D4.2: Implementation and validation of control framework

Revision: v.1.0

<table>
<thead>
<tr>
<th>Work package</th>
<th>WP4</th>
<th>Task</th>
<th>Task 4.2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Due date</td>
<td>31/08/2017</td>
<td>Submission date</td>
<td>31/08/2017</td>
</tr>
<tr>
<td>Editor</td>
<td>TCD</td>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Daniel Fernandes Macedo (UFMG), Magnos Martinello (UNIVBRIS), Johann M. Marquez-Barja, Luiz DaSilva (TCD)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Abstract

This deliverable reports on the implementation of FUTEBOL control framework within the different

Keywords

Control Framework; Wireless Testbed; Optical Testbed; Wireless-Optical Convergence
Document Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of change</th>
<th>List of contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>18/06/2017</td>
<td>Table of Contents</td>
<td>TCD</td>
</tr>
<tr>
<td>V0.5</td>
<td>10/08/2017</td>
<td>Draft version</td>
<td>All contributors</td>
</tr>
<tr>
<td>V0.6</td>
<td>25/08/2017</td>
<td>Reviewers’ version</td>
<td>Reviewers</td>
</tr>
<tr>
<td>V1.0</td>
<td>31/08/2017</td>
<td>Final Version</td>
<td>TCD, UNIVBRIS, VTT, UFRGS, UFMG, UFES</td>
</tr>
</tbody>
</table>

Project co-funded by the European Commission in the H2020 Programme

<table>
<thead>
<tr>
<th>Nature of the deliverable:</th>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU Public</td>
<td>✓</td>
</tr>
<tr>
<td>PP Restricted to other programme participants (including the Commission Services)</td>
<td></td>
</tr>
<tr>
<td>RE Restricted to bodies determined by the FUTEBOL project</td>
<td></td>
</tr>
<tr>
<td>CO Confidential to FUTEBOL project and Commission Services</td>
<td></td>
</tr>
</tbody>
</table>

Copyright notice

© 2016 - 2019 FUTEBOL Consortium

*R: report, P: prototype, D: demonstrator, O: other
EXECUTIVE SUMMARY

This document describes the design and implementation status of the FUTEBOL Control Framework (CF). The objective of the control framework is to enable experimentation at the boundary of wireless and optical networks by facilitating the composition and control of experiments that combine wireless and optical heterogeneous resources. The framework is designed to provision optical and wireless resources among the different testbeds of FUTEBOL and to introduce mechanisms to orchestrate said resources.

By firstly giving a general description of the control framework, this document provides the initial motivation of the control framework, its major objectives and its general architecture. This includes a detailed description of the mechanisms for provisioning resources and for orchestrating different resources (such as virtual functions, wireless or optical resources).

For the experimentation control, orchestration functionalities such as Virtual Network Function (VNF) migration, network orchestration and mechanisms as Service Function Chaining (SFC) are proposed to enable FUTEBOL users to orchestrate the experiments. Users are able to control the resources through a set of control units. The network orchestration is the main component of the CF and has been implemented by making use of control units (e.g. SDN controller) and an experiment orchestrator that uses the controller's northbound API to enable network programmability. The experiment orchestrator provides a single orchestration interface for the experimenter that enables the converged control of the experiment resources. It interacts with the testbed infrastructure manager (i.e. the Aggregate Manager), gathering information about the infrastructure to perform orchestration decisions.

After giving a general description of the design and objectives of the control framework, the specific implementations of the multiple partners are described. This includes Brazilian and European testbeds, with various types of SDR, IoT, WiFi, LTE, SDN and optical resources.
# TABLE OF CONTENTS

EXECUTIVE SUMMARY ........................................................................................................... 3
TABLE OF CONTENTS ............................................................................................................... 4
LIST OF FIGURES .................................................................................................................... 6
LIST OF TABLES ....................................................................................................................... 7
ABBREVIATIONS ..................................................................................................................... 8

1 Introduction ............................................................................................................................ 11

2 FUTUREBOL Control Framework Implementation ............................................................. 13
   2.1 FUTUREBOL Control Framework Experiment Provisioning ............................................. 14
      2.1.1 Aggregate Manager ................................................................................................ 14
      2.1.2 NFV and Service Provisioning .............................................................................. 17
   2.2 FUTUREBOL Control Framework Experiment Orchestration .......................................... 19
      Wireless Orchestration ..................................................................................................... 19
      Optical/Packet Orchestration ......................................................................................... 24
      NFV and service orchestration ..................................................................................... 24
   2.3 Experimentation and convergence control ...................................................................... 26

