

This is a postprint version of the following published document:

García-Avilés, G., et al. ACHO: a framework for flexible re-orchestration of virtual network functions, In *Computer Networks*, 158, Oct. 2020, 107382, 11 Pp.

DOI: <https://doi.org/10.1016/j.comnet.2020.107382>

© 2020 Elsevier B.V. All rights reserved.



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

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

Gines Garcia-Aviles<sup>1</sup>, Carlos Donato<sup>2</sup>, Marco Gramaglia<sup>1</sup>, Pablo Serrano<sup>1</sup>,  
Albert Banchs<sup>1,3</sup>

---

## 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

---

<sup>1</sup>University Carlos III of Madrid

<sup>2</sup>IMEC

<sup>3</sup>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 Virtual Network Functions (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 Radio Access Network (RAN) part some similar efforts have been proposed [8], but almost no implementation is available (the one in [9] is not provided as open

38 source). In this paper, we propose ACHO (Adaptive slice re-Configuration  
39 using Hierarchical Orchestration), a novel open source framework for *flexible*  
40 orchestration of network functions, which (i) provides the ability to relocate  
41 VNFs at run-time, and (ii) supports their fine-grained re-configuration.

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

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

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

54 By open-sourcing ACHO, we aim to foster 5G experimenting repeatabil-  
55 ity, to improve code reliability, and to enable other researchers to extend the  
56 number of supported scenarios beyond those studied in this paper. The code-  
57 base, available on GitHub<sup>4</sup> under the AGPLv3 license, is the first open-source  
58 solution of basic 5G Core functionality with seamless re-location capabilities  
59 (to the best of our knowledge).

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

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

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

---

<sup>4</sup><https://github.com/wn1UC3M/>

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

78 In the following, we first make the case for these two features, and then  
79 discuss the state of the art technology and the current implementation land-  
80 scape.

### 81 *2.1. The case for flexible re-orchestration of VNFs*

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

#### 86 *2.1.1. Advantages of a dynamic re-orchestration*

87 Some of these advantages obtained with a dynamic re-orchestration are:  
88 **Adapting to user mobility.** Low-latency services such as tactile or ve-  
89 hicular communications require that VNFs affecting latency are as close as  
90 possible to the user, to minimize the delay between these functions and the  
91 user. When a user moves to a new location, VNFs should move as well to  
92 keep close to the user's new location. This requires the ability to relocate  
93 those functions without disrupting the ongoing service.

94 **Service enhancements in run-time.** Flexible re-location also gives the  
95 ability to re-compose a service provided by a given slice, to add, substitute,  
96 remove, or relocate VNFs in the chain. This enables introducing a variety  
97 of features during run-time operation, such as, e.g., adding or relocating a  
98 firewall, or replacing a more efficient (but slower) video encoder by a quicker  
99 but less efficient one to adapt to changes in the measured delay while keeping  
100 quality of experience.

101 **Improved de/scaling.** Resource scaling refers to the ability to assign re-  
102 sources as needed. While the *traditional* vertical and horizontal scaling could  
103 provide this feature to some extent, the use of relocatable VNFs introduces  
104 an additional level of flexibility without disrupting the service: when a VNF  
105 runs out of resources, it can be relocated to a different location with more

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

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

#### 114 *2.1.2. Advantages of a fine-grained re-orchestration*

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

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

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

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

#### 141 *2.2. State of the art*

142 Despite the advantages discussed above, the technology currently avail-  
143 able does not support a flexible re-orchestration of VNFs. In the following,  
144 we review the state of the art, highlighting the most relevant initiatives and  
145 contributions.

### 146 *2.2.1. General VNF placement and orchestration problems*

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

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

### 166 *2.2.2. General-purpose orchestration technologies*

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

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

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

- 185 • They especially target limited parts of a VM such as its memory. These  
186 techniques use an iterative process [23] that starts from the memory  
187 pages that were the least frequently accessed, keep updating them until  
188 the ones that are the most used are moved. While relocating memory,  
189 usually a significant part of the VM has to be kept in a fixed location  
190 (e.g., a NAS hosting the disks). As a result, the relocation capability  
191 is limited.
  
- 192 • The relocation of a VM is limited within the boundaries of a single  
193 datacenter of a single VIM. These limitations are acceptable for cloud  
194 computing environments, which typically focus on very high reliability  
195 and therefore VMs are only relocated in case of, e.g., disk failures or  
196 programmed maintenance, but are inadequate for dynamic scenarios  
197 such as the use cases discussed above, involving the movement of VNFs  
198 across the network to reduce latency or to improve efficiency across  
199 datacenters. As a matter of fact, the topic of migration over WAN links  
200 (which is a relevant scenario for networking purposes, e.g., migration to  
201 edge cloud) is currently overlooked by the bulk of available literature, as  
202 also confirmed by the authors of [24]. Among the more than 200 works  
203 reviewed there, just a handful deal with migration over long distances  
204 and none of them provide experimental results.

205 As discussed, state of the art orchestration platforms provide some meth-  
206 ods to relocate VMs. For instance, OpenStack provides live-migration tools  
207 [25]; however, their use precludes fast VNF re-location. Their operation is  
208 sketched in Fig. 1a: in contrast to ACHO, the full Virtual Machine has to be  
209 copied to the targeted destination. This includes common data such as the  
210 guest kernel, libraries, and the file system structure [23]: these elements are  
211 not copied when employing ACHO's context migration. Also, as these tech-  
212 niques are very expensive in terms of exchanged data, they are only available



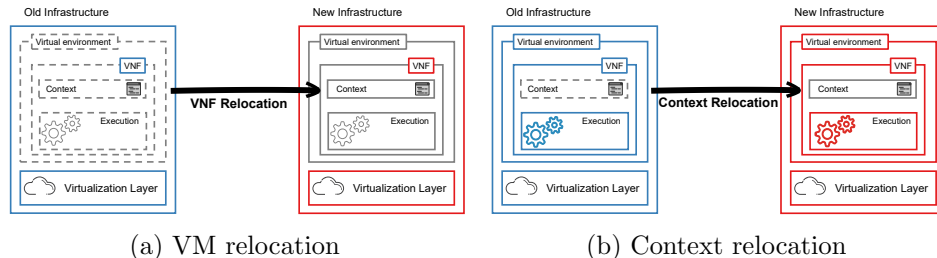


Figure 1: Relocation strategies: a full VNF relocation (left) vs. the context relocation performed with ACHO (right).

