides an abstraction on top of its composing distributed resource elements. A tenant can therefore activate the required service components on each slice transparently, with the guarantee that the service performance will be independent of the status of the services running on other slices. This can be achieved by each tenant even without the need of adapting the components of their existing service orchestration layer.

### 1.1. Contribution overview

The **main contribution** of this work is the formal definition, the design and the implementation of the notion of *end-to-end slice*, whose suitability to answer the *main research question* stated earlier is assessed in this paper through a qualitative evaluation. The slice concept presented here unifies the management of compute, storage and network resources by creating on-demand, and under software control, a per-tenant resource partition across the cloud-to-edge continuum, whose elements can be orchestrated as a single abstracted entity, for the instantiation of distributed services. This section will briefly explain how this resource orchestration approach compares to related existing solutions in the state-of-the-art. The discussion will be driven by the presentation of the **three features** that underpin the devised idea of *end-to-end slice*.

Existing work on orchestration in the cloud-to-edge continuum is limited to the definition of reference architectures, with a limited number of contributions discussing system implementations, validated via either simulations or the usage of real testbeds [6]. Many of these systems are built via extending existing cloud orchestration solutions towards the edge, rather than implementing a new unified orchestration strategy. Also, they do not consider the federation of resources between multiple providers for the delivery of IoT services, as it usually happens for cloud infrastructures. In this paper, a real system based on a comprehensive orchestration technique, able to work seamlessly across the different layers of the cloud-to-edge continuum, is proposed. Resources for the slices are traded between providers using a marketplace inspired approach. Moreover, a unified model for resources and slices, based on undirected graphs, allows dynamic resource discovery, selection and mapping via different algorithms, according to the desired optimisation goals. The on-demand, software-enabled and programmable allocation of groups of compute, storage, and network resources, selected from the cloud-to-edge continuum, is the **first feature** of our slicing approach.

Multi-tenancy capabilities are provided in multi-domain scenarios by many of the existing (network) slicing solutions. The management of the various resource elements that form a slice is delegated to each of the providers that own those resources, via their (centralised) resource management systems. Whilst this preserves the confidentiality and security of those providers, it makes the overall slice management more complex, due to multi-domain interactions required after a slice has been put in place. As result, a slice resembles a loosely coupled set of resources rather than a fully-manageable abstract object.

Our end-to-end slice encompasses several *compute slice*s and *storage slice*s, along with *network slice*s, as its internal constituting elements. Each of those resource slices is built from resources, offered by a given provider, that can be managed independently and in full isolation of other resource slices. This is achieved by allocating on-demand a *separate* instance of a *Virtual Infrastructure Manager (VIM)* for a compute (or storage) slice [11], and an instance of a *WAN Infrastructure Manager (WIM)* for a network slice [18]. By following this approach, the multi-domain interplay is somehow simplified, as the provider that initiates the slice creation workflow will have direct management access to the sliced resources – as if they were locally available – thanks to the created abstractions and the dedicated management and control points. Finally, the allocated resources are all aggregated and exposed to a tenant as a single manageable object, which hides the distributed nature of the underlying multi-layered fog infrastructure. Such transparent usage of sliced resources, offered to a tenant via a dedicated set of abstractions, is the **second feature** of our slicing solution.

Likewise bare-metal clouds, the slice-based orchestration solution proposed here allows fully-isolated data centre resources to be allocated and utilised as part of an end-to-end slice. Those cloud resources can then be combined with other types of resources from different layers of the multi-provider cloud-to-edge continuum. This allows the seamless interwork of private and public infrastructures and can ultimately reduce the issues associated to vendor lock-in. The tenants are also allowed finer-grained customisation capabilities on the slice's resources and, differently from traditional multi-tenant clouds, are not restricted to the usage of solutions pre-defined by a provider. Each tenant can choose the most suitable VIM/WIM technologies for the delivery of their services and the achievement of the desired performance. This enables a high degree of resource isolation and can also minimise the likelihood of service performance degradation. Ensuring high isolation between resources allocated to different tenants, to support use-cases with demanding security and performance requirements – such as large-scale compute-intensive distributed services – is the **third feature** of our slicing concept.

The design and implementation of a system that supports the setup of *cloud-network* slices was presented in our previous work [19]. However, the problem of ensuring full isolation between different slices, and enabling mechanisms to map the tenants' service components onto the created slices automatically, were not investigated there, especially in the context of the cloud-to-edge continuum. Beyond our previous work, this paper addresses those open challenges and makes the following novel **contributions**:

- it introduces the notion of *end-to-end slice* and its formal definition via a graph-based model, which is used to select resources across the cloud-to-edge continuum (Section 3);
- it thoroughly discusses a system architecture that supports the on-demand creation of *end-to-end slices* (Section 4);
- it describes the above system's implementation and a workflow to allocate services transparently within fully-isolated *end-to-end slices* (Section 5);
- it demonstrates the effectiveness of the proposed solution in orchestrating resources in the cloud-to-edge continuum, and supporting the setup of IoT services, through a new set of experiments (Section 6).

A qualitative functional evaluation of our system shows that *end-to-end slice*s can be allocated **on-demand**, and under software control – in a matter of seconds – on a real testbed, with resources located across Europe and Brazil. Distributed services can be instantiated on the slices **transparently**, as the tenant's software systems need not be aware of any of the details of the underlying slicing, and are able to work without substantial modification. When *end-to-end slice*s are utilised for the deployment of compute-intensive IoT services at large-scale, our slicing approach proves to be effective in ensuring resource **isolation** and preserving the level of service **performance** expected by the tenants.

## 2. Related work

Different research communities, standardisation groups and vendors have worked independently on the topics of *orchestration* in the cloud-to-edge continuum, *network slicing* and *bare-metal clouds* for the past few years. Although relevant progress has been made in each of those areas, a unified orchestration approach that considers the intersection between these topics has not been proposed yet, and it is the main contribution of this paper. Existing work, related to the ideas presented in our paper, has been reviewed and it is discussed in this section.

### 2.1. Cloud-to-edge continuum orchestration

The authors of [6] present a recent comprehensive investigation of the current challenges and solutions in the area of fog orchestration. According to the survey, a few papers only present work that goes beyond the mere architectural design, and evaluate their proposals in a simulated environment; even less contributions describe implementations deployed on real testbeds. The authors of [20] describe a reference architecture of a Fog Computing Platform that targets Industrial IoT Applications. This solution provides both service and resource orchestration; it is based on deterministic virtualisation and networking to ensure safety and security along with interoperability. The work [21] is based on a centralised approach focused on resource orchestration, which uses deep-learning in a simulated environment in order to adjust the resource allocation at run-time. The authors of [22] discuss an approach to orchestrate the initiation of applications in the cloud-to-edge continuum, however, the resource orchestration is somehow abstracted, as a pool of infinite resources is considered for the cloud domains.

Differently from the actual system described in this paper, the work in [20,21] has not produced real implementations, and does not consider the usage of slicing and bare-metal clouds as part of the proposed solutions. A real solution was built and tested as part of the work [22] but, likewise other related work mentioned above, it is different from our approach as it does not support the unified orchestration of network and cloud resources across the cloud-to-edge continuum. Even though the concept of federation between public and private cloud providers is a well-known concept, a similar approach is not adopted in fog computing, as found from the previous analysis of the state-of-the-art. Therefore, further investigation on the topic is desirable, in order to achieve unified orchestration strategies able to ensure the seamless execution of services in a federated cloud-to-edge continuum of resources.

### 2.2. Network slicing

Network slicing has gained popularity with the 5G, as it is considered a key technology to facilitate the co-existence of multiple logical self-contained networks on a common physical infrastructure. Many standardisation activities in this area provide their network slicing definitions, as it is widely discussed in recent surveys [23–25]. The 3GPP SA2 defined a system architecture whereby a network slice can provide on-demand customised 5G network services by selecting specific control plane and user plane network functions [26]. The ETSI ZSM ISG worked on the 5G end-to-end network slicing management issues, and recognised SDN and NFV as enablers for multi-tenant and multi-domain environments in 5G infrastructures [27]. The IETF published the "Network Slicing – Revised Problem Statement" [28]. The ITU-T defined slices as isolated network partitions, each representing a unit of programmable network, computation and storage resources [29]. Finally, NGMN considered a network slice as an isolated, manageable and programmable entity that enables multi-service and multi-tenancy via the combination Service Instance Layer, Network Slice Instance Layer, and Resource Layer [10].

The slicing initiatives related to collaborative research projects have been extensively analysed in [23]. They have mainly driven the design and implementation of systems that support multi-service and multi-tenancy, either considering specific segments of the network or the end-to-end multi-domain network infrastructure. Focusing on QoS management in the 5G architecture, the PPP FP7 FIWARE [30], 5G-NORMA [13] and MIUR PLATINO project [31] have been working specifically towards the realisation of orchestration algorithms for control decisions, and different mechanisms for subjective QoS personalisation/differentiation. Projects such as 5G-Xhaul [32] and SELFNET [33] considered self-healing, self-configuration and self-optimisation capabilities of slicing for 5G networks. The work done in [34] and [15] defines models for the end-to-end mobile network infrastructure, which are used to determine the optimal placement of network slices, using either game theory or MILP. This is similar to the slice mapping problem discussed in this paper, although our model uses compute, storage and network slices as the building blocks of the end-to-end slices, instead of network slices only. Moreover, our framework does not prescribe the usage of a particular mapping algorithm, and was deployed on a real testbed rather than in a simulated environment.

Our previous paper [7] provides a qualitative comparison of existing work done on network slicing. Whilst the slicing aspects specifically related to networking and connectivity have been extensively investigated, few of those initiatives addressed cloud slicing and its combined usage with network slicing, especially across multiple administrative domains. The same work [7] also identified the existence of four *slicing operational modes*, and highlighted that the ordinary approach for sharing sliced resources is based on peer-to-peer interactions between orchestrators belonging to different providers. In contrast to the slicing approach presented in this work, as there is no obvious concept of slice abstraction, a slice becomes closer to a set of data structures and APIs, rather than an end-to-end isolated, fully-manageable object. Peer-to-peer also requires standardised interfaces and/or adaptations to the APIs between the tenant's OSS/BSS and the (slice-ready) orchestrator. Finally, due to the way the sliced resources are managed and exposed, this model does not allow for the selection of the technologies underpinning a slice. This limitation is addressed in our work by allowing the tenants to select those technologies that allow higher resource isolation and performance.

### 2.3. Bare-metal clouds

The multi-tenant nature of the VM-based clouds, raises substantial concerns for both security and performance. Since VMs from different users share the same physical servers, interference can be exploited to infer confidential information. Conflicts can also lead to unpredictable performance fluctuations on top of the already existing virtualisation overhead. With bare-metal clouds, single-tenant dedicated physical servers can be rented, in their entirety, to one user at a time. The user has exclusive, full access to the hardware and the complete freedom to run the OS of choice. This ensures the same level of security, performance and isolation as the physical server, as well as the elasticity of the cloud.

