ACHO: A Framework for Flexible Re-Orchestration of Virtual Network Functions

Gines Garcia-Aviles[1], Carlos Donato[2], Marco Gramaglia[1], Pablo Serrano[1], Albert Banchs[1,3]

Abstract

Network Function Virtualization enables network slicing as a novel paradigm for service provisioning. With network slicing, Virtual Network Functions (VNFs) can be instantiated at different locations of the infrastructure, choosing their optimal placement based on parameters such as the requirements of the service or the resources available. One limitation of state-of-the-art technology for network slicing is the inability to re-evaluate orchestration decisions once the slice has been deployed, in case of changing service demands or network conditions.

In this paper, we present ACHO, a novel software framework that enables seamless re-orchestration of VNFs of any kind, including RAN and Core. With ACHO, VNFs and resources can be easily re-assigned to match, e.g., varying user demands or changes in the nodes' load. ACHO uses lightweight mechanisms, such as splitting the engine of a VNF from the data it requires to perform its operation, in such a way that, when re-allocating a VNF, only the data is moved (a new engine is instantiated in the new location). We demonstrate the use of ACHO in a small scale testbed, showing that (*i*) the proposed re-orchestration is feasible, (*ii*) it results much faster than existing alternatives (especially for relocation), and (*iii*) the framework can be readily applied to existing VNFs after minimal changes to their implementation.

Keywords: 5G mobile communication, Computer network management, Network architecture, Network function virtualization

[1]University Carlos III of Madrid
[2]IMEC
[3]IMDEA Networks Institute

1. Introduction

One key technology of 5G Networking is *network slicing* [1], which allows the use of the same infrastructure to support very diverse services. Enabled by the irruption of the Network Function Virtualization (NFV) paradigm [2], network slicing breaks the traditional "one size fits all" network paradigm, by permitting the deployment of multiple VNFs in different general-purpose clouds. This coordination between the hardware elements of a given deployment and the software running on top of them is usually referred to as *network orchestration*, which enables tailoring each virtual network deployment to a particular service.

However, current orchestration solutions are not flexible, in the sense that this mapping between resources (e.g., hardware, radio spectrum) and software is decided once per service and it is hard to modify afterward. This reduced flexibility results in the following issues, which could be addressed by a more flexible orchestration:

Lack of re-location While the current state of the art solutions allow for basic scaling or migration of a VNF, they are usually limited to replica creation within the same datacenter on the same hardware platform. This falls short when the target is to flexibly adapt to the envisioned dynamic demand in a cost-efficient way, as the technology needs to support seamless re-location and re-configuration of VNFs [3] across datacenters, without any assumption on the underlying hardware. This approach naturally couples with hierarchical and network slicing native architectures that have been recently proposed for next generation networks [4], and received very low attention from the research community, with the exception of [5].

Lack of fine re-configuration The transition towards a full *cloud native* suite of network functions is still ongoing. While the traditional functions only exposed very few configuration parameters such as power management or frequency control [6], with network softwarization the variables that may be controlled from the management perspective can be much more, allowing a fine grained control of aspects such as the sharing of spectrum across different slices or tenants, or the configuration of radio resource blocks. Despite this possibility, and especially in the access network, a cloud-oriented control of network functions is lacking. In fact, only recently and for the Core Network, the Service Based Architecture [7] has been proposed, while for the RAN part some similar efforts have been proposed [8], but almost no implementation is available (the one in [9] is not provided as open source). In this paper, we propose ACHO a novel open source framework for *flexible* orchestration of network functions, which (*i*) provides the ability to relocate VNFs at run-time, and (*ii*) supports their fine-grained re-configuration.

The main cornerstone of the ACHO design is the adjacency to the relevant standard solutions, mainly 3GPP and ETSI NFV. This further demonstrates the applicability of the ACHO's concepts and vision on top of the relevant state of the art technology.

Thus, the contribution of this paper are summarized as follows:

- The design of a flexible re-orchestration framework that allows enhanced operations such as VNF re-location and fine re-configuration, including the definition of the required interfaces that support these operations.
- A library of VNFs adapted to this framework, including the basic set of features to have an operational 5G network.
- A proof-of-concept evaluation of the overall ACHO solution.

By open-sourcing ACHO, we aim to foster 5G experimenting repeatability, to improve code reliability, and to enable other researchers to extend the number of supported scenarios beyond those studied in this paper. The codebase, available on GitHub[4] under the AGPLv3 license, is the first open-source solution of basic 5G Core functionality with seamless re-location capabilities (to the best of our knowledge).

The rest of the paper is structured as follows: in Section 2 we describe the advantages of flexible network orchestration and the challenges to achieve it, discussing also the state of the art solutions. Then, Section 3 describes our solution, while Section 4 provides quantitative performance figures in terms of re-orchestration delay and achieved isolation across slices. Finally, concluding remarks are provided in Section 5.

2. Flexible network orchestration: advantages and state of the art

Network orchestration [10, 11] can be defined as the coordination between the hardware elements of a given deployment and the software modules running on top of them. Current orchestration solutions only support a static

[4] `https://github.com/wnlUC3M/`

and coarse-grained operation: once instantiated, it is hard to modify the resources associated to a specific network slice (e.g., re-locate a VNF to the edge), or to support a fine-grained re-configuration (e.g., scale-up just the flows belonging to a specific network slice). We define a flexible network orchestration as the one supporting a dynamic and fine grained operation. These characteristics would enable the so-called elastic orchestration of network slices [12] which, in turn, would improve the resource utilization in the network.

In the following, we first make the case for these two features, and then discuss the state of the art technology and the current implementation landscape.

2.1. The case for flexible re-orchestration of VNFs

As defined above, a re-orchestration solution is flexible only if it is both dynamic and fine-grained, features which are currently unavailable with existing orchestration solutions (we discuss these solutions in Section 2.2). In the following, we discuss the main advantages of these two features.

2.1.1. Advantages of a dynamic re-orchestration

Some of these advantages obtained with a dynamic re-orchestration are:
Adapting to user mobility. Low-latency services such as tactile or vehicular communications require that VNFs affecting latency are as close as possible to the user, to minimize the delay between these functions and the user. When a user moves to a new location, VNFs should move as well to keep close to the user's new location. This requires the ability to relocate those functions without disrupting the ongoing service.
Service enhancements in run-time. Flexible re-location also gives the ability to re-compose a service provided by a given slice, to add, substitute, remove, or relocate VNFs in the chain. This enables introducing a variety of features during run-time operation, such as, e.g., adding or relocating a firewall, or replacing a more efficient (but slower) video encoder by a quicker but less efficient one to adapt to changes in the measured delay while keeping quality of experience.
Improved de/scaling. Resource scaling refers to the ability to assign resources as needed. While the *traditional* vertical and horizontal scaling could provide this feature to some extent, the use of relocatable VNFs introduces an additional level of flexibility without disrupting the service: when a VNF runs out of resources, it can be relocated to a different location with more

resources. When few functions are running in different locations, this allows to relocate them in a single resource and deactivate the unused nodes, saving resources by implementing infrastructure on-demand schemes [13].

Resilient operation. The ability to relocate VNFs in real-time enables novel methods to provide resiliency. In case of service disruption due to, e.g., the congestion of a node, or a hardware failure, it would be possible to relocate the required functions seamlessly trigger their "activation", thus providing resilience against impairments of various kinds.