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

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

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

### 236 2.2.3. *Ad hoc solutions*

237 Enabling flexible re-orchestration in softwarized network deployment re-

238 ceived attention by the research community in the last few years. The work  
239 most closely related to ours is **SENATUS** [5], a framework that internally lever-  
240 ages on state of the art VIMs; because of this, this framework yields to very  
241 poor performance, in particular in challenging (i.e., very dynamic) scenarios,  
242 as we quantify in Section 4.

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

252 A similar idea is also currently included in the 3GPP specification [31], to  
253 relocate the information related to a specific User Equipment (UE) between  
254 different instances of a network function, in particular for load balancing  
255 purposes or to keep providing connectivity when a specific function is de-  
256 commissioned. Still, this procedure is available for the core functions only.

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

### 261 *2.3. Main ACHO novelties*

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

- 267 • Firstly, ACHO targets mobile networking, a more heterogeneous sce-  
268 nario with very diverse network functions with very different require-  
269 ments (e.g., access network vs. core network functions).
- 270 • ACHO provides a clear methodology to adapt existing VNFs, which  
271 follows the recent architectural trends of 5G networking, and is aligned  
272 with the ongoing standardization efforts.

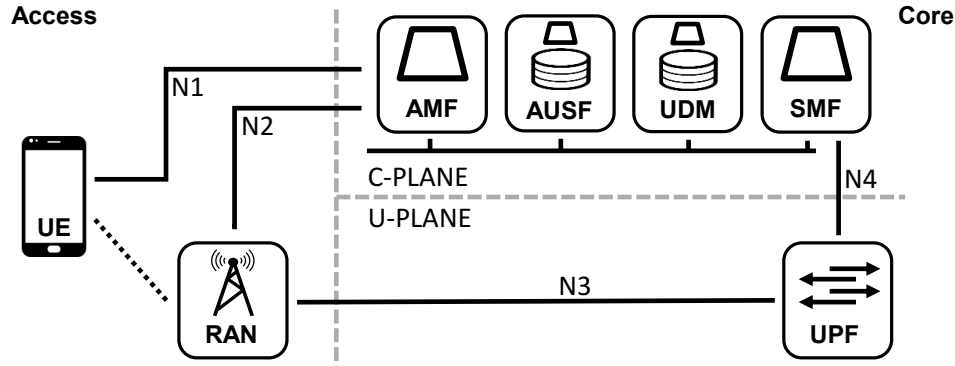


Figure 2: The selected 5G Core functions implemented for the tests.

- 273 • ACHO specifies two novel interfaces to support and dynamic and fine-  
 274 grained orchestration, which can be easily implemented with existing  
 275 off-the-shelf orchestrators.
- 276 • ACHO also provides a fully-featured implementation of 5G VNFs and  
 277 orchestration elements, which can be easily downloaded, customized,  
 278 and tested with off-the-shelf hardware.
- 279 • Finally, we discuss different implementation strategies, also aligned  
 280 with existing standardization efforts, to maximize the practicality of  
 281 ACHO.

### 282 3. ACHO: A suite for flexible 5G networking

283 We next present ACHO (Adaptive slice re-Configuration using Hierarchi-  
 284 cal Orchestration), a software framework consisting of an implementation of  
 285 5G Functionality (the most critical 3GPP Rel. 15 Core Network Functions,  
 286 depicted in Fig. 2 and marked with SBA in Fig. 5), the radio access network  
 287 functions and the MANO modules to handle them. ACHO provides a full  
 288 network-slicing aware solution that includes all the MANO modules to en-  
 289 able a flexible re-orchestration of the mobile network. Some of the network  
 290 components of ACHO have been adapted from existing open source projects  
 291 (e.g., srsLTE), while other components that were not available as open source

292 have been implemented from scratch. As a result, the software codebase pro-  
293 vided by ACHO is very complete and has no match in the landscape of the  
294 open source mobile networking initiatives.

### 295 *3.1. Efficient re-configuration of VNFs through context migration*

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

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

### 316 *3.2. The VNF context*

317 As discussed above, by introducing the *c/e* split, ACHO trades flexibil-  
318 ity (i.e., the orchestration framework needs to know what kind of VNFs are  
319 running), with the compactness of the exchanged data, which is the bare min-  
320 imum data representing the internal state of a VNF. The context is specific  
321 to each VNF but it is independent of the execution environment: for in-  
322 stance, we implemented ACHO for a VM-based deployment, but it can work  
323 with containers or unikernels. A context may comprise the specific rules of  
324 a firewall, or the information of the authenticated User equipment (UE) for  
325 an Authentication Server Function (AUSF). To support this, network func-  
326 tions need to be re-implemented to enable a clear separation between the  
327 context and the engine, a re-implementation that is specific to each function.

328 Furthermore, these re-implemented VNF have to expose this new capability,  
329 which could be achieved by e.g. extending the Network Exposure Function  
330 (NEF) to support c/e split through an API.

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

342 Thus, according to operator-defined re-orchestration triggers (which can  
343 be computed from QoS metrics), the MANO pulls the context from the source  
344 VNF and injects it in the destination VNF (details on the specific interfaces  
345 are provided in Section 3.4). Hence, in the SMF case discussed here, all the  
346 information related to the gNB and Gateway, including the tunnel id for each  
347 user is moved to the new location. Hence, the target VNF can immediately  
348 start serving the UE traffic from the new location.