Bare-metal clouds are offered today *as a service* by some of the most popular public cloud providers. Amazon launched a new bare-metal option for its EC2 C5 server instances [35], which are commonly used for running compute-heavy workloads like batch processing, distributed analytics, and high-performance computing. Alibaba's ECS Bare Metal Instance [36] offers both the elasticity benefits of virtualisation and the performance advantages of physical servers via a next-generation virtualisation technology. IBM positions its Cloud Bare Metal Servers offering [37] as a more affordable alternative to AWS in many cloud computing scenarios, and runs more than 60 cloud data centres across 19 countries.

Private infrastructures can support bare-metal clouds through the usage of open source tools. Metal-stack is a software that provides an API for provisioning and managing physical servers in the data centre [38]. Likewise conventional cloud providers, users can manage their resources (servers, networks, etc.) by themselves, which effectively turns a data centre into an elastic cloud infrastructure. RackHD provides free, open source development tools and APIs that developers can use to automate hardware management and orchestration [39]. It serves as an abstraction layer between other management layers and the underlying, vendor-specific physical hardware. OpenStack Ironic aims to provision bare-metal machines instead of virtual machines, forked from the Nova bare-metal driver [40]. It is best thought of as a bare-metal hypervisor API and a set of plugins, which interact with the bare-metal hypervisors.

Bare-metal cloud offerings are publicly available, and private data centre can also support this model via the usage of open source tools. However, the integration of those resources as part of a unified infrastructure would be challenging, due to the lack of standardised interfaces and the potential issues associated to vendor lock-in. Even more complicated would be the usage of those resources as part of the continuum that includes various fog layers and the centralised cloud, because of the diverse underlying technologies involved. The end-to-end slice idea of this paper utilises compute slices built from data centre resources via a bare-metal-like approach. Thanks to the usage of resource partitioning mechanisms, and of the VIM on-demand model, the well-known drawbacks of bare-metal clouds are somehow ameliorated via enforcing mechanisms to adjust the amount of physical resources assigned to a compute slice at run-time, in order to trade-off tenants' QoS requirements with resource providers' policies (i.e., user density, resource costs, etc.). Most importantly, the allocated compute slices can be orchestrated via a unified approach thanks to the provided abstractions. This allows the interwork of heterogeneous types of compute resources in different fog layers and clouds, and their seamless integration with the connectivity provided by the network slices.

### *2.4. Discussion*

The analysis of the state-of-the-art on fog orchestration highlighted the lack of actual systems able to seamlessly integrate various resources across the cloud-to-edge continuum. Existing work in this area, considered in this section, produced either reference architectures or systems tested in simulated environments, whose main focus was on the management of compute resources. Research conducted in the area of network slicing, on the other hand, was mainly focused on the network management aspects and allows compute resources, assigned to different tenants, to share the same physical hosts, hence jeopardising the performance of the compute-intensive services assigned to those slices.

These observations provide evidence of the lack of a unified resource orchestration strategy, able to integrate the selection, configuration and management of compute and network resources, as part of a single abstracted object exclusively assigned to a tenant. Our work attempts to bridge this gap by devising and implementing the notion of *end-to-end slice*. The constituting compute (and storage) building blocks of those slices are allocated on-demand, and under software control, via a bare-metal-like approach; then, they are linked together via the dedicated connectivity resources provided by network slices. This creates an abstraction on top of the distributed resources of the cloud-to-edge continuum, and allows the tenants to use their service orchestration tools transparently. Thanks to their inherent high-level of isolation, end-to-end slices ensure that the services assigned to them are executed according to the level of performance expected by the tenants.

## 3. End-to-end slicing model

The main concepts of the end-to-end slicing solution, and of the associated business ecosystem, are presented in this section. After an introductory overview, a formal definition of our slicing approach is provided using a graph-based model.

The setup of and end-to-end slice involves three separate layers: *(i)* the *Resource Substrate*, *(ii)* the *Slice Substrate*, and *(iii)* the *Service Substrate*, represented on the right-hand side in Fig. 1. The figure also depicts how the various actors of the business ecosystem participate in the creation, delivery, and utilisation of the slices, and highlights how their roles are related to the three layers. The *Slice Provider* creates slices on-demand according to the *Tenant*s' requests. Based on the requirements for those slices and/or service requests (e.g., resource availability, business strategy, etc.), the Slice Provider acts as a broker for resources offered by the *Resource Provider*s. These expose (a share of) their available infrastructure and hand out the building blocks (e.g., compute, storage, and network resources) utilised by the Slice Providers for setting up the different end-to-end slices. We define those building blocks as the *Resource Slice*s; where each Resource Slice has an associated management and control point, i.e., either a VIM for compute and storage slices, or a WIM for network slices.

A slice of the infrastructure's resources, distributed across the cloud-to-edge continuum, is delivered to the Tenant that originally asked for it. The Tenant will then be able to fulfil the instantiation of the services requested by the *End-users*. In this ecosystem, as well as dynamic creation, an end-to-end slice can also be adjusted (namely it can grow or shrink) and deleted at run-time by considering the Tenant's demands, the requirements of the services, together with the status of the whole distributed infrastructure. The end-to-end slicing approach to which the above actors are pertained is now formally described via a graph-based model.

### *3.1. Resource Substrate*

The Resource Substrate is a federated infrastructure that includes different types of resource domains (from the edge to the cloud), from various providers, as shown in the bottom layer of Fig. 1. By defining $N_c^R$ as the set whose elements are all the compute domains, and $N_s^R$ as the set formed of all the storage domains, the Resource Substrate can be modelled as a weighted undirected graph $G^R = (N^R, E^R)$, where $N^R = N_c^R \cup N_s^R$, and $E^R$ includes the (physical) *network* resources between the compute and storage domains of $N^R$.

A compute domain $n_c^R \in N_c^R$ consists of a number of compute resource elements (i.e., physical servers) and has $c\ (n_c^R)$ CPU capacity and $m\ (n_c^R)$ Memory capacity. Likewise, a storage domain $n_s^R \in N_s^R$ includes a set of storage resources (i.e., storage devices) and $s\ (n_s^R)$ describes the associated storage capacity. Finally, $e^R(i, j) \in E^R$ represents network resources (i.e., physical link) between two resource domains $i$ and $j$. It has an associated bandwidth capacity $b(e^R)$ and delay $d(e^R)$, which are modelled in the graph $G^R$ as attributes of the corresponding edge. Multiple links of $E^R$ constitute a network path $P$. $P^R$ refers to the set of all substrate paths, and $P^R(u, v)$ denotes the set of substrate paths between two resource domains $u$ and $v$.

### *3.2. Slice Substrate*

The Slice Substrate is represented as the middle layer of Fig. 1. It is a group of (heterogeneous) resources from domains of the Resource Substrate that have been logically aggregated by a Slice Provider. In our model, a Slice Substrate is built on *compute slices*, *storage slices* and *network slices*, provisioned by various Resource Providers via reserving a dedicated amount of resource in specific domains. Those *Resource Slice*s are combined and utilised in a complementary way, becoming the building blocks of a single Slice Substrate. Similar to the Resource Substrate, a Slice Substrate is modelled as a weighted undirected graph $G^S = (N^S, E^S)$, where $N^S = N_c^S \cup N_s^S$ is the set that includes all the *compute* $(N_c^S)$ and *storage* $(N_s^S)$ slices; $E^S$ is the set of *network* slices that interconnect them. This model assumes that a VIM instance is created for each *compute* slice $n_c^S \in N_c^S$ and storage slice $n_s^S \in N_s^S$, in order

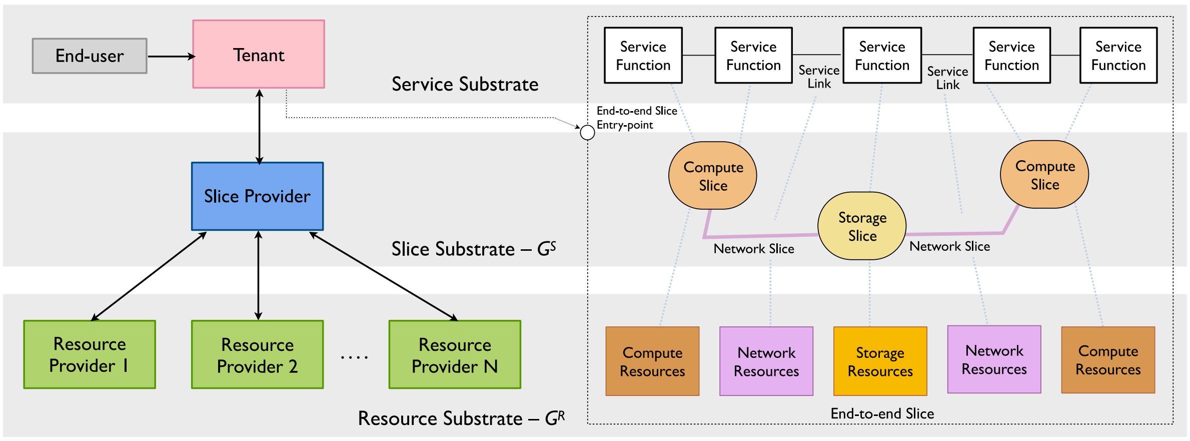

**Fig. 1.** Slicing model and business ecosystem.

to manage and control the related resources. Given a *network* slice $e^S(m^S, n^S)$, which interconnects two compute (or storage) slices $m^S$ and $n^S$, a separate WIM instance is created, allowing similar management and control functions to be performed on the associated network resources. A *compute* slice $n_c^S$ has some associated CPU $c\,(n_c^S)$ and Memory requirements $m\,(n_c^S)$, while a *storage* slice $n_s^S$ demands $s\,(n_s^S)$ storage resources; a *network* slice $e^S$ has an associated bandwidth constraint $b\,(e^S)$ and maximum tolerable delay $d\,(e^S)$ between its end-points.