3 FUTUREBOL CF deployment in TCD testbed .................................................................... 29
   3.1 Testbed description ......................................................................................................... 29
   3.2 Aggregate Manager implementation ............................................................................ 30
   3.3 NFV Manager implementation .................................................................................... 31
   3.4 Validation of the experiment control and service orchestration .................................... 31

4 FUTUREBOL CF deployment in UNIVBRIS testbed ............................................................. 33
   4.1 Testbed description ......................................................................................................... 33
   4.2 Aggregate Manager implementation ............................................................................ 33
   4.3 NFV Manager implementation .................................................................................... 34
   4.4 Validation of the experiment control and service orchestration .................................... 35
      4.4.1 Convergent slice allocation .................................................................................. 36

5 FUTUREBOL CF deployment in VTT testbed .................................................................. 38
   5.1 Testbed description ......................................................................................................... 38
   5.2 Aggregate Manager implementation ............................................................................ 38
   5.3 NFV Manager implementation .................................................................................... 38
   5.4 Validation of the experiment control and service orchestration .................................... 38

6 FUTUREBOL CF deployment in UFRGS testbed ................................................................. 40
   6.1 Testbed description ......................................................................................................... 40
   6.2 Aggregate Manager implementation ............................................................................ 41
6.3 NFV Manager implementation ........................................................................................................ 41
6.4 Validation of the experiment control and service orchestration ................................................. 43
7 FUTEBOL CF deployment in UFMG testbed ................................................................................... 44
  7.1 Testbed description ....................................................................................................................... 44
  7.2 Aggregate Manager implementation .......................................................................................... 44
  7.3 NFV Manager implementation .................................................................................................. 45
  7.4 Validation of the experiment control and service orchestration .................................................. 45
8 FUTEBOL CF deployment in UFES testbed .................................................................................... 48
  8.1 Testbed description ..................................................................................................................... 48
  8.2 Aggregate Manager implementation .......................................................................................... 49
    8.2.1 VM provisioning and LAN connectivity .................................................................................. 50
    8.2.2 Intelligent Space provisioning ............................................................................................... 53
    8.2.3 SDN switches provisioning .................................................................................................... 55
  8.3 NFV Manager implementation .................................................................................................... 56
  8.4 Validation of the experiment control and Service orchestration .................................................. 57
    8.4.1 Showcase description ............................................................................................................. 57
    8.4.2 Experiment Control Application ............................................................................................ 59
    8.4.4 Experiment SDN Controller ................................................................................................... 60
    8.4.5 Experiment Orchestrator ......................................................................................................... 60
    8.4.6 Service Orchestration ............................................................................................................. 65
9 Conclusions ...................................................................................................................................... 69
References ........................................................................................................................................... 70
LIST OF FIGURES

Figure 1: FUTEBOL Control Framework (CF) Architecture ................................................................. 13
Figure 2: Overall architecture of IoT control and provisioning .............................................................. 15
Figure 3: Extensions of OpenFlow Protocol to support control of flexi grid WDM ................................. 17
Figure 4: Overall Architecture of COPA .................................................................................................. 18
Figure 5: Sample of COPA’s Web-based GUI ....................................................................................... 19
Figure 6: Architecture of SDR controller ............................................................................................... 20
Figure 7: Example of access to COPA’s REST API by the experimenter’s own orchestration script ....... 25
Figure 8: Reservation of Optical and Wireless Resources ....................................................................... 27
Figure 9: Iris Testbed Architecture ....................................................................................................... 29
Figure 10: Overall architecture of the CBTM system ............................................................................. 31
Figure 11: UNIVBRIS testbed ............................................................................................................... 33
Figure 12: Convergent Orchestrator at Bristol Testbed ......................................................................... 35
Figure 13: Container management based on the integration of CBTM and COPA ................................. 42
Figure 14: Example of experiment with network functions and services deployed across testbeds ....  43
Figure 15: Ensuring the throughput in the wireless link using Ethanol .................................................. 46
Figure 16: Network throughput without packet prioritization ............................................................... 47
Figure 17: UFES testbed ......................................................................................................................... 48
Figure 18: O2CMF Architecture ............................................................................................................ 49
Figure 19: RSPEC for VMs and LAN connectivity .................................................................................. 51
Figure 20: Openstack Networking (Adapted from: OpenStack Kilo Networking Guide) ....................... 53
Figure 21: RSPEC for Intelligent Space ................................................................................................. 55
Figure 22: RSPEC for SDN switches ..................................................................................................... 56
Figure 23: RSPEC for VNFs .................................................................................................................. 57
Figure 24: Intelligent Space Application Architecture ............................................................................ 58
Figure 25: Intelligent Space Application Message Flow ......................................................................... 59
Figure 26: Horizontal Scaling Example .................................................................................................. 67
Figure 27: SFC Example ......................................................................................................................... 68
LIST OF TABLES