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

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

```

1 {
2   "enb_tun_ip_addr": "192.16
3     8.10.12",
4   "gw_tun_ip_addr": "192.168
5     .10.10",
6   "enb_tun_hw_addr": "xx:xx:
7     xx:xx:xx:xx",
8   "gw_tun_hw_addr": "yy:yy:
9     yy:yy:yy:yy",
10  "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

```

1 {
2   "nsl_id": "1",
3   "rnti": ["1", "2", "3"],
4   "start_rb_id": 1,
5   "stop_rb_id": 20,
6 }

```

Figure 4: Representation of the MAC context

### 363 3.3. Baseline 5G implementation

364 To illustrate the benefits of the c/e split we need a working baseline  
365 implementation of certain 5G functionality, neither supported by current  
366 versions of 3GPP (i.e., 4G/LTE) nor existing open-source implementations.  
367 We next describe the baseline architecture that we have developed, which  
368 includes: (i) a multi-slice capable access network (both in the UE and eNB),  
369 to support end-to-end network slicing, and (ii) a *modular* implementation of  
370 the Core Network, as mandated by recent 3GPP standards.

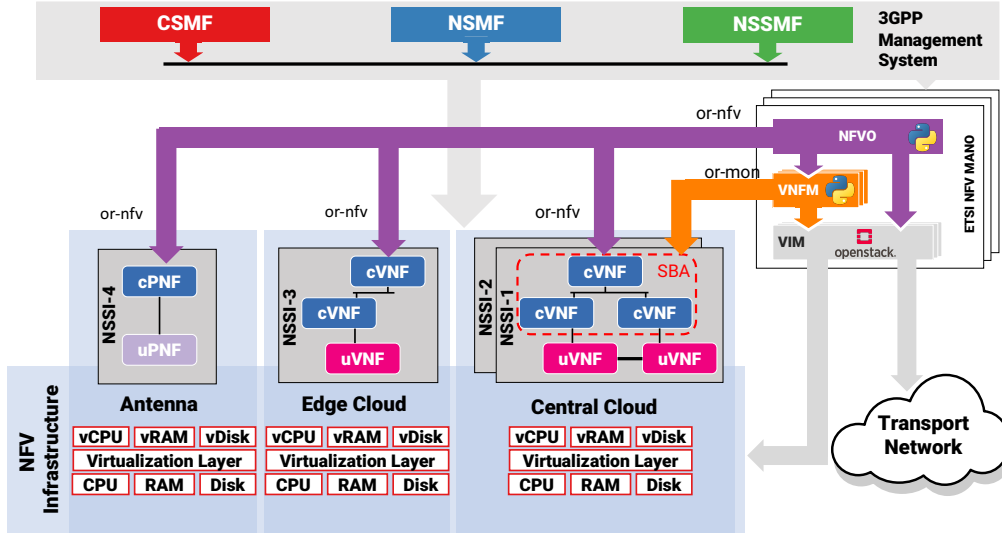


Figure 5: MANO Implementation and new Interfaces. ACHO creates new interfaces in the reference points defined by ETSI and acts on the underlying virtual or physical NFs (both c-plane and u-plane) to provide a fast re-location.

371 **Radio Access Network** Our Radio Access Network (RAN) implementation  
 372 is based on the open source software suite srsLTE [33], which is extended to  
 373 support multiple slices. This *Multi-slice RAN* builds on a modified version  
 374 of srsLTE to support multiple slices on the same radio VNFs (the full imple-  
 375 mentation details of the baseline are available in [34]), that we have extended  
 376 to support fine-grained reconfiguration of radio resources (see Section 3.5).

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

### 384 3.4. New MANO functionality

385 The adoption of the c/e split requires novel MANO functionality, to en-  
 386 able the relocation of network functions, and new interfaces between the  
 387 management and orchestration layers and the VNFs, to extract and install  
 388 the contexts.

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

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

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

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

418 (i) `or-nfv`: This interface is used to extract and push the context of  
419 the VNFs. This interface is used by the Orchestrator (the NFVO), which  
420 is in charge of all the operational logic of a Network Slice. In particular,  
421 when deciding to relocate a function, the NFVO first extracts the context  
422 of the network function and then re-orchestrates this function, by pushing  
423 the context into the function available at the new location. This interface is  
424 similar to the one already included in the 5G system between the management  
425 service and the core network functions. This interface [36], connects the  
426 capabilities provided by the Network Exposure Function (NEF) and Network



427 Repository Function (NRF) to extract and set configuration parameters from  
428 the network functions. In our implementation, following the current trends in  
429 network softwarization, this interface is implemented through a REST API.

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

### 439 3.5. *Re-orchestrable VNFs*

440 The proposed c/e split can be applied to any VNF, provided it implements  
441 the interfaces described above to extract and install the context. To show  
442 this, we have implemented different VNFs following the c/e split and thus  
443 making them “re-orchestrable.” We note that the c/e split nicely fits with  
444 the SDN approach, which is an easy way to extract and inject the context  
445 from and to a VNF (i.e., in traditional SDN, the context of a switch are  
446 its forwarding rules). Thus, to implement the VNFs, for simplicity we have  
447 selected the Open Source *Lagopus* switch<sup>5</sup> as basis for our implementation  
448 (alternatives such as ONOS [37] may be used for larger deployments). The  
449 diversity of the chosen functions shows the generality of our approach and  
450 the ability to apply it to any network function:

451 **UPF**: This function provides the encapsulation, decapsulation, and forward-  
452 ing to the Packet Data Network. The implementation of this module follows  
453 the c/e split and includes the corresponding interfaces with our MANO sys-  
454 tem to extract and install the context. The context consists of the current  
455 rules applied to encapsulate/decapsulate packets and to forward them. We  
456 have implemented the UPF module building on the *Lagopus* switch.