Considering the capabilities of the Resource Substrate's domains, and the demands associated to the Slice Substrate's elements, the following expressions can be introduced in order to describe the residual capacity of a resource domain. In particular, the CPU capacity of a compute domain $n_c^R$ is

$$R_C\,(n_c^R) = c\,(n_c^R) - \sum_{\forall n_c^S \in n_c^R} c\,(n_c^S)\ , \tag{1}$$

where $\forall n_c^S \in n_c^R$ are the compute slices hosted by the domain $n_c^R$. A similar expression can be formulated for a domain's residual Memory capacity $R_M\,(n_c^R)$. The residual capacity of a storage domain $n_s^R$ is defined as

$$R_S\,(n_s^R) = s\,(n_s^R) - \sum_{\forall n_s^S \in n_s^R} s\,(n_s^S)\ . \tag{2}$$

Finally, the residual bandwidth capacity associated to the network resources $e^R(i,j)$, between domains $i$ and $j$, is defined as

$$R_B\,(e^R) = b\,(e^R) - \sum_{\forall e^S \in e^R} b\,(e^S)\ , \tag{3}$$

where $\forall e^S \in e^R$ are all the network slices that use the resources of $e^R$. From the previous expression, the available bandwidth capacity of a Resource Substrate's path $P \in P^R$ can be expressed as

$$R_B\,(P) = \min_{E^R \in P} R_B\,(E^R)\ . \tag{4}$$

### 3.2.1. Slice mapping

Based on the previously presented resource model, we define the *slice mapping* as the problem of finding a suitable assignment for the compute, storage and network elements of a Slice Substrate onto the available domains of the Resource Substrate. Referring to the previously introduced graph notation, the problem can be seen as two steps: the *node assignment*, i.e., mapping compute and storage slices on resources of selected compute and storage domains; the *link assignment*, i.e., mapping the network slices on network domains' paths that interconnect the compute and storage domains.

Each resource slice from a given Slice Substrate is assigned to a different resource domain by a mapping function. More specifically, $M_C : N_c^S \to N_c^R$ assigns compute slices to compute resources (domains) such that $\forall n_c^S, m_c^S \in N_c^S$,

$$M_C\,(n_c^S) \in N_c^R \tag{5}$$

$$M_C\,(m_c^S) = M_C\,(n_c^S),\ \Leftrightarrow m_c^S = n_c^S \tag{6}$$

subject to

$$c\,(n_c^S) \le R_C\,(M_C\,(N_c^S)) \tag{7}$$

$$m\,(n_c^S) \le R_M\,(M_C\,(N_c^S)) \tag{8}$$

The mapping function first identifies a compute domain $\hat{n}_c^R \in N_c^R$ based on the residual CPU ($R_C$) and Memory ($R_M$) capacities of all the available compute domains. Then a subset of $\hat{n}_c^R$ resources are specifically assigned to $n_c^S$. We use the notation $\hat{n}_c^R\,(n_c^S)$ to indicate the resources of domain $\hat{n}_c^R$ allocated to $n_c^S$. The residual capacities $R_C\,(\hat{n}_c^R)$ and $R_M\,(\hat{n}_c^R)$ are finally updated.

A similar mapping function $M_S : N_s^S \to N_s^R$ can be defined for assigning storage slices to available storage domains. Lastly, the mapping function $M_E : E^S \to P^R$ assigns a network slice to a substrate path between the domains that host the compute/storage slices to be linked, $\forall e^S(m^S, n^S) \in E^S$,

$$M_E\,(m^S, n^S) = P \tag{9}$$

$$P \in P^R\,(M_C\,(m^S), M_C\,(n^S)) \tag{10}$$

$P$ from Eq. (10) is chosen such that

$$b\,(e^S) \le R_B\,(P) \tag{11}$$

$$d\,(e^S) \le D\,(P) = \max_{e^R \in P} d\,(e^R) \tag{12}$$

The slicing model here revolves around the principle of exclusively assigning dedicated partitionable resource elements to different Slice Substrates. More formally, given two Slice Substrates $G^{S_i}$ and $G^{S_k}$, the following expression holds for the compute resources

$$n_c^R\,(n_c^S) \cap n_c^R\,(m_c^S) = \emptyset\ , \tag{13}$$

with $n_c^S \in N_c^{S_i}, m_c^S \in N_c^{S_k}, \forall i \neq k$. Similar expressions can be defined for the storage slices and network slices. These mechanisms can be applied according to the resource isolation and performance requirements expressed by the Tenants. For bare-metal compute slices, physical machines are considered as the elements of a compute domain $n_c^S$, whereas the elements of a storage domain $n_s^S$ can either be whole discs or disc partitions. Two different network slices can share the same physical path of a network domain and, thanks to state-of-the-art SDN techniques, separate virtual networks can be created for different network slices by the control plane. This is achieved by defining packet-handling rules that are then propagated to the data plane devices of that network domain for execution. Such a dynamic approach ensures a full separation between the involved network flows and good performance, without the need to rely on separate physical network devices.

Once created, a Slice Substrate $G^S$ can dynamically be adapted, at run-time, according to the Tenant's demands (such as target KPIs, QoS constraints, etc.) and the status of the underlying resource infrastructure. More specifically, each of the resource slices belonging to the sets $N_c^S$, $N_s^S$ and $E^S$ can be adjusted by adding (scale up) or removing (scale down) individual elements (such as servers, discs, VMs, disc partitions, overlay networks, etc.) selected from the respective resource domains; more resource slices (from other resource domains) can also be added (scale out) or the removed (scale in) to/from those sets at run-time, throughout the Slice Substrate's lifetime, in order to comply with the requirements specified by a Tenant [19].

### 3.3. Service Substrate

The Service Substrate is represented by the top layer of Fig. 1. It includes different *Service Functions* associated with a Tenant's service instance, as well as the *Service Links* that interconnect them. Although this substrate does not represent a constituting part of the slice, it can be seen as the reason that triggered its creation in first instance. In the cloud-to-edge continuum context considered in this paper, a service is a topology of virtual compute and storage entities (i.e., VMs, Containers, Virtual Storage Blocks, etc.), interconnected via virtual links, that need to be distributed across different geographical locations in order to serve certain users and/or minimise the perceived (IoT) data processing latency [41].

The layered approach to slicing presented in this section provides a Tenant with an abstracted partition of resources of the cloud-to-edge continuum. Whilst this may look the same as a centralised set of resources, the underlying resource elements are in fact geographically distributed. Moreover, from a management and control perspective, the composing resource slices are not much different from a whole resource domain, except from the fact that they are formed of a smaller set of resources. By designing such an architectural abstraction and the run-time elements, the problem of instantiating a Service Substrate can be performed transparently by a Tenant using their existing software systems, regardless of the underlying (slicing) resource management approach.

## 4. System design

A system architecture that utilises the end-to-end slicing model discussed in Section 3, is described here. The architecture includes four main subsystems, namely: the *Tenant*, the *Slice Provider*, the *Resource Provider*, and the *Resource Marketplace*. How these subsystems are related to the three substrates of Fig. 1 is shown in Fig. 2. The description of the functionalities of this slicing system is the main subject of this section, together with some of the research into end-to-end slicing conducted by the NECOS project [42]. Then Section 5 will present a detailed description of its prototype implementation.

### 4.1. Tenant

The Tenant subsystem provides the mechanisms whereby a Tenant can request the dynamic instantiation of end-to-end slices. A new slice instance is created from a blueprint, which defines the desired Slice Substrate according to one of the following options: *(i)* the amount of resources to be allocated in the compute, storage, and network slice parts; or *(ii)* a list of desired slice requirements (e.g. resource facing KPIs, geographic locations, etc.); or *(iii)* the type of services that will run on the slice. The Tenant triggers the slice creation by sending the above blueprint to a Slice Provider (in Fig. 2) chosen according to particular constraints, e.g., locality, business relationships, contractual obligations, etc.

A Tenant utilises the reference to a newly allocated Slice Substrate in order to deploy the Service Substrate (i.e., the Service Functions and Service Links), requested by the End-user, onto the allocated sliced resources. Thanks to the abstractions created by the Slice Provider, the implementation details of a Slice Substrate are not directly exposed to the Tenant, who is rather offered access to an abstracted topological representation of that slice of the cloud-to-edge continuum.

### 4.2. Slice Provider

The Slice Provider subsystem includes the functionalities for managing and orchestrating end-to-end slices, namely: *(i)* receiving and processing the Slice Substrate blueprints sent by the Tenants; *(ii)* searching and selecting slice resource availability among different Resource Providers; *(iii)* requesting the allocation of resource slices to Resource Providers based on such availability; *(iv)* abstracting all of the allocated resource slices as a single Slice Substrate; *(v)* performing the allocation of the Service Substrate's elements on the slice resources, based on the service placement information requested by a Tenant; and *(vi)* ensuring that the slice performance is compliant with the run-time requirements expressed by a Tenant.

Fig. 2 shows that the Slice Provider can interact with the *Resource Marketplace* via the corresponding *Marketplace Interface*. The Marketplace takes care of finding a set of Resource Providers able to offer resources for the compute, storage, and network slices specified in the slice blueprint. The Slice Provider utilises the *Instantiation and Runtime Interface* to contact those particular Resource Providers and to request the allocation of resources for the above slices.

The Slice Provider is responsible for aggregating resource slices from different Resource Providers as a single end-to-end slice object. This process is performed by collecting the handles to the allocated VIM/WIM instances for the various resource slices, followed by the registration of the new Slice Substrate, and by its delivery to the Tenant. These functionalities, whereby an end-to-end slice is created by the Slice Provider via interworking with the other subsystems, is consistent with the *first* of the *three features* of our slicing solution (presented in Section 1.1), i.e., it is performed in an on-demand, software-based fashion.

After an *entry-point* to the end-to-end slice is allocated, the Tenant can utilise it to retrieve the topology of the Slice Substrate and specify how the Service Substrate should be mapped onto the slice resources. The implementation details of the Resource Substrate are not exposed by the Slice Provider to the Tenant, who only sees the slice abstract topology. The Service Substrate mapping information is specified by the Tenant via a descriptor, and the Slice Provider is then responsible for its translation into the actual entities (e.g., Virtual Machines, Containers, Virtual Links, etc.) that need to be deployed on the underlying compute, storage, and network slices. These abstraction mechanisms are

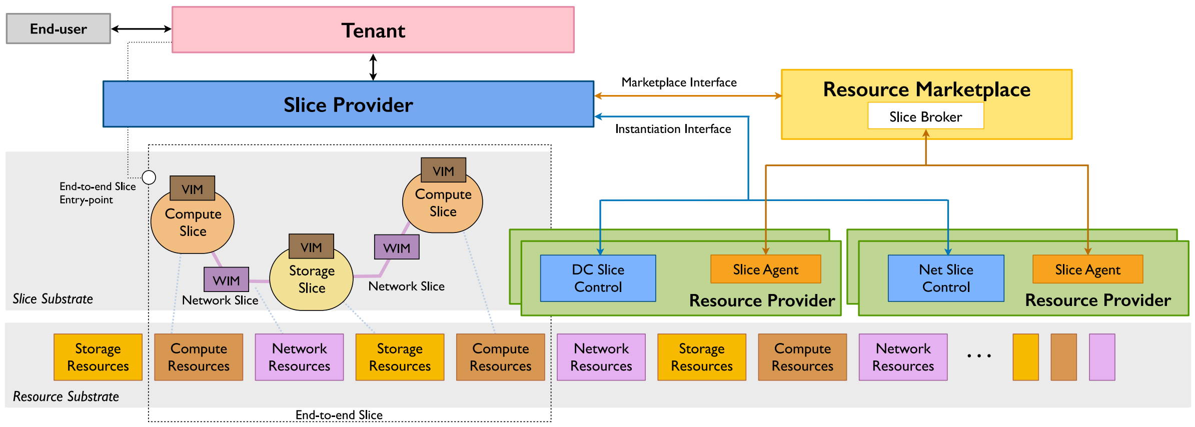

**Fig. 2.** Slicing system architecture.

paramount, as they allow the decoupling of the Tenant's components that deal with the instantiation of services from the technologies that underpin a slice. This concept is related to the *second feature* of our slicing approach, namely transparency: the Tenant's functionalities should work regardless of both the underlying resource management strategies and whether the resources have been sliced or not.