Table 1: IEEE802.15.4 Wireless Parameters ................................................................. 21
Table 2: Summary of northbound and southbound APIs ............................................. 27
## ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>Aggregate Manager</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AT</td>
<td>Attention</td>
</tr>
<tr>
<td>ANATEL</td>
<td>Agência Nacional de Telecomunicações (Brazil)</td>
</tr>
<tr>
<td>BBU</td>
<td>Base-Band Unit (BBU)</td>
</tr>
<tr>
<td>CBTM</td>
<td>Cloud Based Testbed Manager</td>
</tr>
<tr>
<td>CEPT</td>
<td>European Conference of Postal and Telecommunications Administrations</td>
</tr>
<tr>
<td>CF</td>
<td>Control Framework</td>
</tr>
<tr>
<td>CH</td>
<td>Clearinghouse</td>
</tr>
<tr>
<td>C-RAN</td>
<td>Cloud Radio Access Network</td>
</tr>
<tr>
<td>DC</td>
<td>Data Center</td>
</tr>
<tr>
<td>DWDM</td>
<td>Dense Wavelength Division Multiplexing</td>
</tr>
<tr>
<td>Ex.x</td>
<td>FUTEBOL experiment x.x</td>
</tr>
<tr>
<td>E2E</td>
<td>End to End connectivity</td>
</tr>
<tr>
<td>EPC</td>
<td>Evolved Packet Core</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standard Institute</td>
</tr>
<tr>
<td>FIRE</td>
<td>Future Internet and Research Experimentation</td>
</tr>
<tr>
<td>FTTH</td>
<td>Fiber-to-the-Home</td>
</tr>
<tr>
<td>FOAM</td>
<td>OpenFlow Aggregate Manager</td>
</tr>
<tr>
<td>FUTEBOL</td>
<td>Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory</td>
</tr>
<tr>
<td>GENI</td>
<td>Global Environment for Network Innovations</td>
</tr>
<tr>
<td>H2M</td>
<td>Human to Machine</td>
</tr>
<tr>
<td>IMC</td>
<td>Intel Mobile Communications GmbH (FUTEBOL’s partner)</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
</tbody>
</table>
IT Institute de Telecomunicações – Aveiro (FUTEBOL’s partner)
LSA Licensed Shared Access
LTE Long Term Evolution
MAC Medium Access Control
MCC Mobile Cloud Computing
MDC Micro Data Center
M2M Machine to Machine
MNO Mobile Network Operator
NFV Network Functions Virtualization
OCF OFELIA Control Framework
OF OpenFlow
OMF Control and Management Framework
PMSE Programme Making and Special Events equipment
PNF Physical Network Function
PON Passive Optical Network
QoE Quality of Experience
QoS Quality of Service
RAN Radio Access Network
RAU Remote Antenna Units
RF Radio Frequency
ROADM Reconfigurable Optical Add-drop Multiplexer
RoF Radio over Fiber
RRH Remote Radio Heads
SDN Software Defined Networks
SDR Software Defined Radio
SFA Slice-based Federation Architecture
TCD  Trinity College Dublin (FUTEBOL partner)
TCP  Transmission Control Protocol
UDP  User Datagram Protocol
UFC  Universidade Federal do Ceará (FUTEBOL partner)
UFES  Universidade Federal do Espírito Santo (FUTEBOL partner)
UFMG  Universidade Federal de Minas Gerais (FUTEBOL partner)
UFRGS  Universidade Federal do Rio Grande do Sul (FUTEBOL partner)
UNICAMP  Universidade Estadual de Campinas (FUTEBOL partner)
UNIVBRIS  University of Bristol (FUTEBOL partner)
USRP  Universal Software Radio Peripheral
VM  Virtual Machine
VNF  Virtual Network Function
VTAM  Virtualization Aggregate Manager
VTT  Teknologian Tutkimuskeskus VTT Oy (FUTEBOL partner)
WAN  Wide Area Network
WSN  Wireless Sensor Network
WSS  Wavelength Selective Switch
1 INTRODUCTION

The FUTEBOL project is establishing a set of federated distributed optical and wireless network testbeds under a common control framework. The ability to break the barriers between isolated wireless and optical research platforms by using different deployment scenarios, in a coordinated way, is one of the goals of the FUTEBOL project. The FUTEBOL control framework enables experimentation at the boundary of wireless and optical networks by facilitating the composition and control of experiments that combine wireless and optical heterogeneous resources. The focus of the FUTEBOL platform is on enabling experimental research on wireless-optical network convergence.