457 **SMF**: This c-plane function controls and configures the UPF instances on  
458 the u-plane through the N4 [14] interface. Thus, the context here also con-  
459 sists of the rules to encapsulate/decapsulate/forward packets, in this case  
460 for all the UPF functions controlled by the SMF. For the implementation of

---

<sup>5</sup><http://www.lagopus.org>

461 this module, we leverage available SDN-capable implementations, enriching  
462 them with mobile network functionality, and employing a Ryu Controller<sup>6</sup> to  
463 implement the N4 interface between the UPF and the SMF.

464 **IoT broker:** The IoT broker acts as middleware between the sensors con-  
465 nected to a mobile network and a data sink that may be located in a central  
466 location. We have implemented this module in Python from scratch, in-  
467 cluding specific libraries for the handling of traffic flows from the sensors.  
468 Our lightweight and flexible implementation allows to dynamically transfer  
469 the broker context to a new location, which is particularly suitable for Mo-  
470 bile Edge Computing (MEC) deployments, as it allows moving the broker  
471 functionality across different edge infrastructures.

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

479 **MAC scheduler:** One of the main functions of MAC layer in LTE is the  
480 scheduling, which basically consist of assigning a given amount of resources  
481 to different users. The context of this function is, therefore, the amount of  
482 available resources, and the different users requesting them. Our implemen-  
483 tation includes an interface at MAC layer level to enable a dynamic resource  
484 management: each time a user gets authenticated, the MAC layer notifies  
485 the orchestrator, which replies with the amount of resources to be assigned  
486 to this user. We use ACHO just on selected events, to allow enough stability  
487 on the radio link. Although the ACHO mechanism does not impose any con-  
488 straint on the frequency of re-orchestration, each re-orchestration imposes a  
489 price in terms of resource re-allocation. Finally, by employing ACHO at the  
490 network edge, allows for a better network slice isolation, as demonstrated in  
491 Section 4

### 492 3.6. ACHO adoption strategies

493 As discussed in the previous subsections, adopting ACHO in the state of  
494 the art architecture requires fundamentally two new features: (i) the intro-

---

<sup>6</sup><https://osrg.github.io/ryu/>

495 duction of new relocatable functions (see Section 3.5) and (ii) their interac-  
496 tion with the MANO (see Section 3.4). Indeed, they require an important  
497 re-structure of the current network implementation strategies, but we be-  
498 lieve that the advantages brought by our approach (i.e., the possibility of  
499 a fast re-orchestration of network functions) will certainly be considered in  
500 the upcoming transition to novel paradigms such as the cloud-native network  
501 functions [38].

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

- 505 • **Network parameters:** as discussed in Section 3.4, ACHO envisions a  
506 new interface between the VNF and the MANO domains, that is used  
507 to perform extraction and injection of the context to and from virtual  
508 appliances. This kind of approach is totally aligned with current trends  
509 of network softwarization, which propose a profound restructuring of  
510 interfaces with an API based approach.
- 511 • **Network resource models:** the *context* of a VNF is tightly bound  
512 with its internal state, which is represented by a set of parameters  
513 usually associated to different granularity levels: per user (such as the  
514 bearer information), per user group or slice (such as the IoT broker)  
515 or globally to the VNF (like the eNB configuration). All these aspects  
516 are discussed in Section 3.5.

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

- 519 • **Transparent mode:** While the network functions shall provide an  
520 API to extract and inject their *context*, its definition may be actually  
521 up to the vendor. Therefore, the data blob comprising the context of a  
522 network function at a certain point in time can be transparently han-  
523 dled by the MANO through the *or-nfv* interface, which simply transfers  
524 it to another location. Then the consistency is provided internally by  
525 the VNF vendor. This is the strategy used in our implementation dis-  
526 cussed in Section 4.
- 527 • **Exposed mode:** defining the parameters that are used by a VNF is  
528 a task that has already been carried out by 3GPP SA5 for manage-  
529 ment purposes. For instance, [39] defines such parameters list for every

530 network function defined in the 5G Core and RAN. Thus, *context* can  
531 be exposed following a standardized approach, to enable inter-vendor  