The runtime management of the Slice Substrates is the last function performed by the Slice Provider. It is enforced by adjusting either the number of elements within the existing resources slices (vertical slice scaling) or the number of resource slices of a given Slice Substrate (horizontal slice scaling). These scaling operations can both be performed by interacting with relevant Resource Providers, and by requesting changes to the current resources allocation via the *Instantiation and Runtime Interface*, according to the runtime performance requirements expressed by the Tenants.

### 4.3. Resource Marketplace

The Resource Marketplace represents the rendezvous point between Resource Providers and Slice Providers that wish to trade resources and build end-to-end slices on-demand. This market-inspired mechanism for sharing resource information fits perfectly with the dynamic nature of an end-to-end slice, as opposed to the usage of a-priori federation agreements between providers, which are difficult to change in relatively short time frames. Resource Providers can dynamically join the slicing ecosystem and share information about their resource availability. Similarly, Slice Providers can gather that information and use it to request the allocation of relevant resource slices, based on specific Tenants' constraints [43].

The above dynamic trading scenario is facilitated by the inherent resource partitioning and isolation of the slices, and by the usage of the VIM/WIM on-demand approach. These allow sharing resources among different providers more easily when compared to traditional peer-to-peer federation models. Resources can be managed and controlled by a Slice Provider as if they were locally available, without the burden of interacting with a shared domain's management system once a slice has been setup. This considerably simplifies the management of the distributed resources, minimises the establishment of inter-provider federation mechanisms, and can ultimately ensure a stronger level of isolation and security [7].

The Resource Marketplace is built as a dynamic Pub/Sub system, whereby Resource Providers and Slice Providers can trade the resources required for the setup and delivery of the desired Slice Substrates to the Tenants. In each Resource Provider, dedicated *Slice Agent*s publish information about the availability of resources in their pertaining compute, storage and network domains. This includes an abstracted topological view of the domains' interconnections [44], together with the latest measured residual capacity expressed by Eqs. (1)–(3). The price charged for the allocation of those resources can also be conveyed as part of the published information [45].

On the Slice Provider side, a *Slice Broker* gathers and filters this information by subscribing to updates for resources that match desired features only. In other words, a Slice Provider can request updates for resource domains that are, e.g., specifically available in a particular geographical area (or equivalently fulfil some specific delay constraints) and/or have a minimum amount of residual compute (CPU and Memory), storage or network (bandwidth and delay) capacity. The domains topological information, and the associated resources availability, are finally used by a Slice Provider in order to build the Resource Substrate graph $G^R$, and to perform the Slice mapping process initially introduced in Section 3.2. A preliminary discussion of the Resource Marketplace was also presented in [45], however, further slice mapping approaches are investigated in this paper considering the proposed graph-based resource model, as will be detailed in Section 5.2.2.

### 4.4. Resource Provider

The Resource Provider includes the functionalities for the allocation of slices of resources from the cloud-to-edge continuum, along with their associated management and control points. This is performed by partitioning compute/storage resources, and by interconnecting them via dedicated network resources. A *DC Slice Control* function is in charge of allocating resources for either compute slices or storage slices, and similarly, a *Net Slice Control* function partitions the connectivity resources of a network domain and assigns them to different network slices, as shown in Fig. 2.

The *DC Slice Control* creates a *compute slice* (*storage slice*) by selecting a subset of computational (storage) resources from a domain of the cloud-to-edge continuum, according to the performance requirement expressed by a Tenant. For bare-metal slices, physical resources of a given resource domain (e.g., a group of resource-constrained devices, a cloudlet, a data centre, etc.) are bijectively assigned to a compute (storage) slice, in order to dynamically create a "smaller" resource domain on-demand. A VIM instance is also created and activated on-the-fly for a compute (storage) slice as requested by the Tenant, allowing a fine-grained management and control of the associated resources during the allocation of the service functions. Different types of VIM can be requested in order to trade off resource manageability versus performance overhead (e.g., from full virtualisation to bare-metal resource access). Additional details and discussion about this data centre slicing model can be found in [11]. Nonetheless, this subsystem may also deliver different flavours of resource slices to Tenants that do not require such a high level of isolation and performance. This can be achieved, for instance, by creating a shim onto a shared instance of an existing VIM.

In a similar way, the *Net Slice Control* partitions and assigns resources of a network domain to a *network slice*. More specifically, it enables the dynamic activation of virtualised network paths (such as, MPLS LSPs, GRE tunnels, VxLAN connections, optical lambdas, etc.) on a domain's physical networking resources. Symmetrically to the data centre slicing, an on-demand instance of a WIM is created, allowing in-depth management and control over the sliced network resources. The WIM performs the mapping of different network flows, associated to the service instances, on the sliced resources. It also supports the collection of performance and fault information, facilitating monitoring and alarm management of those interconnections. According to those data, further control can be exerted on the elements of a network slice, in order to adjust its capacity in an on-demand fashion. As for the compute slices, some Tenants may not require a separate WIM instance. In this case, a WIM agent may be allocated and configured to interact with an already available shared WIM. Further details on this network slice management model, and on the usage of the WIM on-demand, can be found in [18]. Examples of how the WIM on-demand has been used by Telco operators as an enabler for network softwarisation can be found in [46,47].

In the context of the system architecture shown in Fig. 2, the main purpose of the Resource Provider is to support functionalities and mechanisms related to the *third feature* of our slicing solution, namely guaranteeing full resource isolation among different end-to-end slices.

### 4.5. Summary

The system architecture here was designed around the actors of the business ecosystem presented in Section 3: the *Tenant*, the *Slice Provider*, and the *Resource Provider*. The functionalities of its composing subsystems deal with realising the end-to-end slicing model, whereby resources of the cloud-to-edge continuum can be traded by different providers via a *Resource Marketplace*, and different *Slice Substrates* can be created and delivered to the *Tenant*s for the instantiation of the required distributed services. Considering the *three features* of our slicing approach, it has been discussed *(i)* how those subsystems provide mechanisms to allocate resources *dynamically*; *(ii)* how those resources can be accessed *transparently* by a Tenant via a dedicated set of abstractions; and *(iii)* how full *isolation* is enforced. A software implementation of this system, and its evaluation, will be presented in the next two sections.

## 5. System implementation

The details of an experimentation environment, built via a prototype implementation of the system described in Section 4, is presented here. This prototype supports the main features of the Tenant, of the Slice Provider, and of the Resource Provider subsystems; it also includes the implementation of a Slice mapping algorithm for the Resource Marketplace. Fig. 3 highlights these software components, and shows how each of them is related to the *Resource Substrate*, to the *Slice Substrate*, and to the *Service Substrate*, as well as the way they interwork once deployed on the testbed. As this system was designed to facilitate the orchestration of resources in the cloud-to-edge continuum, the infrastructure on which it was deployed is fully distributed, with computation and network resources located both in Europe and Brazil.

### 5.1. Testbed setup

Three resource-constrained *compute* domains were created from resources located at University College London (UCL), in the UK. Each of those cloudlets, featured three physical servers with 4x Intel Xeon E5520 (16 cores) running at 2.27 GHz and 32 GB of memory. An existing 1 Gbps LAN interconnection between these mini DCs mimicked the function of multiple *network* domains. These were partitioned during the execution of the tests, when overlay connections were created for different network slices, as it is shown in Fig. 3. An additional data centre, with similar resources, was located at the Federal University of Pará (UFPA), in north Brazil. In order to build a cloud-to-edge continuum scenario, and demonstrate the distributed nature of the slicing approach, these resources were used as a cloud domain, inter-connected to the 3 mini DCs in the UK through the Internet.

Instances of the Tenant, of the Slice Provider, and of the Resource Provider components were deployed on a portion of such infrastructure specifically reserved to host the control plane. Using separate geographical locations for those components proved that the exchange of management and control information, required to setup and keep the slices up and running, was not impacted by any intrinsic delay in the communication between the entities of the testbed. Moreover, the inherent dispersed nature of the control plane allows the system to minimise the likelihood of introducing bottlenecks and central points of failure into the system. Considering the current system implementation, the compute resource utilisation of the associated control plane components was negligible, and it never impacted the execution of the workflow that is presented in this section.

This experimentation environment was devised to validate the concept of end-to-end slice, whose composing elements are allocated via the bare-metal model with on-demand VIM instances. The experiments are based on the *lightweight* VLSP VIM [48] for managing sliced compute resources, together with a newly devised lightweight WIM prototype for the management of sliced network resources. Thanks to their small footprint, these management and control points could easily be instantiated on-demand, in a matter of seconds, during the setup of the resource slices. Moreover, they suit the setup of service components in resource-constrained domains. The VLSP VIM allows for the allocation of lightweight virtual Service Functions, implemented as independent Java containers. These are, in turn, interconnected via virtual Service Links based on a modified version of the UDP protocol, called USR [48]. The WIM utilised in this experimental environment complements the features of the VLSP VIM, and allows the mapping of USR-based virtual links on its own managed set of sliced network resources. It also provides basic management and control mechanisms, such as retrieving the slice endpoints, collecting relevant network utilisation KPIs, etc.

### 5.2. End-to-end slice creation workflow

The implementation details and the role of the components of the system in Fig. 3 are now discussed. This description includes the way the Tenant and the Slice Provider inter-play when a new end-to-end slice is requested, as well as the details of how the required resources are allocated via the Resource Providers. Each of the steps, 1 to 15, is labelled on Fig. 3, and highlighted in the description.