The FUTEBOL Control Framework (CF) is the set of tools and functionalities that will support the operation of the FUTEBOL experimental platform. The FUTEBOL CF will enable FUTEBOL users to reserve and to allocate resources to run their experiments using heterogeneous wireless and optical resources in the different FUTEBOL testbeds. Our vision of the CF involves the adoption and integration of SDN and NFV over the wireless and optical domains.

The focus of this document is on how to implement the FUTEBOL CF in order to support the concepts and functionalities defined in the deliverable D4.1. A fundamental concept of the CF design is the converged experimentation slice that allows the experimenters to allocate converged resources. In a general sense, the converged resources are manageable units, with specific capabilities that can be used to deliver a service. A network slice is composed by a collection of resources that, appropriately combined, meet the requirements of the use case/experiment. For instance, an experimentation slice can be composed by i) a network slice (wireless and/or optical and/or packet switched networks), ii) computing resources (e.g. virtual machines, containers or physical resources); and iii) virtualized network functions (VNFs) that provide functional blocks with specific network capabilities (e.g. BBU processing; image processing; controller).

For the experimentation control, orchestration functionalities such as VNF migration, and mechanisms such as service function chaining (SFC) are proposed to enable FUTEBOL users to orchestrate the experiments. Users are able to control the resources through a set of control units. The network orchestration is the main component of the CF and has been implemented by making use of control units (e.g. SDN controller) and an experiment orchestrator that uses controller's northbound API to enable network programmability. The experiment orchestrator provides an orchestration interface for the experimenter that enables the converged control of the experiment resources. It interacts with the testbed infrastructure manager (i.e. the Aggregate Manager) gathering information about the infrastructure to perform orchestration decisions.

One of the fundamental differences between the existing experimentation CFs and the FUTEBOL CF is the convergent network programmability. The FUTEBOL CF has advanced network programmability by enriching the range of physical parameters that are configurable and have been exposed in a common interface from a large variety of devices (switches, optical devices, IoT devices, cameras, wireless APs, robots, SDRs). Thus, an experimenter can develop an application that is able to control resources through the following actions:

- To map specific packet flows (identified by VLANs tags) to optical wavelengths (lambdas) in SDN-enabled optical equipment;
- To control wireless SDR parameters, such as sampling rate;
- To set wireless AP parameters such as transmission power and channel of operation;
- To control wireless handover based e.g. on the mobile node’s (a robot, or a user equipment) location;
To set camera (IoT device) parameters such as number of frames per second, color system, and resolution;
To change IoT device parameters, such as the personal area network that they belong (PAN ID);
To retrieve IoT sensor devices information, such as luminosity and temperature measurements;
To retrieve monitoring and measurement information from the network resources;
To perform horizontal scaling of VNF components (increasing or decreasing the number of instances).

Also, VNF migration is offered to allow the placement of the VNFs within the experimentation slice. SFC is an enabler mechanism to allow the creation of an ordered set of service functions and subsequent steering of traffic through them. The service chain is a set of connected functional blocks offering an integrated SDN-NFV mechanism to facilitate the experiment deployment and the running of the experiment.
2 FUTEBOL Control Framework Implementation

This section presents an overview of the design of the FUTEBOL CF architecture. The goal of this architecture is to support various experimental use cases in FUTEBOL and to enable FUTEBOL testbed experimenters to reserve and run experiments for testing innovative ideas and technologies dealing with wireless/optical convergence.

The FUTEBOL CF architecture is designed to be flexible and programmable. The components of the architecture are described in boxes within separate layers as illustrated in Figure 1. The layers are sufficiently independent to simplify reasoning about the functionalities provided between layers. The separation of the functionalities and components distinguishes between: (i) service layer; (ii) experiment control layer; (iii) testbed management layer, represented vertically; (iv) virtualization layer and; (v) converged physical infrastructure layer (including optical, wireless, and data center resources).

Network softwarization and programmability form the basis for the FUTEBOL CF design and impact the logical and functional view of the architecture. It is worth noting that the purpose of NFV concept in the FUTEBOL CF is not to be fully supported as a complete NFV solution, as defined by ETSI. Rather, our choice of implementation for NFV is to orchestrate distributed and heterogeneous federated testbeds through a new orchestrator, named COPA but also to bring the network closer to the cloud by an extended Openstack framework.