2.1.2. Advantages of a fine-grained re-orchestration

The transition to a more modular and software-based architecture such as the one in the 3GPP Release 15 [14] opens the door for a more precise resource management. Also, the API-based control of the core network function (the so-called SBA architecture) allows for an easier way of re-configuring functions, following the slice needs. Among the features that such fine-grained re-orchestration capabilities would enable, we have:

Per-slice re-configuration. Network slices (or Sub Network Slices [15]) are the "least common multiple" when it comes to service management. As VNFs can be shared among slices [15], they should expose APIs that enable the per-slice configuration. Besides the per-slice parameter re-configuration, the orchestration framework shall also support operations such as join and split, i.e., grouping into the same virtual instance (a Virtual Machine or a container) a group of VNFs belonging to different slices, and vice-versa.

Joint parameters and resource configuration. Modifying a parameter of a given VNFs may have an impact on its resource footprint, and also on the one from other VNFs, both from the network resources perspective (i.e., more or different frequency bands) and the computational (i.e., more CPU). An orchestration algorithm shall be able to assess the impact of a change of parameters on the underlying infrastructure and act accordingly.

Access network re-configuration. While the core network functions already have incorporated softwarization principles since the standardization, access network functions are more "grounded" in a less flexible architecture, which is partly also due to their need to comply with stringent timing requirements. Just very recently, industrial fora such as Open RAN [8] started to advocate for a finer programmable management of the radio access. Such concepts shall be incorporated in the orchestration framework.

2.2. State of the art

Despite the advantages discussed above, the technology currently available does not support a flexible re-orchestration of VNFs. In the following, we revise the state of the art, highlighting the most relevant initiatives and contributions.

2.2.1. General VNF placement and orchestration problems

There is a bulk of literature available on the problems of VNF placement and orchestration, summarized by a number of surveys, e.g., [16, 17, 18]. In general, the different proposals can be classified depending on various axes: (*i*) the variables to be optimized, e.g., power, cost, latency; (*ii*) if the optimization is mono- or multi-objective; and (*iii*) whether the matching of physical and virtual resources is carried out in an offline manner (gathering inputs, requirements, etc.) or in an online manner, following, e.g., the crossing of a threshold, or a periodic trigger. It should be noted, though, that even if the approaches falling into this latter category are referred to as "dynamic" in [17], these solutions are not tested in scenarios considering quick variations over time (see e.g. [19]).

In fact, despite this remarkable amount of previous work, actually few proposals deal with the implementation of such algorithms on real VIMs, with the use of real-life traces being among the most common approaches for the performance evaluation. Furthermore, for those proposals performing a real-life evaluation, they typically rely on existing orchestration technologies that, as discussed in the next section, lack both the dynamism and granularity required for what we refer to as a flexible re-orchestation (i.e., dynamic and fine-grained).

2.2.2. Genaral-purpose orchestration technologies

Existing NFV Management and Orchestration (MANO) software solutions such as, e.g., Open Source MANO (OSM) [20] or the Open Networking Automation Platform (ONAP) [21], are continuously evolving solutions used in many fields to manage the VNF lifecycle (design, configuration, termination, etc.). Their interactions with the underlying infrastructure (to instantiate, connect, and terminate virtual resources) are done through a Virtual Infrastructure Managers (VIM), a software element that abstracts the complexity of the cloud. To enable the discussed advantages of a flexible orchestration of network slices, this VIM has to support (*i*) flexible relocation of virtual resources, and (*ii*) their fine-grained re-configuration.

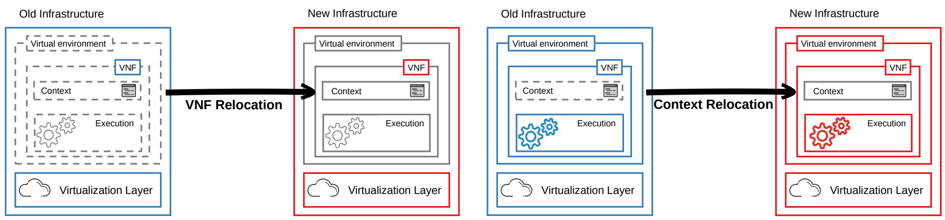
On the one hand, a fine-grained orchestration is tough: state-of-the-art orchestration platforms only allow to re-configure very basic parameters such as the IP address of the VNF, while other fine-grained parameters (such as the ones described in the Information Model [22] are left to the implementation of each VNF.

On the other hand, VNF re-location technologies are also lacking in terms of dynamism. Although existing VIMs can relocate a Virtual Machine (VM) from one compute node to another, this operation has notable limitations:

- They especially target limited parts of a VM such as its memory. These techniques use an iterative process [23] that starts from the memory pages that were the least frequently accessed, keep updating them until the ones that are the most used are moved. While relocating memory, usually a significant part of the VM has to be kept in a fixed location (e.g., a NAS hosting the disks). As a result,

- The relocation of a VM is limited within the boundaries of a single datacenter of a single VIM. These limitations are acceptable for cloud computing environments, which typically focus on very high reliability and therefore VMs are only relocated in case of, e.g., disk failures or programmed maintenance, but are inadequate for dynamic scenarios such as the use cases discussed above, involving the movement of VNFs across the network to reduce latency or to improve efficiency across datacenters. As a matter of fact, the topic of migration over WAN links (which is a relevant scenario for networking purposes, e.g., migration to edge cloud) is currently overlooked by the bulk of available literature, as also confirmed by the authors of [24]. Among the more than 200 works reviewed there, just a handful deal with migration over long distances and none of them provide experimental results.

As discussed, state of the art orchestration platforms provide some methods to relocate VMs. For instance, OpenStack provides live-migration tools [25]; however, their use precludes fast VNF re-location. Their operation is sketched in Fig. 1a: in contrast to ACHO, the full Virtual Machine has to be copied to the targeted destination. This includes common data such as the guest kernel, libraries, and the file system structure [23]: these elements are not copied when employing ACHO's context migration. Also, as these techniques are very expensive in terms of exchanged data, they are only available



(a) VM relocation (b) Context relocation

Figure 1: Relocation strategies: a full VNF relocation (left) vs. the context relocation performed with ACHO (right).

between the same NFV infrastructure point of presence (i.e., the infrastructure controlled by the same VIM instance), excluding thus the migration among VIMs, which ACHO supports.

Also, commercial products such as `VMWare` implement sophisticated techniques that can perform live migrations in very short times, by incrementally copying the memory of running virtual instances. However, these methods require very high bandwidth and a very short latency between the endpoints, as well as a shared disk image. The technical report from `VMWare` [26] declares migration times in the order of tenths of seconds over a 10 Gbps Ethernet connection. All these requirements preclude their use in our target scenario, which should support re-locations between endpoints relatively "far away" (e.g., different datacenters). Similar techniques are also employed in the context of containers, e.g., Voyager [27]. Besides being substantially lighter than a VM, this technique however still has to copy all the memory and the disk used by the container, making it unsuitable for far re-locations.

Similar considerations apply for other virtualization platforms that are particularly optimized for the VNF migration. For instance, `unikernels` can perform live migration within few milliseconds [28], but they are currently not part of any large scale NFV infrastructure deployment and therefore they are not integrated into commonly used MANO platforms such as ONAP or OSM. Because of this, they lack the required infrastructure management capabilities, and therefore they are unsuited accommodate basic features such as i.e., re-orchestration triggers in a seamless way.

2.2.3. Ad hoc solutions

Enabling flexible re-orchestration in softwarized network deployment received attention by the research community in the last few years. The work most closely related to ours is `SENATUS` [5], a framework that internally leverages on state of the art VIMs; because of this, this framework yields to very poor performance, in particular in challenging (i.e., very dynamic) scenarios, as we quantify in Section 4.