#### 5.2.1. Slice request submission

A new end-to-end slice is defined by a Tenant, via a blueprint, using a YAML *Slice Descriptor*. The descriptor specifies the compute, storage, and network slices, the amount of required resources and the desired types of VIMs and WIMs. Additional

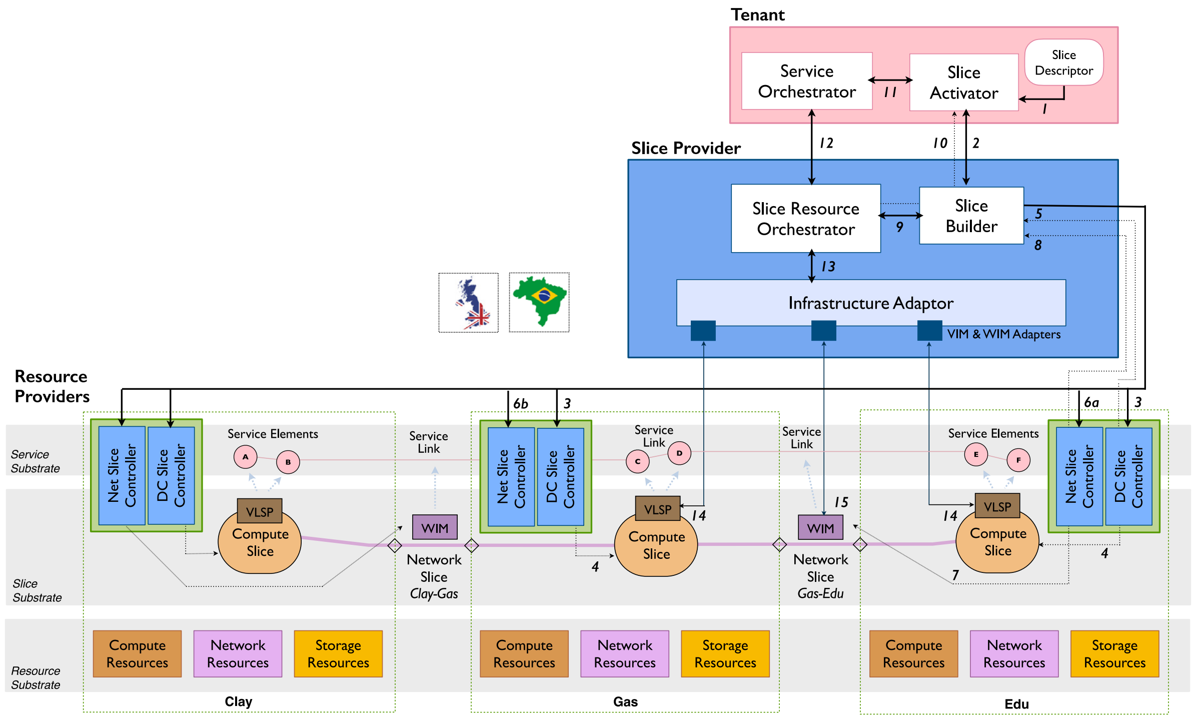

**Fig. 3.** Slicing system implementation: workflow and deployment on the testbed.

requirements may include geographical location, or delay constraints, for some of those resources slices; these would be defined according to the potential distribution across the cloud-to-edge continuum of users, data sources, data processing functions, etc. Finally, any runtime requirements that the Tenant may wish to be fulfilled throughout the lifetime of the Slice Substrate can also be included to the descriptor. The *Slice Activator* module provides a REST interface that an administrator (or external software component) can use to upload and submit to the system the above Slice Descriptor [step 1]. The Slice Activator acts like an agent of the Tenant as it passes on the slice requests to the Slice Provider, and keeps track of the status of all the slices requested by that Tenant.

### 5.2.2. *Slice Substrate mapping*

Once the Slice Descriptor is received and validated by the Slice Activator, it is sent to the *Slice Builder* module of the Slice Provider [step 2]. This parses the descriptor's content, which includes the number of resource slices and the associated demands, and builds the graph $G^S$ for that Slice Substrate. This is then forwarded to the *Resource Marketplace* shown earlier in Fig. 2. Inside the Resource Marketplace, the Slice Broker holds an up to date view of the resources available in several domains, i.e., the graph $G^R$, which was built through the information pushed by the associated Slice Agents. (Please note that these entities are not included in Fig. 3 for the sake of simplification.) Based on the above graphs $G^S$ and $G^R$, the Slice Broker performs the Slice mapping first presented in Section 3.

Many algorithms to embed virtual network requests into a physical resource infrastructure [49] exist and may also be applied to this problem. Our approach builds on the solution presented in [50], and proposes a slightly different version of the mapping algorithm, in order to place resource slices onto available compute, storage and network domains. The algorithm takes into account the fact that heterogeneous types of resources can be considered during the mapping, and that network slices may also have a maximum associated delay [51], as expressed by Eq. (12). Similar to [50], this solution uses a one-stage backtracking placement algorithm, based on breadth-first search (BFS), and maps compute slices, storage slices and network slices at the same stage. The algorithm also computes a topological ranking of the nodes in $G^S$ and $G^R$, similar to the PageRank used by Google's search engine [52]. In the long-term, this approach can potentially increase the overall number of (slice) request that are successfully mapped, and can ultimately lead to higher revenues for the Resource Providers [50]. Moreover, the proposed one-stage mapping can reduce the utilisation of network resources compared to uncoordinated node-link mapping approaches [49]. Nonetheless, alternative algorithms may be selected and utilised by our framework to support several types of constraints and optimisation goals.

The Slice mapping algorithm starts by creating a BFS tree for the nodes in $N^S$, where the root is the node with the highest rank. For each resource slice in the above tree, a list of candidate resource domains is built considering the slice type (compute or storage) and the domain's residual capacities described by either Eq. (1) or Eq. (2). More specifically, the list will include only those domains whose available residual compute $R_C$ and $R_M$ (or storage resources $R_S$), and the sum of available bandwidth resources $R_B$ (Eq. (3)) from the adjacent network domains, are greater than those demanded by the slice. For a compute slice, this can be expressed by the constraints of Eqs. (7) and (8). Hence, domains that are not compatible with the geographical constraints associated to a slice (note that this can also be expressed in terms of delay from a given location) will not be added to the list. Each of the candidate lists is finally sorted by domain ranking in non-increasing order.

The mapping of the slices onto the available resource domains is performed by traversing the above BFS tree starting from its root, which is assigned to the domain with the highest ranking. Subsequent slices in the tree are then mapped one by one by considering the following criteria. We assume that a slice to be mapped at the current algorithm's iteration, $n^S$, has an associated parent slice $m^S$ in the tree, which was mapped on a domain $m^R$ during the previous iteration. The candidate domains list of $n^S$

is further restricted by computing the shortest path from those domains to $m^R$, and selecting only the graph nodes that are within $h$ hops (initially $h = 1$).

The domains are further sorted based on their distance from $m^R$. Starting from the first domain in the list, the *k-shortest paths* from $m^R$ are calculated using the algorithm described in [53]. Then, it is checked whether the network slices attached to $n^S$ can be mapped on those paths considering the bandwidth demands and delay constraints described by Eqs. (11) and (12). If none of those path fulfils the specified network slices' constraints, then the next domain in the candidate list is selected, and the associated k-shortest paths are evaluated.

If all the candidate domains in the list have been considered, and no suitable mapping has been found, then $h$ is incremented by one and a new list of candidate domains is created. This can be iterated up to $h = maxHops$ times, and if the mapping is still unsuccessful at that point, then the algorithm backtracks to the previous stage, and a different mapping solution is searched for the parent node $m^S$. As in [50], the maximum number of allowed backtrack steps is bounded by a parameter $\theta$, which can be selected in order to limit the overall computational complexity of the Slice mapping procedure and make it almost comparable with polynomial-time algorithms.

### 5.2.3. Slice Substrate instantiation

After a suitable mapping for the composing slices of $G^S$ is determined by the Resource Marketplace, as described in the previous subsection, a list with the resource domains selected from $G^R$ is returned to the Slice Builder. We assume $G^S$ consisted of three compute slices and two network slices (storage slices are not considered in order to simplify the discussion); also, three compute domains, among those available in the cloud-to-edge continuum, were chosen by the mapping algorithm according to the specified compute and network constraints, namely *Clay*, *Gas* and *Edu*, together with the network domains *Clay-Gas* and *Gas-Edu*.

The Slice Builder is now informed by the Resource Marketplace about the entry-points of all the above resource domains to be contacted. First, this information is used to interact with the associated DC Slice Controllers, and to request the allocation of resources for the *compute slices* [step 3]. The required resources are reserved on those domains, and an instance of the VLSP VIM is created on-demand for each compute slice. In this example workflow, we assume bare-metal slices have been requested, hence physical resource elements in each domain are exclusively assigned to the resource slices; moreover, as expressed by Eq. (13), a given resource element will never be shared between compute slices of different Slice Substrates. Once all of the above compute slices have been allocated [step 4], a handle to each of the dynamically instantiated VIMs is passed back to the Slice Builder [step 5]. (The arrows related to step 5 have been drawn for the *Edu* domain only, but they also apply to the *Clay* and *Gas* domains.)

A symmetric approach is performed for the allocation of the *network slices*, namely the setup of the connectivity across the nominated network domains considering the requested bandwidth and delay constraints. As our framework was designed to be independent of the particular data plane technology of each resource domain, it can seamlessly interwork with different types of network resources. For this workflow description, we assume that the data plane of each network domain is managed by a central controller, and the setup of the connectivity between different compute (or storage) domains is performed via a distributed peer-to-peer approach.

Based on the (network) domains' entry-points returned by the Resource Marketplace, the Slice Builder starts the allocation of the required network slices by sending separate requests to the designated Net Slice Controllers. An instance of a WIM is created on-demand for all the network slices, and the handles to those WIMs are returned to the Slice Builder. More specifically, to link the compute slices in the *Gas* and *Edu* domains via the *Gas-Edu* network slice, the Slice Builder first sends a request to the *Edu* Net Slice Controller [step 6a], which contains the information about the *Gas* destination domain. The Net Slice Controller identifies the (physical) ingress/egress points to be used in order to reach the nominated destination — this is represented in Fig. 3 by the diamond shape at the left edge of the *Edu* resource domain. The network resources inside the *Edu* domain are now configured: a path from the selected ingress/egress point towards the previously allocated compute slice is calculated by the Net Slice Controller considering the given network slice constraints (see Eqs. (11) and (12)).

Likewise, the setup of the other end of the *Gas-Edu* network slice is driven by the Slice Builder via interaction with the *Gas* Net Slice Controller [step 6b]. Finally, an overlay connection that fulfils the requested network bandwidth constraints is created on a path $P$ between the ingress/egress points of the *Gas* and *Edu* domains. The handle to a WIM instance, allocated on-demand by one of the Net Slice Controllers [step 7] is finally sent back to the Slice Builder [step 8]. This WIM handle is passed on by the Slice Builder to the *Slice Resource Orchestrator* [step 9], where it will be utilised in order to control and monitor the network slice between the *Edu* and *Gas* domains. The WIM will also be used, at service instantiation time, to map network flows on the reserved network slice's resources. The end-to-end allocation of the Slice Substrate terminates when the above steps 6–9 are reiterated for the setup of the *Clay-Gas* network slice.

The Slice Resource Orchestrator is now aware of all the components of the Slice Substrate $G^S$. The topology of this newly allocated end-to-end slice, which includes the references to the composing resource slices and to the allocated VIMs/WIMs (in the relevant domains), is finally ready to be registered in an internal database of the Slice Provider.