532 migration and enhanced management functionality at the MANO side  
533 (e.g., extract the context from one VNF and split it into several virtual  
534 appliances).

## 535 4. Performance evaluation

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

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

### 550 4.1. Testbed description

551 The testbed, depicted in Figure 7 is entirely composed of commodity  
552 hardware, which shows that ACHO does not have any particular hardware  
553 requirement. The access network consists of a physical UE and virtual UEs.  
554 The physical UE runs in a laptop with Ubuntu 16.04, and the radio link is  
555 implemented by two Ettus USRP B210 SDR cards cross-connected with RF  
556 cables. The multi-slice eNB software runs in an Intel NUC with an Ubuntu  
557 18.04. The same machine hosts the Virtual RAN software for the mMTC  
558 deployment.

559 The datacenters run OpenStack, with one controller node that manages  
560 the virtual links connecting the VNFs. They are hosted in Ubuntu 16.04  
561 servers, each server equipped with two network cards: one acting as the  
562 provider network (i.e., carrying the 3GPP Network traffic), and the other

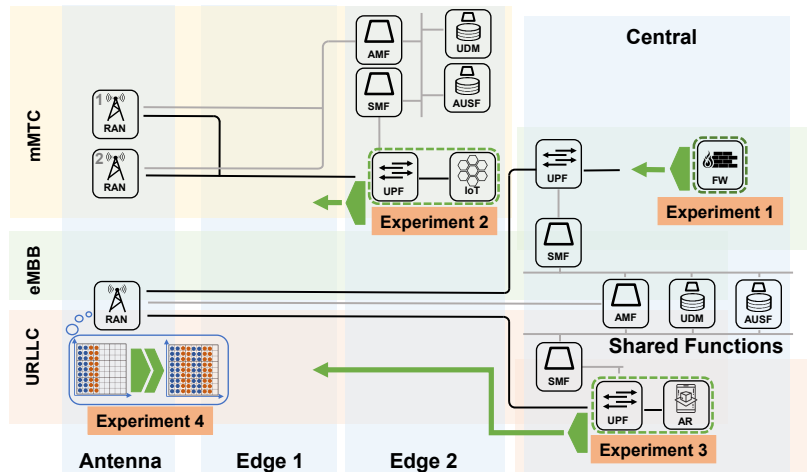


Figure 6: The Network Slice setup employed in the experimental evaluation, consisting of 3 slices.

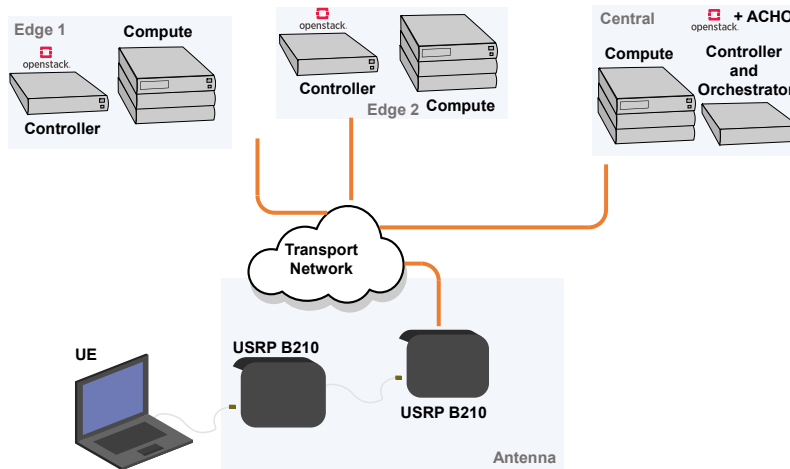


Figure 7: The Physical testbed setup.

563 carrying the control and management traffic. The transport network con-  
 564 necting the different datacenters consists of four Northbound Networks Zo-  
 565 diac FX Openflow-enabled switches. To emulate long-distance links (i.e.,  
 566 between edge and cloud), we use the Linux traffic shaper `tc`.

VNF	ACHO				OpenStack
	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.

567 *4.2. VNF relocation delay*

568 We start our evaluation by focusing on the delay to perform a relocation  
569 of a VNF, which is defined as the time elapsed between the MANO taking  
570 the relocation decision, and the moment in which the VNF is up and running  
571 in the new location.<sup>7</sup> We measure the relocation delay for three of the VNFs  
572 described in Section 3.5: the UPF and the firewall (FW), each one running  
573 in a `nano` instance, and the IoT broker, which runs in a `small` instance (these  
574 VM flavors are inspired by the Amazon EC2 service). For all the considered  
575 VNFs, we evaluate the relocation delay incurred when using two different  
576 orchestration platforms: (i) ACHO, with four different configurations (dis-  
577 cussed below), and (ii) the one obtained with OpenStack live migration.  
578 We provide the resulting relocation delays, corresponding to the average of  
579 5 repetitions, in Table 1. This comparison allows us to quantify what are  
580 the advantages of a lightweight solution like ACHO with respect to a heavy  
581 migration technique such as the one provided by OpenStack. This scenario,  
582 which reflects a typical central cloud to edge cloud migration, cannot be  
583 properly handled directly through the VIM.

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

---

<sup>7</sup>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).

591 created and bootstrapped for the first time.

592 OpenStack results order of magnitude slower than any of these configura-  
593 tions, as moving a VNF requires moving a full copy of the engine (including  
594 memory and disks). This is the main showstopper for the direct application  
595 of the live migration in an environment such as the one depicted here, in  
596 which a flat NFV infrastructure may not be available. In contrast, delays  
597 are much smaller with ACHO, which furthermore enables having an engine  
598 already running in the destination node, thus making the relocation delay  
599 almost negligible (note that delays could be further reduced by employing  
600 more lightweight engines, such as, e.g., Containers or Unikernels). These  
601 results are also aligned with the ones provided in [40].

602 We finalize this section by analyzing the relocation delay of **SENATUS** [5],  
603 the orchestration framework closest to our proposal (as we discussed in Sec-  
604 tion 2.2). **SENATUS** leverages the native OpenStack APIs to perform a full  
605 snapshot of the image running the VNF before moving it to the new location,  
606 which requires the service to be stopped during the migration. Using similar  
607 images to the ones reported in [5]<sup>8</sup>, we obtained migration times of approx.  
608 130 s, a performance comparable to ACHO’s *non-cached* configuration (in  
609 both cases, the image has to be created and bootstrapped for the first time).  
610 We note, however, that ACHO supports a “make before break” paradigm, as  