The idea of splitting the context of a function from its execution engine has also been proposed by some works in the literature, most notably OpenNF [29] and Split Merge [30]. These papers propose the fast relocation of VNF by moving the least amount of information between different virtualization environments. While these cases are relatively similar to ours, the solutions lack two key features that preclude their use in mobile networks: (*i*) the considered VNFs do not constitute part of the "3GPP ecosystem," and (*ii*) they lack an interface with a modern orchestrator, which is required to enable must-have features of mobile networks e.g., network slicing.

A similar idea is also currently included in the 3GPP specification [31], to relocate the information related to a specific UE between different instances of a network function, in particular for load balancing purposes or to keep providing connectivity when a specific function is decommissioned. Still, this procedure is available for the core functions only.

Finally, if we consider more targeted solutions that also specifically include the access network, the available material is even less. The most remarkable solution is Orion [9], which allows for a per-slice re-configuration of radio resources but the software is not freely available.

2.3. Main ACHO novelties

Based on the above analysis of the state of the art, we conclude that there is no practical open source solution for flexible orchestration of VNFs in a mobile network architecture. This motivated the design and implementation of ACHO, a framework that provides the following novelties as compared vs. the state of the art:

- Firstly, ACHO targets mobile networking, a more heterogeneous scenario with very diverse network functions with very different requirements (e.g., access network vs. core network functions).

- ACHO provides a clear methodology to adapt existing VNFs, which follows the recent architectural trends of 5G networking, and is aligned with the ongoing standardization efforts.

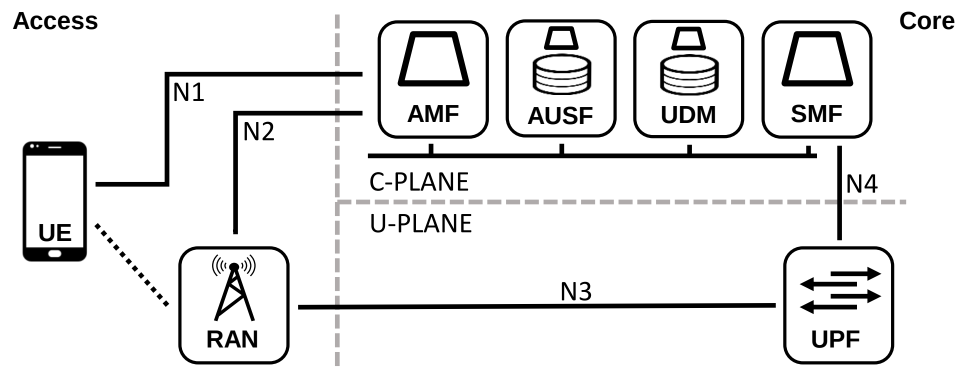


Figure 2: The selected 5G Core functions implemented for the tests.

- ACHO specifies two novel interfaces to support and dynamic and fine-grained orchestration, which can be easily implemented with existing off-the-shelf orchestrators.

- ACHO also provides a fully-featured implementation of 5G VNFs and orchestration elements, which can be easily downloaded, customized, and tested with off-the-shelf hardware.

- Finally, we discuss different implementation strategies, also aligned with existing standardization efforts, to maximize the practicality of ACHO.

3. ACHO: A suite for flexible 5G networking

We next present ACHO (Adaptive slice re-Configuration using Hierarchical Orchestration), a software framework consisting of an implementation of 5G Functionality (the most critical 3GPP Rel. 15 Core Network Functions, depicted in Fig. 2 and marked with SBA in Fig. 5), the radio access network functions and the MANO modules to handle them. ACHO provides a full network-slicing aware solution that includes all the MANO modules to enable a flexible re-orchestration of the mobile network. Some of the network components of ACHO have been adapted from existing open source projects (e.g., srsLTE), while other components that were not available as open source have been implemented from scratch. As a result, the software codebase provided by ACHO is very complete and has no match in the landscape of the open source mobile networking initiatives.

3.1. Efficient re-configuration of VNFs through context migration

As discussed in Section 2.2, a plethora of orchestration algorithms rely on dynamically migrating a VNF *on the fly*. However, very few of them deal with the actual implementation of the migration mechanism, with the work of [5] being among the notable exceptions, but providing very poor performance. This motivates the design of the ACHO framework. The key enabler of ACHO is a clean split between the *context* of a network function and its *execution* engine, which we refer to as the c/e split. The context is defined by the current values of all the variables employed by the function, while the engine is the part responsible for the actual execution of the function. In this way, when relocating a network function, it is sufficient to move to the new location just the context, which contains the "state" of the function. Therefore, we can instantiate a new function engine in the new location and feed it with the data corresponding to the context extracted from the previous location (this strategy is depicted in Fig. 1b).

By moving the context of a VNF only, we reduce the amount of information that has to be moved to the bare minimum, without incurring into large penalties as done by VNF unaware solutions [23]. For instance, the tests performed in [32] show how the total amount of data transferred is almost a linear function of the VM size. In the following, we discuss in details how such VNF migration can take place.

3.2. The VNF context

As discussed above, by introducing the c/e split, ACHO trades flexibility (i.e., the orchestration framework needs to know what kind of VNFs are running), with the compactness of the exchanged data, which is the bare minimum data representing the internal state of a VNF. The context is specific to each VNF but it is independent of the execution environment: for instance, we implemented ACHO for a VM-based deployment, but it can work with containers or unikernels. A context may comprise the specific rules of a firewall, or the information of the authenticated UE for an Authentication Server Function (AUSF). To support this, network functions need to be re-implemented to enable a clear separation between the context and the engine, a re-implementation that is specific to each function. Furthermore, these re-implemented VNF have to expose this new capability, which could be achieved by e.g. extending the Network Exposure Function (NEF) to support c/e split through an API.

An example of the context extracted from the SMF Network Function is depicted in Fig. 3. Given that the SMF is in charge of handling the end user session, routing them from the base station to the UPF, the context of this function includes all the required information to re-install the relocated flow into the new VNF. Fig. 3 provides a JSON representation of the context, but binary formats such as e.g. Google PBF could also be used. The context does not contain information about the resources utilized in the underlying infrastructure (i.e., number of CPUs, amount of RAM) that are left to the MANO framework by using the standard technologies (e.g., the VNFD file descriptors). A full description of the implementation of such relocatable functions is provided in Section 3.5.

Thus, according to operator-defined re-orchestration triggers (which can be computed from QoS metrics), the MANO pulls the context from the source VNF and injects it in the destination VNF (details on the specific interfaces are provided in Section3.4). Hence, in the SMF case discussed here, all the information related to the gNB and Gateway, including the tunnel id for each user is moved to the new location. Hence, the target VNF can immediately start serving the UE traffic from the new location.

Analogously, we depict in Fig. 4 the context used in our implementation of the MAC scheduler, which can perform re-orchestration based on per-slice. As discussed in Section 4, we use it to enforce isolation across UEs belonging to different slices, changing the RB allocation according to the number of served slices. This allows for a very granular per-slice (hence partial) re-orchestration, as all the parameters handled by ACHO are related to specific slice instances, as we also show with our proof of concept results in Section 4.

Hence, in addition to the re-implementation of these functions, we also need the means to transfer of the context from one location to another, i.e., instantiate the engine in the new location, feed it with the context, and update the corresponding communication paths. ACHO provides an open-source and practical implementation of this functionality, demonstrating that it is indeed feasible to relocate VNFs without disrupting ongoing services.