### 5.2.4. Service Substrate instantiation

The Tenant is notified, and provided with the handle to the allocated end-to-end slice $G^S$. This is represented in Fig. 3 by the thin dotted line that connects the Slice Resource Orchestrator to the Slice Activator through the Slice Builder [step 10]. The Service Orchestrator is informed by the Slice Activator about the availability of $G^S$. The slice handle is passed to the Service Orchestrator [step 11], who will now be able to trigger the allocation of a Service Substrate on the available sliced resources. The Service Orchestrator utilises the received Slice Substrate's handle to retrieve the abstract topological view of $G^S$, and then determining the placement of the service functions and links onto the available Slice Substrate. The discussion of the particular mechanisms and algorithms utilised by the Service Orchestrator is out of the scope of this paper, but it can be assumed their execution to be similar to [54]. It should be noted that these functions can be performed transparently by the Service Orchestrator, thanks to the way the selected resources of the cloud-to-edge continuum have been sliced, aggregated and finally exposed to the Tenant, as a single Slice Substrate.

The placement decision made by the Service Orchestrator is now forwarded to the Slice Resource Orchestrator [step 12]. The instantiation of the required Service Elements (both Functions and Links) is driven by the Slice Resource Orchestrator by interacting with the VIM/WIM entry-points associated to the resource slices of $G^S$. An adaptation layer, implemented by the Infrastructure Adaptor module, provides an abstraction over the mechanisms to perform the Service Elements instantiation and monitoring, and makes the process independent from the underlying type of VIMs/WIMs [step 13].

In this experimental environment, the Service Substrate is deployed onto the Slice Substrate as a set of VLSP Virtual Service Functions and Links [48]. The Slice Resource Orchestrator deals with the instantiation of such service components on the underlying resource slices, based on the received service placement information. In particular, Service Functions are allocated inside the nominated compute slices via the associated VIM handles [step 14]. Service Links between Service Elements within the same compute slice are also created via interacting with the associated VIM. This is shown in Fig. 3, for instance, for the link between Service Functions *C* and *D* inside the *Gas* domain.

Service Links between Service Functions deployed in different compute domains are instantiated by the Slice Resource Orchestrator via interacting with the WIM of the connecting network slice. The WIM takes care of mapping a new network flow, associated to the Service Link, onto the overlay connection that was previously allocated for that network slice. This is shown in Fig. 3, e.g., for the Service Link between the Service Functions *D* and *E* in the *Gas-Edu* network slice [step 15]. After all those 15 steps have been performed, both the Slice and the Service are instantiated and deployed, hence the runtime phase commences.

### 5.2.5. *Slice Substrate runtime management*

The Slice Resource Orchestrator holds a list of runtime constraints associated to the newly allocated Slice Substrate $G^S$, as it was initially specified by the Tenant inside the associated Slice Descriptor. These constraints are utilised to derive a set of performance requirements that have to be maintained during the whole lifetime of the slice. Through the Infrastructure Adaptor, the Slice Resource Orchestrator is able to collect the current status and performance of each resource slice. Hence, it can use this information to check what rules match the expected runtime requirements and trigger actions accordingly, based on the execution of vertical/horizontal slice scaling.

As an example, if one of the received runtime constraints was max *CPU*% usage of $n_c^S = 80\%$, and the measured value has been 100% for the past 60 s, then the Slice Resource Orchestrator may trigger vertical slice scaling on $n_c^S$, and request the allocation of additional compute resources to the DC Slice Controller where that slice was allocated. Similarly, when horizontal slice scaling is to be performed to deal with similar conditions, the Slice Resource Orchestrator will contact the Slice Builder, ask for additional compute, storage and/or network slices to be allocated in the available resource domains, and finally attach them to the existing Slice Substrate's topology.

It should be noted that the runtime management of the service, performed by the Service Orchestrator, may in turn impact the status of the hosting slice and lead to the execution of the above scaling operations. Additional details on this can be found in [19].

## 6. Experimental results

A qualitative assessment of the *three features* of the end-to-end slicing concept is now discussed. This is based on the system implementation and the workflow deployed onto the experimental testbed, described in the previous section. Three types of experiments were carried out, namely:

1. allocation of Slice Substrates of different size, in an on-demand softwarised fashion;
2. deployment of Service Substrates, with a variable number of Service Entities, on those Slice Substrates; and
3. evaluation of how the execution of those services impacts the slice resource utilisation.

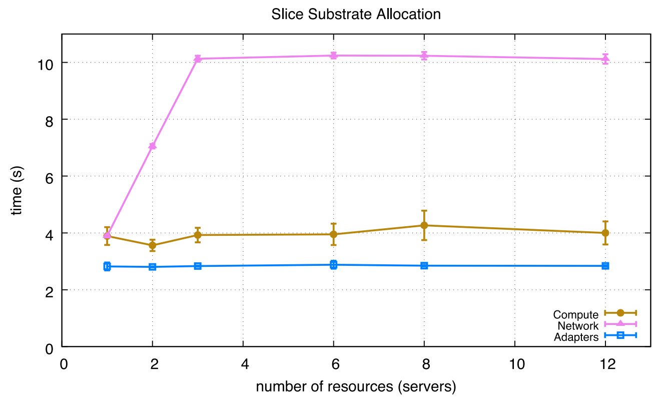

**Fig. 4.** Timings for the allocation of a Slice Substrate.

In addition to these three experiments, a fourth experiment was carried out in order to evaluate the execution of the slice mapping procedure. This specific aspect of the slice creation workflow was tested on a large-scale as a separate experiment.

For these experiments, it is assumed that resources of the cloud-to-edge continuum are sliced and orchestrated to deliver data-intensive processing services to the Tenants. More specifically, as in our previous work [55], massive streams of data are generated via software IoT devices and processed in different stages, according to some latency constraints. The first processing stage requires the lowest available latency, hence it is performed as close as possible to the data sources. Next stages, with less stringent real-time processing requirements but a growing demand for computational resources, are carried out in the available fog layers and, eventually, in the central cloud. Using our prototype implementation, end-to-end slices are setup to include (up to) three compute domains and two network domains. IoT data are then generated, and the processing elements are deployed as a Service Substrate on the created end-to-end slice. As previously introduced in Section 5, these service components are implemented as VLSP virtual Service Functions, each running as a separate Java container. They are also interconnected via virtual Service Links based on the UDP-like USR transport protocol [48].

### 6.1. *Experiment 1: Slice Substrate allocation*

This experiment was devised to validate the *first feature* of the slicing approach, i.e., the Tenant's ability to allocate slices across the cloud-to-edge continuum on-demand, in a software-enabled fashion. Fig. 4 shows the execution timings of the tasks required for the allocation of a Slice Substrate in our real testbed environment. As bare-metal compute slices are allocated during the experiment, the graph shows on the *x*-axis the total number of involved physical servers. The number of compute slices considered in this experiment was as follows: 1 compute slice when using a single server, 2 compute slices when using 2 servers, and 3 compute slice when using 3 or more servers. The y-points of the graph represent a mean value calculated over a set of 30 executions of the same experiment. Confidence intervals (95%) are also reported.

From the graph, it can be noticed that the allocation time for the computing resources (in brown) does not vary with the overall number of compute slices. As the current Slice Builder's implementation asynchronously contacts all the nominated domains, the allocation of the required resource slices can be performed in parallel. Furthermore, when three compute slices are considered, the resource allocation time does not increase with the number of resource elements within each slice. This result

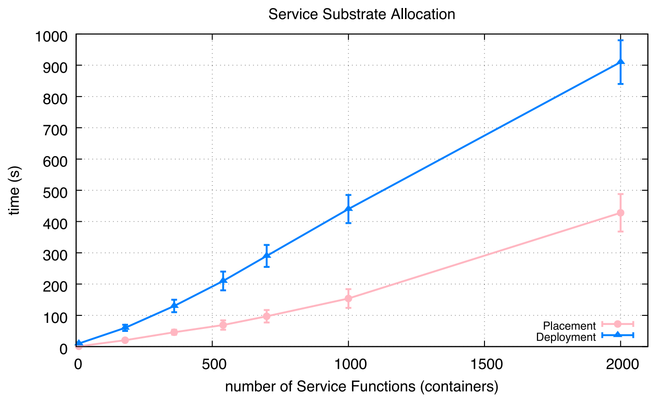

**Fig. 5.** Timings for the allocation of a Service Substrate.

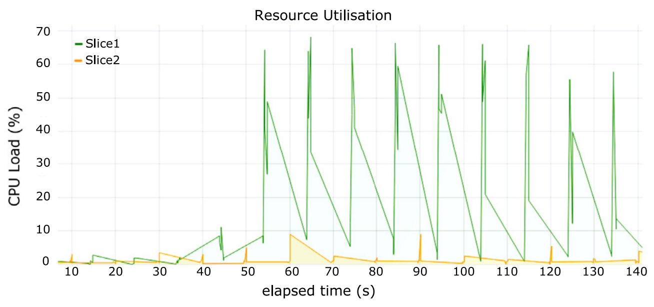

**Fig. 6.** CPU utilisation of two different end-to-end slices.

is linked to the way the DC Slice Controller allocates bare-metal compute slices and performs the on-demand instantiation of a new VLSP VIM instance [11]. A similar system behaviour is shown in the graph for the deployment of the VIMs/WIMs Infrastructure Adaptors (in blue) [56]. (This was only briefly mentioned in the previous section to simplify the workflow.)

On the other hand, the setup of the Slice Substrate's connectivity grows linearly with the number of involved network slices, as it is shown in Fig. 4 (in violet). This is due to the way the allocation of a new network slice is currently performed by this prototype, namely by sequentially and synchronously iterating through each of network slices of the $G^S$ graph. Future versions of the prototype will look at how these operations can be performed more efficiently and in a parallel fashion.

### 6.2. Experiment 2: Service Substrate allocation

This experiment was performed to assess the *second feature* of the proposed slicing solution, i.e., the Tenant's functionalities should work independently of how the underlying resources are managed, and whether they have been sliced or not. Hence, by using the same Service Orchestrator presented in [54], this experiment demonstrates that a Tenant is able to attach a newly created Slice Substrate to their existing systems, without the need of applying any changes to them, and can ultimately perform the deployment of specific services onto the allocated sliced resources in a transparent way.

A Slice Substrate formed of three bare-metal compute slices (with four servers each) and two network slices was first allocated, as discussed in the previous subsection. Then a Service Substrate, representing an IoT data processing service distributed across the cloud-to-edge continuum, was deployed on the allocated end-to-end slice. The size of the Service Substrate is characterised by the number $N_f$ of Service Functions, namely the service components that perform the data computation, and the number $N_l = N_f - 1$ of Service Links that interconnect them. The graph of Fig. 5 shows on the $x$-axis the values related to $N_f$, which for this experiment were selected as follows: $N_f \in \{10, 180, 360, 540, 700, 1000, 2000\}$.