Network applications can be developed by using the service layer implemented in the FUTEBOL CF. The service layer provides a service catalogue (i.e. a set of VNFs and user applications) to support a given use case (or experiment) and using the experiment control layer to orchestrate it. The service catalogue will be built by extending existing tools for managing the testbed resources, for instance VTAM, CBTM, or OpenStack, depending on the testbed resources and purpose.
The testbed management layer acts as a mediator between requests from the federation (Fed4FIRE via SFA calls) interacting with the management components of physical resources. This layer directly accesses the database that stores information about the reserved resources. That information is used to provide a slice and access to its resources. The Aggregate Managers (AMs) are responsible to deal with SFA calls so that the users can reserve resources across the federated testbeds, using a common set of credentials.

The experiment control layer allows the integrated manipulation of resources and promotes the decoupling between the mechanisms that enable federation and the technologies that manage the resources. An orchestrator is the element that interacts with the controllers of physical resources and uses the orchestration functionalities. The virtualization layer contains resource controllers which are responsible for controlling the physical resources.

There are two main stages to an experiment that make use of the FUTEBOL CF: a) the experiment provisioning stage, where resources are allocated and slices are provisioned, and b) the experiment orchestration stage, where the experiment resources adapt to network conditions. The following subsections elaborate on role of the FUTEBOL control framework during both stages.

### 2.1 FUTEBOL Control Framework Experiment Provisioning

In this subsection we will describe the slicing and provisioning of experiments from the FUTEBOL control framework. This will include a brief description of the assignment of resources through the testbed’s federation framework and the allocation of virtual functions, which can be Virtualized Network Functions (VNFs) or different network services, such as applications.

#### 2.1.1 Aggregate Manager

**SDR Resources Provisioning**

In Software Defined Radio (SDR), baseband processing is done by normal CPUs using standard programming languages such as C/C++. The IQ samples are transmitted to and from a frontend device, which is responsible for the up- or down-conversion of the wireless signal and transmission over the air.

This means that to provision SDR resources to experimenters, it is necessary to provide processing capabilities for experimenters and allocate a radio frontend to the processing unit. To do this, Virtual Machines are used that have a pass-through connection to the frontend, either through USB or Ethernet.

It is then the AM’s responsibility to parse the RSpecs and interact with a virtualization library to create VMs and add the user’s credentials to the VMs, so that experimenters can SSH into the VMs. Different SDR frameworks are provided by each of the testbeds, in the form of virtual machine images, so that users can choose the SDR framework that suits their needs.

The specific implementations of the testbed’s AM are elaborated in sections 3, 6 and 7, where the specific implementation details of SDR supporting testbeds are described.

**IoT Resources Provisioning**

In IoT control, a proxy node is used to give the experimenters access to the IoT devices, as depicted in Figure 2. This proxy is either a VM or a container, allocated in a PC with a physical connection to the device. For example, an Advanticsys node is connected to a PC via USB. The commands of the experimenters are sent to the proxy, which then verifies whether the experimenter is authorized to perform them. If the command is authorized, it is forwarded to the IoT device. The proxy also translates the commands into a machine-readable command (e.g. an AT command for an XBee radio, a nl80211
call for WiFi). Hence, the user must book the IoT resources, as well as a VM containing an image of the control software.

![Overall architecture of IoT control and provisioning](image)

*Figure 2: Overall architecture of IoT control and provisioning*

Some of the devices, such as Raspberry Pi nodes, support virtualization in the form of containers. These nodes are connected to the testbed infrastructure via Ethernet, and the aggregate manager creates a user for the experimenter in that device. Only the experimenter that booked that device will be able to login with an SSH certificate. The control unit for those IoT devices comprises three phases of maturity. In the first phase, IoT nodes are directly accessed by experimenters with full access rights to their systems. For example, an experimenter allocating a Raspberry Pi will receive permissions and access credentials of super user through a secure shell session, being able to exploit all capabilities of the IoT node. In the second phase, the direct access is removed, and the experimenter receives a Virtual Machine (VM) or a container having access to reprogramming routines able to indirectly reprogram the IoT nodes. In the third phase, VMs with a container system and a catalog containing previously developed containers are introduced to simplify the control and deployment of experiments with IoT nodes for non-specialist experimenters.

The first phase of the control unit for IoT was already implemented during the first year of the project and it is currently operational.