3.3. Baseline 5G implementation

To illustrate the benefits of the c/e split we need a working baseline implementation of certain 5G functionality, neither supported by current

```
{
  "enb_tun_ip_addr": "192.168.10.12",
  "gw_tun_ip_addr": "192.168.10.10",
  "enb_tun_hw_addr": "xx:xx:xx:xx:xx:xx",
  "gw_tun_hw_addr": "yy:yy:yy:yy:yy:yy",
  "external_src_mac": "zz:zz:zz:zz:zz:zz",
  "external_dst_mac": "cc:cc:cc:cc:cc:cc",
  "ue_ip_addr": "172.16.0.30",
  "teid": "1111111"
}
```

Figure 3: Representation of the SMF context

```
{
  "nsl_id": "1",
  "rnti": ["1","2","3"],
  "start_rb_id": 1,
  "stop_rb_id": 20,
}
```

Figure 4: Representation of the SMF context

versions of 3GPP (i.e., 4G/LTE) nor existing open-source implementations. We next describe the baseline architecture that we have developed, which is far more complete than any existing open source software alternative and includes: (*i*) a multi-slice capable access network (both in the UE and eNB), to support end-to-end network slicing, and (*ii*) a *modular* implementation of the Core Network, as mandated by recent 3GPP standards.

Radio Access Network Our Radio Access Network (RAN) implementation is based on the open source software suite srsLTE [33], which is extended to

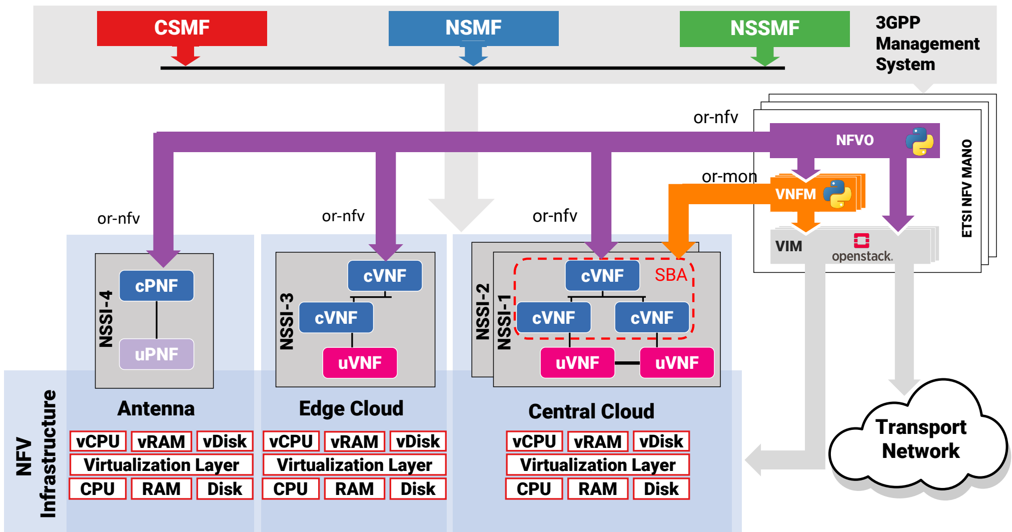


Figure 5: MANO Implementation and new Interfaces. ACHO creates new interfaces in the reference points defined by ETSI and acts on the underlying virtual or physical NFs (both c-plane and u-plane) to provide a fast re-location.

support multiple slices. This *Multi-slice RAN* builds on a modified version of srsLTE to support multiple slices on the same radio VNFs (the full implementation details of the baseline are available in [34]), that we have extended to support fine-grained reconfiguration of radio resources (see Section 3.5).

Core Network ACHO employs an ad-hoc version of the Core Network (CN) functionality that has been specifically implemented for this purpose, as VNFs shall implement the c/e split paradigm. Moreover, our implementation is fully modular and follows the service-based architecture (SBA). More specifically, the implementation of the Access Management Function (AMF), AUSF, User Data Management (UDM) and User Plane Function (UPF) functions is done in `Python 3`, and is detailed in Section 3.5.

3.4. New MANO functionality

The adoption of the c/e split requires novel MANO functionality, to enable the relocation of network functions, and new interfaces between the management and orchestration layers and the VNFs, to extract and install the contexts.

Hierarchical management and orchestration. To implement the relocation of VNFs within a running slice, we design a system that supports this functionality, following the recent efforts from the 3GPP [15] and ETSI [35]. Our design is illustrated in Fig. 5 and follows a hierarchical structure, with the following two main components:

(*i*) A 3GPP management system (top of the figure, as defined by [15]), which provides the entry point towards the business layers (i.e., the tenants that request a specific communication service) and manages services in the underlying network. We implemented these parts as `Python` modules, which includes the mapping of two communication services (namely, eMBB and mMTC) into two Network Function chains. In ACHO, we implement a reduced subset of the ones already defined by 3GPP. Namely, we logically select the VNFs that belong to each slice (including the sharing policies) and create their logic topology.

(*ii*) An ETSI NFV MANO system (top right) in charge of the central part of the network lifecycle management (i.e., instantiation, runtime, and termination). To implement this part, we have developed a composite implementation of the ETSI NFV MANO [35] stack. Specifically, we employ a base-line OpenStack as the VIM, and then developed the other modules (i.e., VNFM and NFVO) as ad-hoc modules, in `Python`. Basically, we leverage OpenStack to trigger the instantiation of different VMs in our infrastructure, by using its API. Also, the interfaces towards the VNFs are implemented using `Python`.

New interfaces. We designed two new interfaces: one to extract and install the context, and another one to estimate network conditions, which is needed to support decisions about VNFs re-locations. We denote these interfaces as `or-nfv` and `or-mon`, respectively (see Fig. 5). These interfaces can be considered as part of the already defined ETSI MANO reference points `or-vfnm`, `ve-vnfm-vnf` and `or-vi`, although other extensions may be considered. They are described next:

(*i*) `or-nfv`: This interface is used to extract and push the context of the VNFs. This interface is used by the Orchestrator (the NFVO), which is in charge of all the operational logic of a Network Slice. In particular, when deciding to relocate a function, the NFVO first extracts the context of the network function and then re-orchestrates this function, by pushing the context into the function available at the new location. This interface is similar to the one already included in the 5G system between the management service and the core network functions. This interface [36], connects the capabilities provided by the Network Exposure Function (NEF) and Network Repository Function (NRF) to extract and set configuration parameters from the network functions. In our implementation, following the current trends in network softwarization, this interface is implemented through a REST API.

(*ii*) `or-mon`: This interface connects the VNF manager (VNFM) with the VNFs through the SBA, and serves to monitor the VNFs, to trigger a relocation when performance falls below a given target (although the VIM has some monitoring capabilities, they typically circumscribe to the Virtual Machines and not the VNFs). This interface is also similar to the one defined by 3GPP between the Network Data Analytics Function (NWDAF) available in the core and the management system. However, in our implementation (based on a REST API), we extend its focus by targeting different metrics (e.g. latency, in addition to load) and also including access functions.

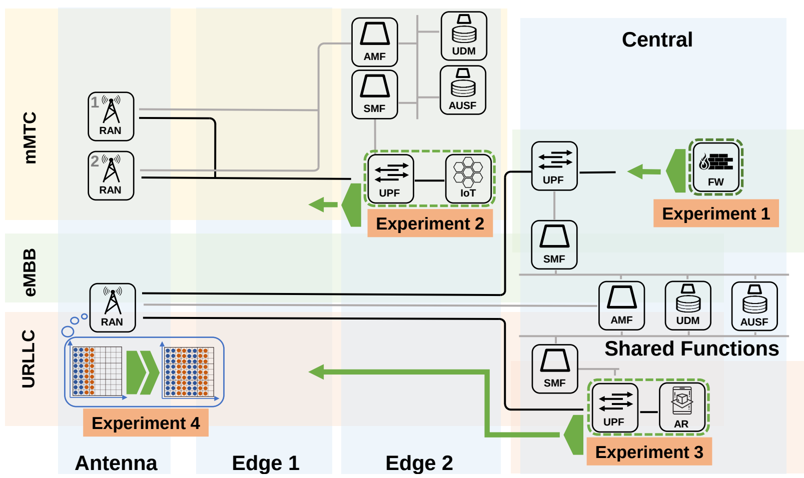


Figure 6: The Network Slice setup employed in the experimental evaluation, consisting of 3 slices.