611 it only needs to stop the VNF in the old location when starting the context  
612 transfer, and not before. As a result, ACHO can re-orchestrate VNFs with-  
613 out any perceptible service interruption (as we confirm next), while **SENATUS**  
614 would incur in a service disruption during this 130 s interval.

#### 615 4.3. Performance under re-orchestration

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

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

---

<sup>8</sup>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.

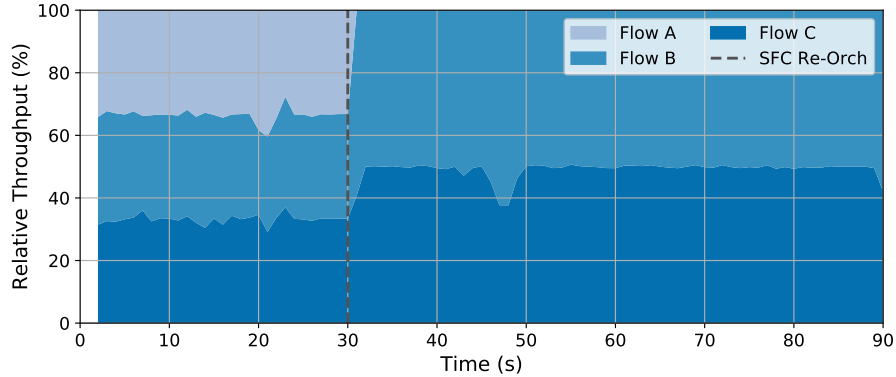


Figure 8: SFC amendment. Flow A (top), B (middle) and C (bottom).

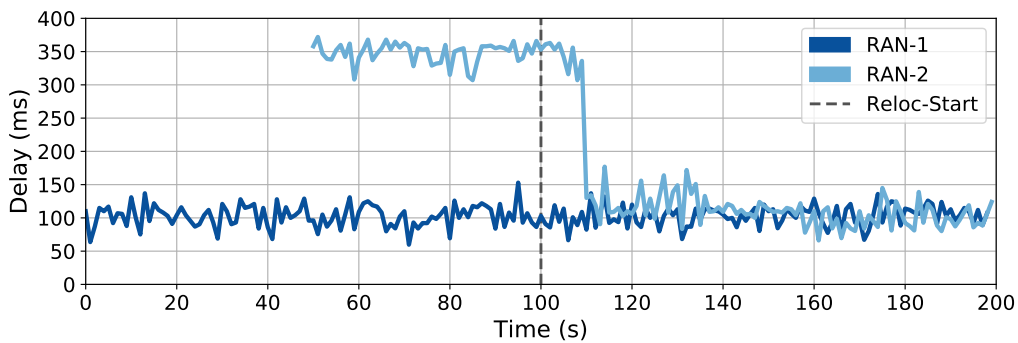


Figure 9: Relocation of the IoT gateway across edge clouds.

623 the state is an atomic operation decoupled from the execution environment  
 624 of the network function.

625 Our experiment starts with the eMBB slice serving three TCP flows,  
 626 namely A, B, and C. After 30 s, we enforce a new policy by adding a firewall  
 627 function into the slice and injecting the firewall rules (as context) through the  
 628 `or-nfv` interface. These rules match flow A, which is immediately interrupted  
 629 without affecting the rest of the flows of this slice. We plot the throughput  
 630 obtained by each flow in Fig. 8, which illustrates that re-orchestration does  
 631 not disrupt the performance of the ongoing services.

632 **Experiment 2: Follow-the-load VNF relocation.** Next, we consider an  
 633 mMTC service in a scenario with two RANs and two edge clouds. Initially,  
 634 all the UEs are connected to RAN 1, which is closest to Edge 1 and therefore



635 both the UPF and the IoT Broker application are orchestrated there. This  
636 results in a Round Trip Time (RTT) for the application of approx. 100 ms,  
637 as Fig. 9 shows. Then, at time  $t=50$  s, half the UEs are moved to RAN 2,  
638 which is farther away from Edge 1, this resulting in RTTs of approx. 370 ms.  
639 This performance degradation is detected by the MANO via the `or-mon` in-  
640 terface, which reacts by instantiating new UPF and IoT Broker in Edge 2  
641 and, once these are available, relocating the context of those UEs that moved  
642 into them. This whole process (i.e., creating a new VM with the VNF image  
643 and, once ready, copy the context) takes approx. 30 s (which corresponds  
644 to the “pool” strategy in Table 1) and the service is never disrupted, nor  
645 for the UEs that stay in RAN 1 nor for those that move to RAN 2. These  
646 results show the ability of ACHO to flexibly relocate only selected parts of a  
647 context.

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

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

667 **Experiment 4: *On-demand radio resources assignment.*** In this experiment,  
668 we consider two users (UE1 and UE2) of a video streaming services. Each  
669 user requests at the beginning of the experiment a low quality video, therefore  
670 the orchestrator assigns the same amount of resources to each of them. At  
671 time  $t=30$  s, *UE1* requests a higher quality video (720p) and the orchestrator  
672 reacts by assigning more resources to that flow. Similarly, at time  $t=60$  s

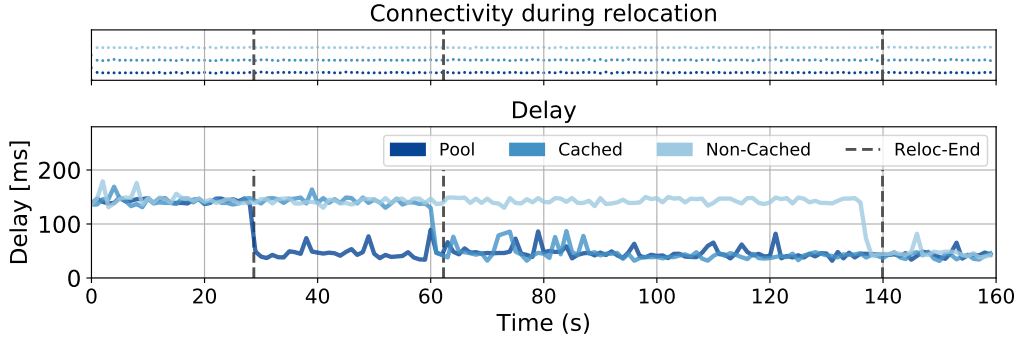


Figure 10: UPF migration from the central cloud to the edge cloud, under different configurations.

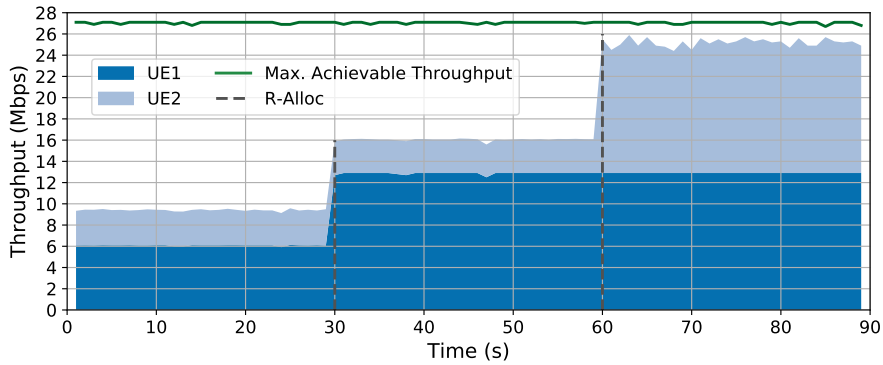


Figure 11: On-demand Radio resources assignment

673 *UE2* requests a higher quality video, triggering a similar re-configuration.  
 674 We provide in Fig. 11, the resulting throughput obtained by each user.

675 The insights about the above reconfiguration are provided next. The eNB  
 676 is configured with a bandwidth of 10 MHz of bandwidth, which translates into  
 677 16 RBGs (Resource Block Groups) of 3 PRBs (Physical Resource Blocks),  
 678 and 1 RBG of 2 PRBs. The initial assignment is 4 RBGs per *UE*, which sup-  
 679 ports the transmission of a 480p video. Then, at time  $t=30$  s ( $t=60$  s) the  
 680 orchestrator assigns four more RBGs to *UE1* (to *UE2*) to support the trans-  
 681 mission of a 720p video. As the Fig. 11 confirms, we can dynamically assign  
 682 resources to UE with strong guarantees on their isolation (i.e., increasing the  
 683 bandwidth for one UE does not affect the other).

684 **5. Conclusion**

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

696 **Acknowledgements**

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

- 700 [1] A. A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5G network slicing  
701 using SDN and NFV: A survey of taxonomy, architectures and future  
702 challenges, *Computer Networks* 167 (2020) 106984. doi:[https://doi.  
703 org/10.1016/j.comnet.2019.106984](https://doi.org/10.1016/j.comnet.2019.106984).