Besides the wireless IoT nodes of UFRGS and UFMG, the UFES testbed also provides cameras that are coupled with special VMs, called agents, that receive SDN commands from the SDN Controller and translate them into native configuration commands to the cameras. The parameters exposed to the users are the image resolution and the image acquisition frequency. In addition, the experimenter can reserve a mobile robot, but no parameter is directly exposed to the experimenter, because the robot platform provides only basic wireless communication capabilities. The robot movement can be controlled, in the orchestration phase, by a set of processing functions hosted in the datacenter.
**LTE Shared Access Facilities Provisioning**

The experimenter reserves the LTE hardware through the federated mechanism. The aggregate manager maintains a database of the LTE equipment reservations and the available LTE equipment will be listed in advertisement RSpec. Due to the high power usage of commercial LTE equipment, the base stations might be turned off sometimes to save energy, affecting the number of available LTE equipment shown in the Fed4FIRE portal and JFed tool.

An SFA client (such as jFed) creates allocation RSpecs containing the reservation information and parameters for the LTE equipment (described in Chapter 5) and possibly for the software simulating extra LTE base stations. The reserved hardware is not shared with other experimenters during the experimentation, therefore the maximum duration of a reservation might be restricted. The plan is to provide unlimited number of simulated LTE base station entities without real hardware to allow testing large scenarios with only part or none of the base stations being real hardware.

The experimenter will be given access to a VM with instances of LSA related software, such as the LSA repository and LSA controller, and based on the request RSpec, the VM has a limited interface to control each reserved LTE equipment.

**Optical/Packet Provisioning**

The optical/packet provisioning is implemented by extending the Openflow protocol to support the control of the optical physical layer based on the ITU-T G 694.1 recommendation. Figure 3 shows the physical programmable parameters extended by the HPN network group at Bristol to support the state-of-the-art flexi grid WDM equipment. Actually, the implementation under progress allows an experimenter to specify i) the central frequency ranging from $193.1 \text{THz} + n \times 0.00625$ and ii) bandwidth slot $12.5 \text{GHz} \times m$. It is worth pointing out that $n, m$ are integers that define the range and their granularity depending on devices and technology. In FUTEBOL, the finest granularity that will be supported due to optical technology sitting in the optical federated devices is at the wavelength level.

In the Bristol testbed, all the federated switches (DWDM Adva and Polatis equipment) are SDN enabled switches that support the extended OpenFlow messages (e.g. `flowmod`). OpenFlow enabled switches allow the experimenter to specify VLAN tag(s) that can be mapped into `wavelengths` in the optical domain. The configuration is carried out by an ODL controller. In terms of convergent network architecture, it means that the southbound interface is a common and uniform interface based on the OpenFlow protocol.
2.1.2 NFV and Service Provisioning

In FUTEBOL, we wish to support network and service virtual functions through the control framework. This is done by providing virtual machines or containers with pre-installed networking software, such as applications, software switches, firewalls, etc.

Previously developed control frameworks, such as CBTM and VTAM, although they provide some functionalities that allow testbeds to deploy VMs with different disk images, they do not provide e.g. isolation between experiments, unless the experimenter configures that himself, assigning a VLAN to his VMs interfaces and setting the network accordingly. Thus, a framework is under development, in which computing resources, named virtual infrastructure (network connectivity + virtual machines), are managed by the OpenStack platform. This framework has been developed by UFES to research specific cloud functionalities enabled by OpenStack, such as horizontal scaling and virtual infrastructure allocation. Other testbeds that already have other control frameworks deployed are not required to change their deployments.

In UFES framework, the experimenter uses the jFed tool to define a RSpec that specifies the virtual function(s) needed. As a result of the provisioning, the experimenter receives SSH access to the VMs in which each virtual function was deployed. The use of OpenStack enables testbeds to provide advanced NFV functionalities, such as definition of TOSCA policies, SFC, isolation, and L2 connectivity between virtual functions. More details can be found on Section 8, which describes the specific implementation of this framework in the UFES testbed.

In FUTEBOL, in addition to the new UFES control framework, we have also added mechanisms to deploy VNFs and other services as containers within the experimenter's VMs. In the testbeds that support this functionality, the virtual functions will be provisioned either by Docker or LXC/LXD, depending on the testbed’s implementation.
This new orchestration and provisioning tool for containers was created to facilitate the migration of virtual functions across testbeds. Live-migrating full virtual machines is only possible if they use a common hypervisor; however, we do not wish to force all testbeds to use the same hypervisor on their physical servers and different testbeds are already using different hypervisors for legacy reasons. VMs can then provide a homogenous environment across testbeds and using containers to implement the virtual functions reduces the extra virtualization overhead for this second layer of abstraction.