3.5. Re-orchestrable VNFs

The proposed c/e split can be applied to any VNF, provided it implements the interfaces described above to extract and install the context. To show this, we have implemented different VNFs following the c/e split and thus making them "re-orchestrable." We note that the c/e split nicely fits with the SDN approach, which is an easy way to extract and inject the context from and to a VNF (i.e., in traditional SDN, the context of a switch are its forwarding rules). Thus, to implement the VNFs, for simplicity we have selected the Open Source `Lagopus` switch[5] as basis for our implementation

[5] `http://www.lagopus.org`

(alternatives such as ONOS [37] may be used for larger deployments). The diversity of the chosen functions shows the generality of our approach and the ability to apply it to any network function:

UPF: This function provides the encapsulation, decapsulation, and forwarding to the Packet Data Network. The implementation of this module follows the c/e split and includes the corresponding interfaces with our MANO system to extract and install the context. The context consists of the current rules applied to encapsulate/decapsulate packets and to forward them. We have implemented the UPF module building on the `Lagopus` switch.

SMF: This c-plane function controls and configures the UPF instances on the u-plane through the N4 [14] interface. Thus, the context here also consists of the rules to encapsulate/decapsulate/forward packets, in this case for all the UPF functions controlled by the SMF. For the implementation of this module, we leverage available SDN-capable implementations, enriching them with mobile network functionality, and employing a `Ryu` Controller[6] to implement the N4 interface between the UPF and the SMF.

IoT broker: The IoT broker acts as middleware between the sensors connected to a mobile network and a data sink that may be located in a central location. We have implemented this module in `Python` from scratch, including specific libraries for the handling of traffic flows from the sensors. Our lightweight and flexible implementation allows to dynamically transfer the broker context to a new location, which is particularly suitable for Mobile Edge Computing (MEC) deployments, as it allows moving the broker functionality across different edge infrastructures.

Firewall: This network function forwards IP packets from an ingress to an egress port following a set of firewall rules. The context of this network function thus consists of these rules. We have implemented the u-plane part of this function as a `Lagopus` switch, and the c-plane part as an extended `Ryu` controller. The latter gathers the rules, which are stored as `Python` objects, and provides them to the MANO system through the corresponding primitives.

MAC scheduler: One of the main functions of MAC layer in LTE is the scheduling, which basically consist of assigning a given amount of resources to different users. The context of this function is, therefore, the amount of available resources, and the different users requesting them. Our implemen-

[6] `https://osrg.github.io/ryu/`

tation includes an interface at MAC layer level to enable a dynamic resource management: each time a user gets authenticated, the MAC layer notifies the orchestrator, which replies with the amount of resources to be assigned to this user. We use ACHO just on selected events, to allow enough stability on the radio link. Although the ACHO mechanism do not impose any constraint on the frequency of re-orchestration, each re-orchestration impose a price in terms of resource re-allocation. Finally, by employing ACHO at the network edge, allows for a better network slice isolation, as demonstrated in Section 4

3.6. ACHO adoption strategies

As discussed in the previous subsections, adopting ACHO in the state of the art architecture requires fundamentally two new features: (*i*) the introduction of new relocatable functions (see Section 3.5) and (*ii*) their interaction with the MANO (see Section 3.4). Indeed, they require an important re-structure of the current network implementation strategies, but we believe that the advantages brought by our approach (i.e., the possibility of a fast re-orchestration of network functions) will certainly be considered in the upcoming transition to novel paradigms such as the cloud-native network functions [38].

Still, the changes from the architectural perspective are limited and, in some cases, even already partially targeted by the current standardization work. Summarizing, the new architectural interfaces shall be able to expose:

- **Network parameters**: as discussed in Section 3.4, ACHO envisions a new interface between the VNF and the MANO domains, that is used to perform extraction and injection of the context to and from virtual appliances. This kind of approach is totally aligned with current trends of network softwarization, which propose a profound restructuring of interfaces with an API based approach.

- **Network resource models**: the *context* of a VNF is tightly bound with its internal state, which is represented by a set of parameters usually associated to different granularity levels: per user (such as the bearer information), per user group or slice (such as the IoT broker) or globally to the VNF (like the eNB configuration). All these aspects are discussed in Section 3.5.

To this end, we next propose two implementation strategies that are aligned with the current efforts by SDOs.

- **Transparent mode:** While the network functions shall provide an API to extract and inject their *context*, its definition may be actually up to the vendor. Therefore, the data blob comprising the context of a network function at a certain point in time can be transparently handled by the MANO through the *or-nfv* interface, which simply transfers it to another location. Then the consistency is provided internally by the VNF vendor. This is the strategy used in our implementation discussed in Section 4.

- **Exposed mode:** defining the parameters that are used by a VNF is a task that has already been carried out by 3GPP SA5 for management purposes. For instance, [39] defines such parameters list for every network function defined in the 5G Core and RAN. Thus, *context* can be exposed following a standardized approach, to enable inter-vendor migration and enhanced management functionality at the MANO side (e.g., extract the context from one VNF and split it into several virtual appliances).

4. Performance evaluation

To evaluate the performance of ACHO, we have deployed a testbed consisting of an access network and three datacenters, one acting as "central cloud" and two acting as "edge clouds," which run the components presented in the previous section. Over this setup, three services (eMBB, mMTC, and URLLC) are provided as illustrated in Fig. 6. Arrows serve to indicate the four re-orchestrations that we perform and are described in Section 4.3. We remark that, since the same UE may connect to the same attachment point for different slices, the mobility management and authentication procedures can be shared across slices, and so are the AMF, AUSF and UDM functions (the "Shared Functions" in Fig. 6). This relies on the network function sharing functionality, which is mandated by 3GPP [14].

In this section we thus evaluate the performance obtained by ACHO under a set of different metrics: VNF relocation delays (see Section 4.2), and the re-orchestration of the VNFs discussed in Section 2.1 (in Section 4.3).

4.1. Testbed description

The testbed, depicted in Figure 7 is entirely composed of commodity hardware, which shows that ACHO does not have any particular hardware

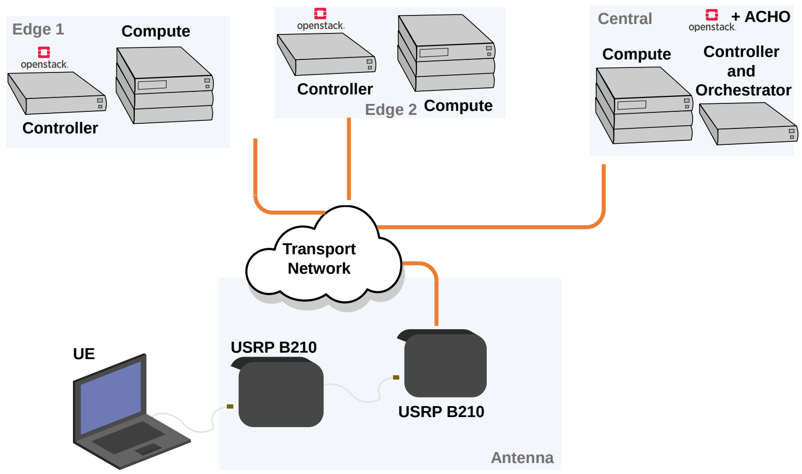


Figure 7: The Physical testbed setup.

requirement. The access network consists of a physical UE and virtual UEs. The physical UE runs in a laptop with Ubuntu 16.04, and the radio link is implemented by two Ettus USRP B210 SDR cards cross-connected with RF cables. The multi-slice eNB software runs in an Intel NUC with an Ubuntu 18.04. The same machine hosts the Virtual RAN software for the mMTC deployment.