- 704 [2] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Aut-  
705 enrieth, V. Lopez, D. Lopez, Integrated SDN/NFV management and  
706 orchestration architecture for dynamic deployment of virtual SDN con-  
707 trol instances for virtual tenant networks, *IEEE/OSA Journal of Optical  
708 Communications and Networking* 7 (2015) B62–B70.
- 709 [3] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, X. Costa-Perez, How  
710 should i slice my network?: A multi-service empirical evaluation of re-  
711 source sharing efficiency, in: *Proceedings of the 24th Annual Interna-  
712 tional Conference on Mobile Computing and Networking, MobiCom '18*,  
713 ACM, New York, NY, USA, 2018, pp. 191–206. URL: [http://doi.acm.  
714 org/10.1145/3241539.3241567](http://doi.acm.org/10.1145/3241539.3241567). doi:10.1145/3241539.3241567.
- 715 [4] V. Sciancalepore, C. Mannweiler, F. Z. Yousaf, P. Serrano, M. Gra-  
716 maglia, J. Bradford, I. Labrador Pavn, A future-proof architecture for

- 717 management and orchestration of multi-domain nextgen networks, *IEEE*  
718 *Access* 7 (2019) 79216–79232. doi:10.1109/ACCESS.2019.2923364.
- 719 [5] S. Troia, A. Rodriguez, R. Alvizu, G. Maier, Senatus: An experimental  
720 sdn/nfv orchestrator, in: 2018 IEEE Conference on Network Function  
721 Virtualization and Software Defined Networks (NFV-SDN), 2018, pp.  
722 1–5. doi:10.1109/NFV-SDN.2018.8725690.
- 723 [6] L. Jorguseski, A. Pais, F. Gunnarsson, A. Centonza, C. Willcock, Self-  
724 organizing networks in 3GPP: standardization and future trends, *IEEE*  
725 *Communications Magazine* 52 (2014) 28–34. doi:10.1109/MCOM.2014.  
726 6979983.
- 727 [7] J. T. J. Penttinen, *Core Network*, Wiley, 2019, pp. 139–  
728 186. URL: <https://ieeexplore.ieee.org/document/8788396>.  
729 doi:10.1002/9781119275695.ch6.
- 730 [8] Open RAN Alliance, O-ran: Towards an open and smart ran, White  
731 Paper (2018).
- 732 [9] X. Foukas, M. K. Marina, K. Kontovasilis, Orion: Ran slicing for a  
733 flexible and cost-effective multi-service mobile network architecture, in:  
734 Proceedings of the 23rd annual international conference on mobile com-  
735 puting and networking, ACM, 2017, pp. 127–140.
- 736 [10] B. Ger, D. Jocha, R. Szab, J. Czentye, D. Haja, B. Nmeth, B. Sonkoly,  
737 M. Szalay, L. Toka, C. J. Bernardos Cano, L. M. Contreras Murillo, The  
738 orchestration in 5G exchange A multi-provider NFV framework for 5G  
739 services, in: 2017 IEEE Conference on Network Function Virtualization  
740 and Software Defined Networks (NFV-SDN), 2017, pp. 1–2.
- 741 [11] L. Ma, X. Wen, L. Wang, Z. Lu, R. Knopp, An SDN/NFV based  
742 framework for management and deployment of service based 5G core  
743 network, *China Communications* 15 (2018) 86–98.
- 744 [12] D. M. Gutierrez-Estevez, M. Gramaglia, A. de Domenico, N. di Pietro,  
745 S. Khatibi, K. Shah, D. Tsolkas, P. Arnold, P. Serrano, The path towards  
746 resource elasticity for 5g network architecture, in: 2018 IEEE Wireless  
747 Communications and Networking Conference Workshops (WCNCW),  
748 2018, pp. 214–219. doi:10.1109/WCNCW.2018.8369027.

- 749 [13] J. Ortin, C. Donato, P. Serrano, A. Banchs, Resource-on-demand  
750 schemes in 802.11 w lans with non-zero start-up times, IEEE Journal on  
751 Selected Areas in Communications 34 (2016) 3221–3233. doi:10.1109/  
752 JSAC.2016.2624158.
- 753 [14] 3GPP TS23.501, System Architecture for the 5G System,, Rel. 15, 2018.
- 754 [15] 3GPP TR28.801, telecommunication management;study on manage-  
755 ment and orchestration of network slicing for next generation network,  
756 Rel. 15, 2018.
- 757 [16] Xin Li, Chen Qian, A survey of network function placement, in: 2016  
758 13th IEEE Annual Consumer Communications Networking Conference  
759 (CCNC), 2016, pp. 948–953.
- 760 [17] A. Laghrissi, T. Taleb, A survey on the placement of virtual resources  
761 and virtual network functions, IEEE Communications Surveys Tutorials  
762 21 (2019) 1409–1434.
- 763 [18] H. Talebian, A. Gani, M. Sookhak, A. A. Abdelatif, A. Yousafzai,  
764 A. V. Vasilakos, F. R. Yu, Optimizing virtual machine placement  
765 in IaaS data centers: taxonomy, review and open issues (????).  
766 URL: <https://doi.org/10.1007/s10586-019-02954-w>. doi:10.1007/  
767 s10586-019-02954-w.
- 768 [19] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R.  
769 Lyu, R. Buyya, Cloud service reliability enhancement via virtual ma-  
770 chine placement optimization, IEEE Transactions on Services Comput-  
771 ing 10 (2017) 902–913.
- 772 [20] OSM Release FIVE Technical Overview, [https://osm.etsi.org/  
773 images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf](https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFIVE-FINAL.pdf),  
774 2019. Online; accessed Apr. 2020.
- 775 [21] ONAP Architecture Overview whitepaper, [https://www.onap.  
776 org/wp-content/uploads/sites/20/2019/07/ONAP\\_CaseSolution\\_  