To deploy and manage these containers, a new container orchestration and provisioning tool was developed in FUTEBOL: COPA (Container Orchestration and Provisioning Architecture). An illustration of FUTEBOL’s COPA can be seen in Figure 4. There, we can see that there are three main components in COPA: the COPA orchestrator, the Service Catalogue and the Container Pool VM. The code for this software will be available to the public under an adequate open-source license and appropriate documentation will be available in the project’s repository.

The COPA Orchestrator is responsible for provisioning the containers in the container pool and providing a GUI platform for the user to interact with. This GUI will take form of a simple website hosted in an Apache server within the VM. To access the website, the user will have to port forward one of its local ports over SSH. A sample of this GUI is shown in Figure 5.
The COPA Orchestrator will translate the user’s requests to LXD’s or Docker’s REST API to provision, delete, migrate, and provide terminal access to the containers. To be able to do this, the orchestrator will receive a list of the IPs of the experiment’s Container Pools. Besides this list, the orchestrator will also push the experimenter’s credentials to the containers, enabling SSH access to the containers, if they are connected to the same subnet as the VMs.

Each testbed that provides this functionality will maintain a Service Catalogue that lists virtual functions available. These will include services such as video servers and clients or wireless baseband processing units. Each testbed must prepare a list of container images that will be copied into the COPA orchestrator by the AM. The virtual functions provided by each testbed will be described in details in the subsections related to the specific implementations of each testbed.

Finally, the Container Pool is a VM with a disk image that has a container system pre-installed, such as Docker or LXD. This container pool will host the actual virtual functions and services.

### 2.2 FUTEBOL Control Framework Experiment Orchestration

#### Wireless Orchestration

**SDR Control Units**

Software Defined Radio strives for flexibility, making it possible to change all the parameters in a physical transmission. This makes it challenging to enable control over SDR while not limiting its flexibility. On top of this, SDR applications are extremely computationally intensive, due to their real-time requirements.
The FUTEBOL CF must enable experimenters to quickly build their own controllers for their own SDR applications. We wish to do this without limiting the flexibility of SDR, introducing minimal dependencies for the SDR applications and keeping a high performance, so that parameters can be changed quickly.

In Figure 6, we can see the architecture of the SDR Controller (SDR-C) and SDR Agent (SDR-A). There, it can be seen that a controller program will communicate with an agent on the SDR application to control and change parameters.

To pass messages between the SDR-C and the SDR-A we will use the messaging library 0MQ\(^1\). Using 0MQ has the following advantages:

- **Extra features** over raw sockets. Publish/Subscribe patterns, being able run the program whether the controller is running or not, etc., makes the life of the developer easier without sacrificing too much performance.
- **Lightweight** compared to XML-RPC or REST. XML-RPC and REST work over HTTP; which means, to use those communication protocols, requires a lightweight HTTP server implementation in the SDR application. We consider this to be a heavy and unnecessary dependency.
- **C/C++ and Python bindings**. The C/C++ bindings make it easier to integrate agents directly into the SDR application, as opposed to running an agent in a separate program. This is important to maintain high-performance in parameter reconfiguration. The Python bindings allow us to develop the controller in Python, where there are no real-time computational requirements.

We are implementing an example controller and agent that can be used to control a PDSCH LTE implementation of srsLTE. The controlled parameters will include carrier frequency, sampling rate (i.e., the number of physical resource blocks) and gain.

---

\(^1\) [http://zeromq.org/](http://zeromq.org/)
It is also possible to integrate the controller with GNU Radio, as there is a 0MQ source and sink block. If experimenters wish to use it, it is their responsibility to implement the integration of their own flowgraph and the 0MQ sources and sinks.

**IoT Control Units**

With the development of the control framework getting mature, the wireless control of the IoT nodes’ wireless interfaces may be enhanced. The wireless interfaces of the IoT nodes may have their physical parameters, such as maximum transmission power and channel in use, exposed to be readjusted remotely. Since each testbed provides different types of IoT resources, in this section we present how to control the IoT devices in each testbed.

**IoT control in UFRGS:** As the IoT nodes are resource-constrained devices, they are not able to handle automatic, elaborate, or complex control routines, otherwise compromising their performance. In this case, XBees are used as IEEE 802.15.4 modules connected to the IoT nodes through a USB or Arduino shield at the UFRGS’ testbed. The change of parameters is made by sending AT commands (attention commands) to the XBee. In order to send commands, the user must SSH into the device, and then open the XBee radio serial port. After that, it is necessary to enter the AT mode using the reserved word ‘+++’ and wait for the XBee to get ready with an ‘OK’ reply. An AT command can be issued according to the following structure: ‘AT’+ID+value. For example, to change the channel to 1A, a command ‘ATCH1A’ must be sent to the XBee module. A sequence of messages must be issued after the parameter is changed to commit the setup. This sequence is composed of ‘ATWR’, which writes the changes in the XBee's memory, followed by an ‘ATAC’ to apply the changes, and, finally, an ‘ATCN’ to exit the AT mode. The following table presents the list of wireless parameters for the XBee radio used in UFRGS.