The datacenters run OpenStack, with one controller node that manages the virtual links connecting the VNFs. They are hosted in Ubuntu 16.04 servers, each server equipped with two network cards: one acting as the provider network (i.e., carrying the 3GPP Network traffic), and the other carrying the control and management traffic. The transport network connecting the different datacenters consists of four Northbound Networks Zodiac FX Openflow-enabled switches. To emulate long-distance links (i.e., between edge and cloud), we use the Linux traffic shaper `tc`.

4.2. VNF relocation delay

We start our evaluation by focusing on the delay to perform a relocation of a VNF, which is defined as the time elapsed between the MANO taking the relocation decision, and the moment in which the VNF is up and running

	ACHO				OpenStack
VNF	Run	Pool	Cached	Non-C.	
UPF	70 ms	28.3 s	1 m 11.2 s	2 m 29.3 s	74 m 40 s
IoT br.	72 ms	28.8 s	1 m 5.7 s	2 m 29.2 s	89 m 35 s
FW	71 ms	27.7 s	1 m 3.3 s	2 m 27.3 s	59 min 48 s

Table 1: VNF relocation delays obtained by ACHO and by OpenStack.

in the new location.[7] We measure the relocation delay for three of the VNFs described in Section 3.5: the UPF and the firewall (FW), each one running in a `nano` instance, and the IoT broker, which runs in a `small` instance (these VM flavors are inspired by the Amazon EC2 service). For all the considered VNFs, we evaluate the relocation delay incurred when using two different orchestration platforms: (*i*) ACHO, with four different configurations (discussed below), and (*ii*) the one obtained with OpenStack live migration. We provide the resulting relocation delays, corresponding to the average of 5 repetitions, in Table 1. This comparison allows us to quantify what are the advantages of a lightweight solution like ACHO with respect to a heavy migration technique such as the one provided by OpenStack. This scenario, which reflects a typical central cloud to edge cloud migration, cannot be properly handled directly through the VIM.

That is, the results confirm that OpenStack results extremely slow as compared with ACHO, for all the configurations. These configurations are: (*i*) *already running* (Run in the table), where the engine is already bootstrapped, (*ii*) *pool*, where the engine is already created in the new location, but not started; (*iii*) *cached*, where the target engine has already been started in the destination machine in the past; and (*iv*) *non-cached* (Non-C.), where the image of the engine is available at the new destination but has to be created and bootstrapped for the first time.

OpenStack results order of magnitude slower than any of these configurations, as moving a VNF requires moving a full copy of the engine (including memory and disks). This is the main showstopper for the direct application

[7] Note that we do not consider "live-migrations," since available orchestrators such as OpenStack can only perform this type of migration when disk or memory is shared across locations, something unfeasible in the scenarios we consider (e.g., a VNF relocated to a different and possibly far node).

of the live migration in an environment such as the one depicted here, in which a flat NFV infrastructure may not be available. In contrast, delays are much smaller with ACHO, which furthermore enables having an engine already running in the destination node, thus making the relocation delay almost negligible (note that delays could be further reduced by employing more lightweight engines, such as, e.g., Containers or Unikernels). These results are also aligned with the ones provided in [40].

We finalize this section by analyzing the relocation delay of `SENATUS` [5], the orchestration framework closest to our proposal (as we discussed in Section 2.2). `SENATUS` leverages the native OpenStack APIs to perform a full snapshot of the image running the VNF before moving it to the new location, which requires the service to be stopped during the migration. Using similar images to the ones reported in [5][8], we obtained migration times of approx. 130 s, a performance comparable to ACHO's *non-cached* configuration (in both cases, the image has to be created and bootstrapped for the first time). We note, however, that ACHO supports a "make before break" paradigm, as it only needs to stop the VNF in the old location when starting the context transfer, and not before. As a result, ACHO can re-orchestrate VNFs without any perceptible service interruption (as we confirm next), while `SENATUS` would incur in a service disruption during this 130 s interval.

4.3. Performance under re-orchestration

Next, we evaluate the impact of re-orchestration on performance. To this aim, we have performed four experiments:

Experiment 1: *Service function chain re-orchestration.* One key feature of ACHO is the ability to seamlessly modify the function chain of a service already running, i.e., adding or removing a VNF. We tested this feature in the eMBB network slice by adding a new firewall function to support a new requirement. Using the interface `or-nfv` described in Section 3.4, injecting the state is an atomic operation decoupled from the execution environment of the network function.

Our experiment starts with the eMBB slice serving three TCP flows, namely A, B, and C. After 30 s, we enforce a new policy by adding a firewall function into the slice and injecting the firewall rules (as context) through the

[8]SENATUS is evaluated using CirrOS images, which by default do not have a context as they do not run a proper VNFs. So we could not test ACHO's mechanism against this setup.

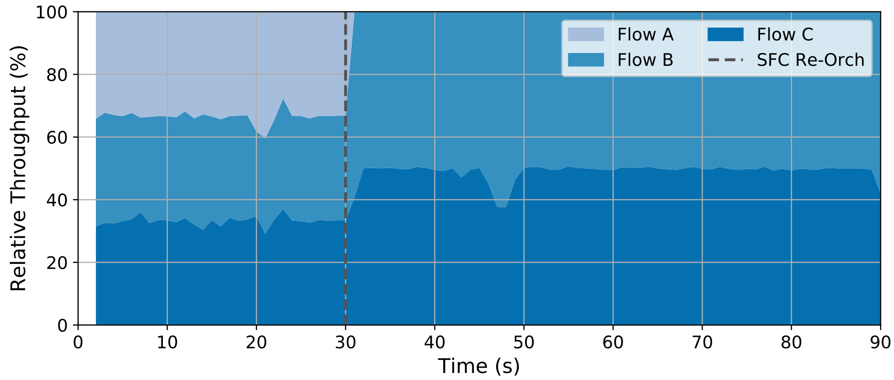


Figure 8: SFC amendment. Flow A (top), B (middle) and C (bottom).

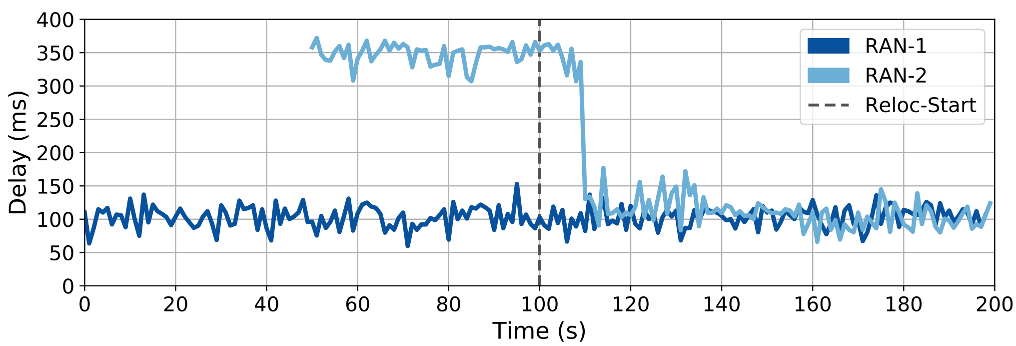


Figure 9: Relocation of the IoT gateway across edge clouds.