Two datasets for $N_f$, related to the Service Substrate instantiation are plotted on the same figure. The service *placement* (in pink) is the time required for determining where the different Service Functions and Service Links have to be deployed on the Slice Substrate's topology. The service *deployment* (in blue) is the time spent by the Slice Resource Orchestrator to activate those Service Substrate's elements onto the resource slices, via interacting with the available VIMs and WIMs therein. The y-points of the graph represent a mean value calculated over a set of 30 executions of the same experiment. Confidence intervals (95%) are also reported. The time required for the execution of the service *placement* grows linearly with the number of Service Functions (and Service Links). The service *deployment* time also increases linearly. These results are specifically linked to this specific prototype implementation, the Service Orchestrator used, and the lightweight VLSP VIM and WIMs selected for the resource management of the slices.

### 6.3. Experiment 3: End-to-end slice resource isolation

The evaluation of the *third feature* of our slicing approach, namely the full resource isolation between different end-to-end slices, is the objective of this last experiment. Two Slice Substrates (*Slice1* and *Slice2*), formed of different interconnected bare-metal compute slices, were first instantiated as explained for the previous experiments. A Service Substrate, consisting of a distributed IoT data processing service similar to *Experiment 2*, was then submitted and placed onto *Slice1*, in order to generate load on the associated resources. For this particular experiment, the total number of Service Functions that performed various types of IoT data processing [55] was $N_f = 2000$; therefore the number of Service Links was $N_l = 1999$ and overall size of the Service Substrate was 3999. *Slice2* had a similar service deployment but the associated Service Functions were left in idle state throughout the duration of the experiment.

The graph of Fig. 6 shows the average CPU utilisation (percentage) of the bare-metal compute slices allocated for the above *Slice1* and *Slice2* (over a time period of 140 s). When the IoT data processing service was activated on *Slice1* at elapsed time $t = 55$ s, the related CPU load started to show a series of spikes (in green). Conversely, the average *Slice2*'s CPU utilisation (in orange) was not impacted by the load exerted by the service instance running on *Slice1*. Therefore, in general, end-to-end slices created according to our slicing model (with resource elements selected as per *Eq.* (13)) can be considered as fully independent bundles of resources. They are not affected by the security and isolation problems seen in traditional multi-tenant clouds where, e.g., highly loaded VMs can take over all of the available physical resources and impact the performance of other tenant's VMs.

### 6.4. Slice Substrate mapping experiment

The execution of the slice mapping procedure – required to select suitable resources prior to the Slice Substrate allocation – involved a small number of available Resource Providers. Due to the relatively small size of the real distributed testbed, the mapping was rather simple.

To enhance the results, the algorithm detailed in Section 5.2.2 was implemented and tested on a large-scale artificial environment using the tool [57], deployed on one of the machines of our testbed. Topologies of different size were generated for the $G^S$ and $G^R$ graphs, according to the BRITE-based Waxman methodology (with $\alpha = 0.5$ and $\beta = 0.5$) [58]. The domains' residual capacities, expressed by Eqs. (1)–(3), were real numbers uniformly distributed between 30 and 300, whereas the corresponding resource slice demands were generated in the range $10 - 100$. Resource domains have indeed a larger (aggregated) amount of available resources compared to individual resource slices' demands, even though some of the resources in their pools may have already been reserved to other previously allocated slices. Some management resource is temporarily needed for the processing of the mapping requests. As these requests may be independent, they can be processed in parallel if needed, which can scale horizontally across many servers [59].

Each experiment consisted of 30 individual runs based on the above settings, where the time required to perform the mapping procedure was measured considering a fixed size for the $G^R$ and $G^S$ graphs. In order to represent a large-scale cloud-to-edge continuum scenario, both 100 and 200 nodes were considered as the size of the $G^R$ graph [15]. On average, it took about 6 s to map a Slice Substrate graph of 10 nodes on a Resource Substrate graph of 100 nodes, and 90 s to map a Slice Substrate graph of 20 nodes on a Resource Substrate graph of 200 nodes. During these runs, the measured CPU utilisation for running the mapping algorithm was approximately 70% of one CPU core.

This is a reasonable overhead, considering the scenario's scale and the fact that the lifetime of these slices can be measured in either hours or days. Nonetheless, alternative algorithms can be implemented if different requirements become more important, e.g., in case more stringent performance is needed, as opposed to the objective of maximising the total number of successfully mapped Slice Substrates [49].

### 6.5. Discussion

The results of the experiments discussed in this section confirm the effectiveness of our slicing solution in enabling automated orchestration of resources of the cloud-to-edge continuum, according to a set of specified Tenant's requirements. Slices are created, via a softwarised approach, using federated (geographically) distributed resources — selected dynamically from many different domains through the proposed Marketplace. Service instances (i.e., Service Substrates) can be activated on the sliced resources by the Tenants transparently, through their existing software systems. In our testbed, the Slice Substrate creation process took (on average) up to 20 s, depending on the number of involved resource slices. The Service Substrate's activation also varied according to the number of involved elements and required up to 15 min. Finally, our experiments highlighted successful results in terms of performance and quality of experience: the workload associated to a specific slice was never affected by the resource utilisation of slices belonging to other Tenants.

## 7. Conclusions and future work

As today's highly distributed Internet services are reshaping the current cloud and network infrastructure, Fog Computing is emerging as an effective way to enable the execution of IoT services with demanding latency and compute requirements. The Fog is inherently distributed over different layers, which can span from the edge to the centralised cloud, creating a spectrum of compute, storage and network resources known as the cloud-to-edge continuum. The management of the above resources, to allow the execution of distributed services, is not trivial. Being highly dispersed by nature, this infrastructure can span multiple administrative domains. Moreover, several tenants can access the shared pool of resources and request the execution of services with diverse performance requirements. This calls for innovative techniques that can facilitate the orchestration of the resource layer and the co-existence of the different services requested by those tenants.

This paper answers the above *research question* by proposing a unified approach for the orchestration of the compute, storage and network resources of the cloud-to-edge continuum, based on the concept of *end-to-end slice*. It goes beyond the state-of-the-art on the areas of *fog orchestration* and *network slicing*, as it attempts to bring together some of the research conducted independently on those topics. The design and implementation of a system able to integrate the selection, configuration and management of compute, storage and network resources, as part of a single abstracted object exclusively assigned to a tenant to satisfy specific service requirements, is the *main contribution* of this work.

The slicing approach is grounded on *three features*, which have been extensively described and validated in this paper via a set of experiments performed on a real tested, with resources located in Europe and Brazil. The results show that resources for the slices can effectively be traded between providers using the *Resource Marketplace*. A graph-based model for those resources, and for the slices, allows dynamic resource discovery, selection, and mapping via different algorithms and optimisation goals; slices can be built on-demand and under software control (first feature). The usage of *bare-metal* compute slices, together with the *VIM/WIM on-demand*, allows to trade-off performance and resource manageability, while it also simplifies the inter-provider communication required after a slice has been created. The allocated resources are aggregated and exposed to a tenant as a *single object* that abstracts the distributed nature of the underlying infrastructure; minimal changes to the tenants' service orchestration tools are needed (second feature). The devised resource allocation model provides the tenants with fine-grained customisation capabilities to support the desired level of *isolation* and *performance*, and to prevent the likelihood of service performance degradation (third feature).

These encouraging results can pave the way towards the usage of our *end-to-end slice* concept in new emerging and demanding applications areas such as real-time IoT analytics, Digital Twins, and AI. As future work, we plan to perform further performance evaluation and testing in the above areas, using alternative VIM/WIMs implementations and additional geographically distributed resource domains that span the whole spectrum of Fog resources, up to resource-constrained end-user devices. This would allow the collection of additional results that could lead to further refinement of some of the concepts and mechanisms already described in this paper, as well as providing a more detailed view of the overheads of the management and orchestration systems.

## CRediT authorship contribution statement

**Francesco Tusa:** Problem conceptualization, Analytic model, Supported system design and implementation, Execution of experiments, Writing – review & editing. **Stuart Clayman:** System architecture, Conceptualization, System design and implementation, Including the VIM/WIM on-demand; Writing – review & editing.

## 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.

## Data availability

No data was used for the research described in the article.

## Acknowledgements

The authors would like to thank the colleagues that worked on the EU/Brazil project NECOS and contributed to the preliminary definition of some of the ideas described in this paper; especially Billy Pinheiro for assisting with both the setup of the testbed and the execution of the initial experiments. This work was partially supported by the DIGITbrain project, No. 952071, European Commission (EU H2020).

## References

[1] Statist, Number of IoT devices worldwide by 2025, 2022, URL https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/.

[2] S. Verma, Y. Kawamoto, Z.M. Fadlullah, H. Nishiyama, N. Kato, A survey on network methodologies for real-time analytics of massive IoT data and open research issues, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1457–1477.

[3] S. Trinks, C. Felden, Edge computing architecture to support real time analytic applications: A State-of-the-art within the application area of Smart Factory and Industry 4.0, in: 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2930–2939.

[4] P. Patel, M. Intizar Ali, A. Sheth, On using the intelligent edge for IoT analytics, IEEE Intell. Syst. 32 (5) (2017) 64–69.

[5] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana, The internet of things, fog and cloud continuum: Integration and challenges, Internet Things 3–4 (2018) 134–155, http://dx.doi.org/10.1016/j.iot.2018.09.005, URL https://www.sciencedirect.com/science/article/pii/S2542660518300635.

[6] B. Costa, J. Bachiega, L.R. de Carvalho, A.P.F. Araujo, Orchestration in fog computing: A comprehensive survey, ACM Comput. Surv. 55 (2) (2022).

[7] A. Galis, F. Tusa, S. Clayman, C.E. Rothenberg, J. Serrat, Slicing 5G networks: An architectural survey, in: Wiley 5G Ref, American Cancer Society, 2020, pp. 1–41.

[8] F. Hofer, M.A. Sehr, A. Iannopollo, I. Ugalde, A. Sangiovanni-Vincentelli, B. Russo, Industrial control via application containers: Migrating from bare-metal to iaas, in: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2019, pp. 62–69.

[9] F. Tusa, S. Clayman, D. Valocchi, A. Galis, Multi-domain orchestration for the deployment and management of services on a slice enabled NFVI, in: IEEE Mobility Support in Slice-Based Network Control for Heterogeneous Environments Co-Hosted with Conference on Network Function Virtualization and Software Defined Networks, Verona, 2018.

[10] NGMN Alliance, Description of network slicing concept, NGMN 5G P1 requirements & architecture, work stream end-to-end architecture, 2020, (Accessed on November 2020). URL https://www.ngmn.org/wp-content/uploads/Publications/2016/161010_NGMN_Network_Slicing_framework_v1.0.8.pdf.

[11] S. Clayman, F. Tusa, A. Galis, Extending slices into data centers: the VIM on-demand model, in: IEEE 9th International Conference on Network of the Future – NoF, Poznań, Poland., 2018.