777 Architecture\\_062519.pdf](https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf), 2019. Online; accessed Apr. 2020.
- 778 [22] OSM Information Model, [https://osm.etsi.org/wikipub/index.  
779 php/OSM\\_Information\\_Model](https://osm.etsi.org/wikipub/index.php/OSM_Information_Model), 2019. Online; accessed Dec. 2019.

- 780 [23] M. E. Elsaid, C. Meinel, Live migration impact on virtual datacenter  
781 performance: Vmware vmotion based study, in: 2014 International  
782 Conference on Future Internet of Things and Cloud, 2014, pp. 216–221.  
783 doi:10.1109/FiCloud.2014.42.
- 784 [24] F. Zhang, G. Liu, X. Fu, R. Yahyapour, A survey on virtual machine  
785 migration: Challenges, techniques, and open issues, IEEE Communica-  
786 tions Surveys Tutorials 20 (2018) 1206–1243. doi:10.1109/COMST.2018.  
787 2794881.
- 788 [25] Openstack Docs live-migrate instances, [https://docs.openstack.  
789 org/nova/pike/admin/live-migration-usage.html](https://docs.openstack.org/nova/pike/admin/live-migration-usage.html), 2019. Accessed:  
790 Dec 2019.
- 791 [26] VMware vSphere vMotion architecture, performance and best practices  
792 in vmware vsphere 5: Performance study, Technical White Paper, 2011,  
793 2019. Online; accessed Dec. 2019.
- 794 [27] S. Nadgowda, S. Suneja, N. Bila, C. Isci, Voyager: Complete Con-  
795 tainer State Migration, in: 2017 IEEE 37th International Confer-  
796 ence on Distributed Computing Systems (ICDCS), 2017, pp. 2137–2142.  
797 doi:10.1109/ICDCS.2017.91.
- 798 [28] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gaza-  
799 gnaire, S. Smith, S. Hand, J. Crowcroft, Unikernels: Library op-  
800 erating systems for the cloud, SIGPLAN Not. 48 (2013) 461–472.  
801 URL: <http://doi.acm.org/10.1145/2499368.2451167>. doi:10.1145/  
802 2499368.2451167.
- 803 [29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl,  
804 J. Khalid, S. Das, A. Akella, Opennf: Enabling innovation in net-  
805 work function control, in: Proceedings of the 2014 ACM Conference  
806 on SIGCOMM, SIGCOMM '14, ACM, New York, NY, USA, 2014,  
807 pp. 163–174. URL: <http://doi.acm.org/10.1145/2619239.2626313>.  
808 doi:10.1145/2619239.2626313.
- 809 [30] S. Rajagopalan, D. Williams, H. Jamjoom, A. Warfield, Split/merge:  
810 System support for elastic execution in virtual middleboxes, in:  
811 Presented as part of the 10th USENIX Symposium on Networked  
812 Systems Design and Implementation (NSDI 13), USENIX, Lombard,

- 813 IL, 2013, pp. 227–240. URL: [`https://www.usenix.org/conference/`](https://www.usenix.org/conference/)  
814 `nsdi13/technical-sessions/presentation/rajagopalan`.
- 815 [31] 3GPP TS23.502, Procedures for the 5G System (5GS); stage 2 (release  
816 16),, Rel. 16, 2020.
- 817 [32] X. Feng, J. Tang, X. Luo, Y. Jin, A performance study of live vm  
818 migration technologies: Vmotion vs xenmotion, in: 2011 Asia Commu-  
819 nications and Photonics Conference and Exhibition (ACP), 2011, pp.  
820 1–6. doi:10.1117/12.905512.
- 821 [33] I. Gomez-Miguel, A. Garcia-Saavedra, P. D. Sutton, P. Serrano,  
822 C. Cano, D. J. Leith, srslte: An open-source platform for lte evolu-  
823 tion and experimentation, in: Proceedings of the Tenth ACM Inter-  
824 national Workshop on Wireless Network Testbeds, Experimental Eval-  
825 uation, and Characterization, WiNTECH '16, ACM, New York, NY,  
826 USA, 2016, pp. 25–32. URL: [`http://doi.acm.org/10.1145/2980159`](http://doi.acm.org/10.1145/2980159).  
827 2980163. doi:10.1145/2980159.2980163.
- 828 [34] G. Garcia-Aviles, M. Gramaglia, P. Serrano, A. Banchs, POSENS:  
829 A Practical Open Source Solution for End-to-End Network Slicing,  
830 IEEE Wireless Communications 25 (2018) 30–37. URL: [`https://`](https://ieeexplore.ieee.org/document/8524891/)  
831 `ieeexplore.ieee.org/document/8524891/`. doi:10.1109/MWC.2018.  
832 1800050.
- 833 [35] ETSI, network functions virtualisation (nfv) release 3; evolution and  
834 ecosystem; report on network slicing support with etsi nfv architecture  
835 framework, 2017.
- 836 [36] 3GPP TS29.510, 5G System; Network function repository services;  
837 Stage 3 (Release 15),, Rel. 15, 2020.
- 838 [37] ONOS Project, [`https://onosproject.org`](https://onosproject.org), 2019. Online; accessed Dec.  
839 2019.
- 840 [38] Cloud-native network functions, Cisco White Paper, 2018. URL:  
841 [`https://www.cisco.com/c/en/us/solutions/service-provider/`](https://www.cisco.com/c/en/us/solutions/service-provider/industry/cable/cloud-native-network-functions.html)  
842 `industry/cable/cloud-native-network-functions.html`.
- 843 [39] 3GPP TS28.541, Management and orchestration; 5G Network Resource  
844 Model (NRM); Stage 2 and stage 3,, Rel. 15, 2018.

845 [40] 5G-CORAL, Refined design of 5G-CORAL orchestration and control  
846 system and future directions, D3.2, 2019.