`or-nfv` interface. These rules match flow A, which is immediately interrupted without affecting the rest of the flows of this slice. We plot the throughput obtained by each flow in Fig. 8, which illustrates that re-orchestration does not disrupt the performance of the ongoing services.

Experiment 2: *Follow-the-load VNF relocation.* Next, we consider an mMTC service in a scenario with two RANs and two edge clouds. Initially, all the UEs are connected to RAN 1, which is closest to Edge 1 and therefore both the UPF and the IoT Broker application are orchestrated there. This results in a Round Trip Time (RTT) for the application of approx. 100 ms, as Fig. 9 shows. Then, at time t=50 s, half the UEs are moved to RAN 2, which is farther away from Edge 1, this resulting in RTTs of approx. 370 ms. This performance degradation is detected by the MANO via the `or-mon` interface, which reacts by instantiating new UPF and IoT Broker in Edge 2 and, once these are available, relocating the context of those UEs that moved into them. This whole process (i.e., creating a new VM with the VNF image and, once ready, copy the context) takes approx. 30 s (which corresponds to the "pool" strategy in Table 1) and the service is never disrupted, nor for the UEs that stay in RAN 1 nor for those that move to RAN 2. These results show the ability of ACHO to flexibly relocate only selected parts of a context.

Experiment 3: *Bringing VNF closer to users.* Next, we demonstrate the ability of ACHO to relocate VNFs inside the same slice. To this aim, we consider the URLLC slice, supporting a 600 kbps application that experiences a delay of approx. 150 ms. At some point, the MANO marks this delay as excessive and triggers a re-orchestration of the slice. This re-orchestration involves the relocation of the UPF and the low latency application (i.e., augmented reality in this case, marked as AR in Fig. 6), bringing both of them closer to the UE (i.e., from the central to the edge cloud). We analyze the resulting performance using three of the ACHO strategies discussed in Section 4.2, namely, Pool, Cached and Non-C. To this aim, we depict in Fig. 10 the performance since the MANO triggered the re-allocation in terms of connectivity (i.e., frames received, top subplot), and delay (bottom subplot).

The results confirm that (*i*) the re-orchestration is performed seamlessly towards the application, which perceives no disruption (i.e., no frames are lost), (*ii*) performance in terms of latency improves due to the relocation of the VNFs, (*iii*) migration delays (time between $t = 0$ and the thick black ticks in the figure) are in line with those presented in Section 4.2, with the "pool" strategy providing the smallest latency and the "non-cached" the largest one.

Experiment 4: *On-demand radio resources assignment.* In this experiment, we consider two users (UE1 and UE2) of a video streaming services. Each user requests at the beginning of the experiment a low quality video, therefore the orchestrator assigns the same amount of resources to each of them. At time t=30 s, *UE1* requests a higher quality video (720p) and the orchestrator reacts by assigning more resources to that flow. Similarly, at time t=60 s *UE2* requests a higher quality video, triggering a similar re-configuration. We provide in Fig. 11, the resulting throughput obtained by each user.

The insights about the above reconfiguration are provided next. The eNB is configured with a bandwidth of 10 MHz of bandwidth, which translates into 16 RBGs (Resource Block Groups) of 3 PRBs (Physical Resource Blocks),

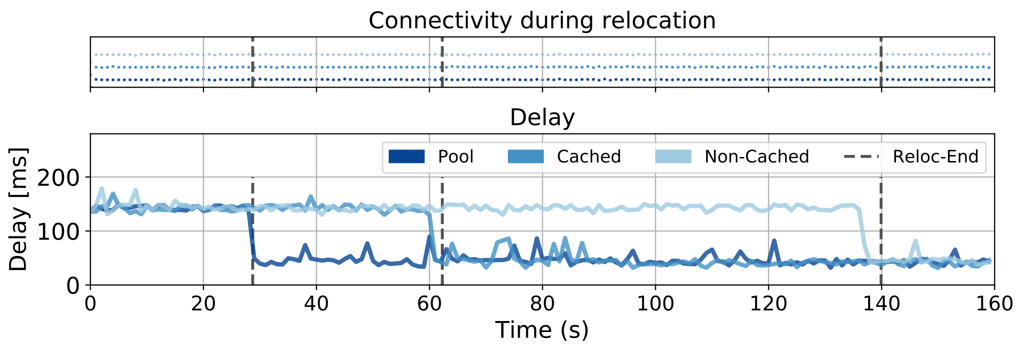


Figure 10: UPF migration from the central cloud to the edge cloud, under different configurations.

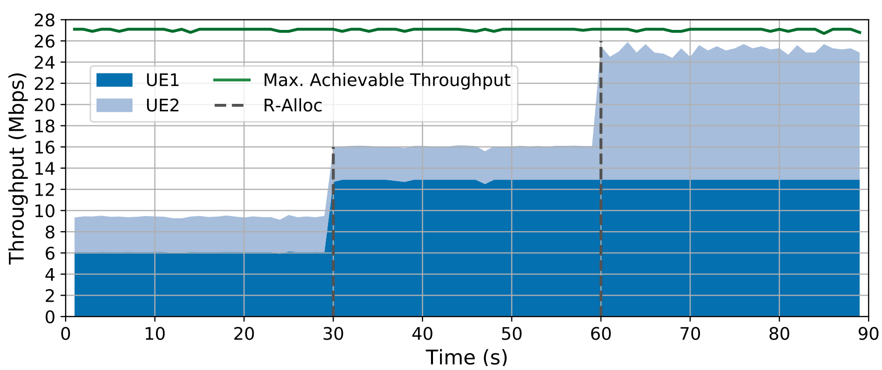


Figure 11: On-demand Radio resources assignment

and 1 RBG of 2 PRBs. The initial assignment is 4 RBGs per *UE*, which supports the transmission of a 480p video. Then, at time t=30 s (t=60 s) the orchestrator assigns four more RBGs to *UE1* (to *UE2*) to support the transmission of a 720p video. As the Fig. 11 confirms, we can dynamically assign resources to UE with strong guarantees on their isolation (i.e., increasing the bandwidth for one UE does not affect the other).

5. Conclusion

We have proposed a new framework to flexibly re-orchestrate a virtualized mobile network. This framework allows to re-orchestrate network slices on the fly without disrupting ongoing services, which can greatly improve performance under changing conditions. We have developed an implementation of a 5G protocol stack that realizes it, and have applied it to VNFs of different nature. We have evaluated the resulting performance in a realistic network slicing setup, showing the feasibility and advantages of flexible re-orchestration. We believe that flexible re-orchestration framework envisioned and implemented for this work fits very well the current trends in network softwarization followed by the industry. As future work, more functions can be implemented, as well as the exposed mode discussed in the paper.

Acknowledgements

This work was partially supported by the European Commission in the framework of the H2020 5G-PPP 5G-TOURS (Grant no. 856950) project and the 5G-EVE (Grant no. 815074) project.

[1] A. A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges, Computer Networks 167 (2020) 106984. doi:https://doi.org/10.1016/j.comnet.2019.106984.

[2] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, D. Lopez, Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks, IEEE/OSA Journal of Optical Communications and Networking 7 (2015) B62–B70.