[12] P. Rad, A.T. Chronopoulos, P. Lama, P. Madduri, C. Loader, Benchmarking bare metal cloud servers for HPC applications, in: 2015 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM, 2015, pp. 153–159.

[13] M. Gramaglia, I. Digon, V. Friderikos, D. Von Hugo, C. Mannweiler, M.A. Puente, K. Samdanis, B. Sayadi, Flexible connectivity and qoe/qos management for 5G networks: The 5G NORMA view, in: 2016 IEEE International Conference on Communications Workshops, ICC, IEEE, 2016, pp. 373–379.

[14] K. Cheng, S. Doddamani, T.-C. Chiueh, Y. Li, K. Gopalan, Directvisor: Virtualization for bare-metal cloud, in: Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 45–58.

[15] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, R. Riggio, Orchestrating end-to-end slices in 5G networks, in: 15th International Conference on Network and Service Management, CNSM, 2019, pp. 1–9.

[16] Y. Jiang, Z. Huang, D.H.K. Tsang, Challenges and solutions in fog computing orchestration, IEEE Netw. 32 (3) (2018) 122–129.

[17] M.S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keils, F. Schreiner, A service orchestration architecture for Fog-enabled infrastructures, in: 2017 Second International Conference on Fog and Mobile Edge Computing, FMEC, 2017, pp. 127–132.

[18] S. Clayman, F. Tusa, A. Galis, L. Contreras, WIM on-demand – A modular approach for managing network slices, in: IEEE Conference on Network Softwarization, Ghent, 2020.

[19] S. Clayman, A. Neto, F. Verdi, S. Correa, S. Sampaio, I. Sakelariou, L. Mamatas, R. Pasquini, K. Cardoso, F. Tusa, C. Rothenberg, J. Serrat, The NECOS approach to end-to-end cloud-network slicing as a service, IEEE Commun. Mag. 59 (3) (2021) 91–97.

[20] P. Pop, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, W. Steiner, The FORA fog computing platform for industrial IoT, Inf. Syst. 98 (2021) 101727, http://dx.doi.org/10.1016/j.is.2021.101727.

[21] M. Etemadi, M. Ghobaei-Arani, A. Shahidinejad, A cost-efficient auto-scaling mechanism for IoT applications in fog computing environment: A deep learning-based approach, Cluster Comput. 24 (4) (2021) 3277–3292, http://dx.doi.org/10.1007/s10586-021-03307-2.

[22] A. Ullah, H. Dagdeviren, R. Ariyattu, J. DesLauriers, T. Kiss, J. Bowden, MiCADO-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum, J. Grid Comput. 19 (2021) http://dx.doi.org/10.1007/s10723-021-09589-5.

[23] A.A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges, Comput. Netw. 167 (2020).

[24] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, H. Flinck, Network slicing and softwarization: A survey on principles, enabling technologies, and solutions, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2429–2453.

[25] X. Foukas, G. Patounas, A. Elmokashfi, M.K. Marina, Network slicing in 5G: Survey and challenges, IEEE Commun. Mag. 55 (5) (2017) 94–100.

[26] J. Kim, D. Kim, S. Choi, 3GPP SA2 architecture and functions for 5G mobile communication system, ICT Express 3 (2017).

[27] ETSI, ETSI zero-touch network and service management industry specification group (ZSM ISG), 2019, URL https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf.

[28] IETF, Network Slicing – Revised Problem Statement: draft-galis-netslices-revised-problem-statement-03, 2018.

[29] ITU-T, Y.3212: Framework for the support of network slicing in the IMT-2020 network, 2020, (Accessed on November 2020). URL https://www.itu.int/rec/T-REC-Y.3112-201812-I/en.

[30] FIWARE, FIWARE (Future Internet Ware), EU FP7-ICT large-scale integrating project (IP), 2014, URL http://www.fi-ware.eu/.

[31] F. Delli Priscoli, A. Di Giorgio, F. Lisi, S. Monaco, A. Pietrabissa, L.R. Celsi, V. Suraci, Multi-agent quality of experience control, Int. J. Control Autom. Syst. 15 (2) (2017) 892–904.

[32] M. Dryjanski, Towards 5G-research activities in Europe (HORIZON 2020 projects, 2017.

[33] J. Nightingale, Q. Wang, J.M. Alcaraz Calero, E. Chirivella-Perez, M. Ulbricht, J.A. Alonso-López, R. Preto, T. Batista, T. Teixeira, M.J. Barros, et al., QoE-driven, energy-aware video adaptation in 5G networks: The SELFNET self-optimisation use case, Int. J. Distrib. Sens. Netw. 12 (1) (2016) 7829305.

[34] H. Halabian, Optimal distributed resource allocation in 5G virtualized networks, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM, 2019, pp. 28–35.

[35] Amazon, Amazon EC2 C5 instances, 2022, URL https://aws.amazon.com/ec2/instance-types/c5/.

[36] Alibaba, Alibaba ECS bare metal instance, 2022, URL https://www.alibabacloud.com/product/ebm.

[37] IBM, IBM cloud bare metal servers, 2022, URL https://www.ibm.com/cloud/bare-metal-servers.

[38] M. Slack, Metal slack – Bring the cloud into your data centre, 2022, URL https://metal-stack.io.

[39] RackHD, RackHD – Hardware management and orchestration, 2022, URL https://rackhd.github.io.

[40] Openstack, Ironic – Openstack bare metal, 2022, URL https://www.openstack.org/use-cases/bare-metal/.

[41] ETSI, Network Functions Virtualisation, Network functions virtualisation (NFV), Manag. Orch. 1 (2014) V1.

[42] NECOS, EU-Brazil – Novel enablers for cloud slicing, 2017, http://www.h2020-necos.eu.

[43] A. Celesti, F. Tusa, M. Villari, A. Puliafito, How to enhance cloud architectures to enable cross-federation, in: 2010 IEEE 3rd International Conference on Cloud Computing, IEEE, 2010, pp. 337–345.

[44] A. Sgambelluri, O. Dugeon, A. Muhammad, J. Martín-Pérez, F. Ubaldi, K. Sevilla, O.G.D. Dios, T. Pepe, C.J. Bernardos, P. Monti, F. Paolucci, Orchestrating qos-based connectivity services in a multi-operator sandbox, J. Opt. Commun. Netw. 11 (2) (2019) A196–A208.

[45] P.D. Maciel, F.L. Verdi, P. Valsamas, et al., A marketplace-based approach to cloud network slice composition across multiple domains, in: 2nd Workshop on Advances in Slicing for Softwarized Infrastructures (S4SI), Co-Hosted at the 5th IEEE NetSoft, Paris, 2019.
[46] L. Contreras, S. Barguil, R. Vilalta, V. López, Architecture for integrating vertical customer's programmability control of network functions and connectivity in a slice-as-a-service schema, EURASIP J. Wireless Commun. Networking 2021 (2021) http://dx.doi.org/10.1186/s13638-021-01992-6.
[47] L. Contreras, J. Ordóñez, On slice isolation options in the transport network and associated feasibility indicators, 2021, pp. 201–205, http://dx.doi.org/10.1109/NetSoft51509.2021.9492546.
[48] S. Clayman, L. Mamatas, A. Galis, Experimenting with control operations in software defined infrastructures, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 390–396.
[49] H. Cao, S. Wu, Y. Hu, Y. Liu, L. Yang, A survey of embedding algorithm for virtual network embedding, China Commun. 16 (12) (2019) 1–33, http://dx.doi.org/10.23919/JCC.2019.12.001.
[50] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, SIGCOMM Comput. Commun. Rev. 41 (2) (2011) 38–47.
[51] H. Cao, L. Yang, H. Zhu, Novel node-ranking approach and multiple topology attributes-based embedding algorithm for single-domain virtual network embedding, IEEE Internet Things J. 5 (1) (2018) 108–120.
[52] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web., Technical Report 1999–66, Stanford InfoLab, 1999, Previous number=SIDL-WP-1999-0120. URL http://ilpubs.stanford.edu:8090/422/.
[53] D. Eppstein, Finding the k shortest paths, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 154–165.
[54] A. Sgambelluri, F. Tusa, M. Gharbaoui, E. Maini, L. Toka, et al., Orchestration of network services across multiple operators: The 5G exchange prototype, in: 2017 European Conference on Networks and Communications (EuCNC), 2017, pp. 1–5.
[55] F. Tusa, S. Clayman, The impact of encoding and transport for massive real-time IoT data on edge resource consumption, J. Grid Comput. 19 (3) (2021) 32.
[56] A. Beltrami, P.D. Maciel, F. Tusa, C. Cesila, C. Rothenberg, R. Pasquini, F.L. Verdi, Design and implementation of an elastic monitoring architecture for cloud network slices, in: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1–7.
[57] Alevin2: tool for the evaluation of ALgorithms for Embedding VIrtual Networks. URL https://sourceforge.net/p/alevin/wiki/home/.
[58] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: an approach to universal topology generation, in: MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001, pp. 346–353.
[59] M.T. Beck, A. Fischer, H. de Meer, J.F. Botero, X. Hesselbach, A distributed, parallel, and generic virtual network embedding framework, in: 2013 IEEE International Conference on Communications, ICC, 2013, pp. 3471–3475, http://dx.doi.org/10.1109/ICC.2013.6655087.

**Francesco Tusa** is an Assistant Professor at University of Westminster and was a Research Associate at University College London. His research interests are in the area of large-scale federated distributed systems, Cloud and Fog computing, Information Security, and the Internet of Things. He is co-editor of a book and coauthored over 40 conference and journal papers. He contributed to several EU funded research projects in the area of Cloud and Networks, such as NECOS, 5GEx, SONATA, VISION Cloud and RESERVOIR. He is involved in the organisation of various conferences and serves as reviewer for prestigious journal and magazines in the area of cloud and networks. Francesco has technical experience in designing and developing large distributed software systems as well as in managing large testbeds, clusters and distributed computing infrastructures. He is currently investigating resource management in the cloud-to-edge continuum and he is looking at the usage of homomorphic encryption in the Cloud and Network domains.

**Stuart Clayman** received his Ph.D. in Computer Science from University College London in 1994. He is currently a Principal Research Fellow at UCL EEE department, and he has worked as a Research Lecturer at Kingston University and at UCL. He co-authored over 70 conference and journal papers. His research interests and expertise lie in the areas of software engineering and programming paradigms; distributed systems; virtualised compute and network systems, network and systems management; sensor systems and smart city platforms, and artificial intelligence systems. He is looking at enhanced mechanisms for end-to-end delivery of digital video, as well as new techniques for large-scale sensor systems in Industry 4.0. He also has extensive experience in the commercial arena undertaking architecture and development for software engineering, distributed systems and networking systems. He has run his own technology start-up in the area of NoSQL databases, sensor data, and digital media.