[3] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, X. Costa-Perez, How should i slice my network?: A multi-service empirical evaluation of resource sharing efficiency, in: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, MobiCom '18, ACM, New York, NY, USA, 2018, pp. 191–206. URL: http://doi.acm.org/10.1145/3241539.3241567. doi:10.1145/3241539.3241567.

[4] V. Sciancalepore, C. Mannweiler, F. Z. Yousaf, P. Serrano, M. Gramaglia, J. Bradford, I. Labrador Pavn, A future-proof architecture for management and orchestration of multi-domain nextgen networks, IEEE Access 7 (2019) 79216–79232. doi:10.1109/ACCESS.2019.2923364.

[5] S. Troia, A. Rodriguez, R. Alvizu, G. Maier, Senatus: An experimental sdn/nfv orchestrator, in: 2018 IEEE Conference on Network Function

Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–5. doi:10.1109/NFV-SDN.2018.8725690.

[6] L. Jorguseski, A. Pais, F. Gunnarsson, A. Centonza, C. Willcock, Self-organizing networks in 3GPP: standardization and future trends, IEEE Communications Magazine 52 (2014) 28–34. doi:10.1109/MCOM.2014.6979983.

[7] J. T. J. Penttinen, Core Network, Wiley, 2019, pp. 139–186. URL: https://ieeexplore.ieee.org/document/8788396. doi:10.1002/9781119275695.ch6.

[8] Open RAN Alliance, O-ran: Towards an open and smart ran, White Paper (2018).

[9] X. Foukas, M. K. Marina, K. Kontovasilis, Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture, in: Proceedings of the 23rd annual international conference on mobile computing and networking, ACM, 2017, pp. 127–140.

[10] B. Ger, D. Jocha, R. Szab, J. Czentye, D. Haja, B. Nmeth, B. Sonkoly, M. Szalay, L. Toka, C. J. Bernardos Cano, L. M. Contreras Murillo, The orchestration in 5G exchange A multi-provider NFV framework for 5G services, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1–2.

[11] L. Ma, X. Wen, L. Wang, Z. Lu, R. Knopp, An SDN/NFV based framework for management and deployment of service based 5G core network, China Communications 15 (2018) 86–98.

[12] D. M. Gutierrez-Estevez, M. Gramaglia, A. de Domenico, N. di Pietro, S. Khatibi, K. Shah, D. Tsolkas, P. Arnold, P. Serrano, The path towards resource elasticity for 5g network architecture, in: 2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2018, pp. 214–219. doi:10.1109/WCNCW.2018.8369027.

[13] J. Ortin, C. Donato, P. Serrano, A. Banchs, Resource-on-demand schemes in 802.11 wlans with non-zero start-up times, IEEE Journal on Selected Areas in Communications 34 (2016) 3221–3233. doi:10.1109/JSAC.2016.2624158.

[14] 3GPP TS23.501, System Architecture for the 5G System,, Rel. 15, 2018.

[15] 3GPP TR28.801, telecommunication management;study on management and orchestration of network slicing for next generation network, Rel. 15, 2018.

[16] Xin Li, Chen Qian, A survey of network function placement, in: 2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC), 2016, pp. 948–953.

[17] A. Laghrissi, T. Taleb, A survey on the placement of virtual resources and virtual network functions, IEEE Communications Surveys Tutorials 21 (2019) 1409–1434.

[18] H. Talebian, A. Gani, M. Sookhak, A. A. Abdelatif, A. Yousafzai, A. V. Vasilakos, F. R. Yu, Optimizing virtual machine placement in IaaS data centers: taxonomy, review and open issues (????). URL: https://doi.org/10.1007/s10586-019-02954-w. doi:10.1007/s10586-019-02954-w.

[19] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, R. Buyya, Cloud service reliability enhancement via virtual machine placement optimization, IEEE Transactions on Services Computing 10 (2017) 902–913.

[20] OSM Release FIVE Technical Overview, https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf, 2019. Online; accessed Apr. 2020.

[21] ONAP Architecture Overview whitepaper, https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf, 2019. Online; accessed Apr. 2020.

[22] OSM Information Model, https://osm.etsi.org/wikipub/index.php/OSM_Information_Model, 2019. Online; accessed Dec. 2019.

[23] M. E. Elsaid, C. Meinel, Live migration impact on virtual datacenter performance: Vmware vmotion based study, in: 2014 International Conference on Future Internet of Things and Cloud, 2014, pp. 216–221. doi:10.1109/FiCloud.2014.42.

[24] F. Zhang, G. Liu, X. Fu, R. Yahyapour, A survey on virtual machine migration: Challenges, techniques, and open issues, IEEE Communications Surveys Tutorials 20 (2018) 1206–1243. doi:10.1109/COMST.2018.2794881.

[25] Openstack Docs live-migrate instances, https://docs.openstack.org/nova/pike/admin/live-migration-usage.html, 2019. Accessed: Dec 2019.

[26] VMware vSphere vMotion architecture, performance and best practices in vmware vsphere 5: Performance study, Technical White Paper, 2011, 2019. Online; accessed Dec. 2019.

[27] S. Nadgowda, S. Suneja, N. Bila, C. Isci, Voyager: Complete Container State Migration, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2137–2142. doi:10.1109/ICDCS.2017.91.

[28] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, Unikernels: Library operating systems for the cloud, SIGPLAN Not. 48 (2013) 461–472. URL: http://doi.acm.org/10.1145/2499368.2451167. doi:10.1145/2499368.2451167.

[29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, Opennf: Enabling innovation in network function control, in: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, ACM, New York, NY, USA, 2014, pp. 163–174. URL: http://doi.acm.org/10.1145/2619239.2626313. doi:10.1145/2619239.2626313.

[30] S. Rajagopalan, D. Williams, H. Jamjoom, A. Warfield, Split/merge: System support for elastic execution in virtual middleboxes, in: Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), USENIX, Lombard, IL, 2013, pp. 227–240. URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/rajagopalan.

[31] 3GPP TS23.502, Procedures for the 5G System (5GS); stage 2 (release 16),, Rel. 16, 2020.

[32] X. Feng, J. Tang, X. Luo, Y. Jin, A performance study of live vm migration technologies: Vmotion vs xenmotion, in: 2011 Asia Communications and Photonics Conference and Exhibition (ACP), 2011, pp. 1–6. doi:10.1117/12.905512.

[33] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, D. J. Leith, srslte: An open-source platform for lte evolution and experimentation, in: Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization, WiNTECH '16, ACM, New York, NY, USA, 2016, pp. 25–32. URL: http://doi.acm.org/10.1145/2980159.2980163. doi:10.1145/2980159.2980163.

[34] G. Garcia-Aviles, M. Gramaglia, P. Serrano, A. Banchs, POSENS: A Practical Open Source Solution for End-to-End Network Slicing, IEEE Wireless Communications 25 (2018) 30–37. URL: https://ieeexplore.ieee.org/document/8524891/. doi:10.1109/MWC.2018.1800050.

[35] ETSI, network functions virtualisation (nfv) release 3; evolution and ecosystem; report on network slicing support with etsi nfv architecture framework, 2017.

[36] 3GPP TS29.510, 5G System; Network function repository services; Stage 3 (Release 15),, Rel. 15, 2020.

[37] ONOS Project, https://onosproject.org, 2019. Online; accessed Dec. 2019.

[38] Cloud-native network functions, Cisco White Paper, 2018. URL: https://www.cisco.com/c/en/us/solutions/service-provider/industry/cable/cloud-native-network-functions.html.

[39] 3GPP TS28.541, Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3,, Rel. 15, 2018.

[40] 5G-CORAL, Refined design of 5G-CORAL orchestration and control system and future directions, D3.2, 2019.