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH</td>
<td>Channel</td>
<td>0x0B - 0x1A</td>
<td>IEEE 802.15.4 channel number</td>
</tr>
<tr>
<td>ID</td>
<td>PAN ID</td>
<td>0x0 - 0xFFFF</td>
<td>Personal Area Network ID. 0xFFFF sends to all PANs</td>
</tr>
<tr>
<td>DH</td>
<td>Destination Address High</td>
<td>0x0 - 0xFFFFFFFF</td>
<td>Set the DH register to zero and DL 0xFFFF</td>
</tr>
<tr>
<td>DL</td>
<td>Destination Address Low</td>
<td>0x0 - 0xFFFFFFFF</td>
<td>to broadcast to the PAN</td>
</tr>
<tr>
<td>MY</td>
<td>16-bit Source Address</td>
<td>0x0 - 0xFFFF</td>
<td>16-bit source address for the modem</td>
</tr>
<tr>
<td>MM</td>
<td>MAC Mode</td>
<td>0 - 3</td>
<td>Select the MAC Mode (default is 0)</td>
</tr>
<tr>
<td>RR</td>
<td>XBee Retries</td>
<td>0x0 - 0x6</td>
<td>Set number of retries the modem will execute in addition to the 3 retries provided by the 802.15.4 MAC</td>
</tr>
<tr>
<td>RN</td>
<td>Random Delay Slots</td>
<td>0x0 - 0x3</td>
<td>Minimum value of the back-off exponent in the CSMA-CA algorithm that is used for collision avoidance</td>
</tr>
<tr>
<td>NT</td>
<td>Node Discover Time</td>
<td>0x1 - 0xFC</td>
<td>This sets the maximum time for the Node Discover command</td>
</tr>
<tr>
<td>NO</td>
<td>Node Discover Options</td>
<td>0x0 - 0x1</td>
<td>Enable node discover self-response</td>
</tr>
<tr>
<td>TO</td>
<td>Transmit Options</td>
<td>0x00 - 0x05</td>
<td>Bit0 - Disable MAC ACKs. Bit2 - Send to broadcast PAN ID. - All other bits must be set to 0</td>
</tr>
<tr>
<td>----</td>
<td>------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>C8</td>
<td>802.15.4 Compatibility</td>
<td>0x0 - 0x3</td>
<td>Bit0 - Transmit like legacy 802.15.4 at the cost of reduced throughput, bit1 - Do node discovery like 802.15.4 which locks out other AT commands longer than necessary</td>
</tr>
<tr>
<td>CE</td>
<td>Coordinator Enable</td>
<td>0 - 1</td>
<td>0 - End Device, 1 - Coordinator</td>
</tr>
<tr>
<td>SC</td>
<td>Scan Channels</td>
<td>0x0 - 0xFFFF</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>Scan Duration</td>
<td>0x0 - 0x0F</td>
<td>Scan Time = N* (2 ^ SD) * 15.36ms. N=# channels: XBee = 16</td>
</tr>
<tr>
<td>NI</td>
<td>Node Identifier</td>
<td>0 - 20 ASCII characters</td>
<td>Name of the node</td>
</tr>
<tr>
<td>PL</td>
<td>Power Level</td>
<td>0 - 4</td>
<td>Transmitter output power</td>
</tr>
<tr>
<td>PM</td>
<td>Power Mode</td>
<td>0 - 1</td>
<td>If enabled, boost mode improves sensitivity by 2dBm and increases output power by 3dB</td>
</tr>
<tr>
<td>CA</td>
<td>CCA Threshold</td>
<td>0x28 - 0x50</td>
<td>Clear Channel Assessment (CCA) threshold. If the modem detects energy above the CCA Threshold, it will not transmit. The CCA parameter is measured in units of -dBm.</td>
</tr>
</tbody>
</table>

**IoT control in UFMG:** There are two types of IoT devices in UFMG: Raspberry Pis and Advanticsys nodes. For Raspberry Pis, the orchestration of the WiFi interface during the experiment will be performed either using common embedded Linux CLI tools (iwconfig, ifconfig, etc) or via Ethanol [1]. Ethanol is an SDN interface for the control of IEEE 802.11 radios being developed in UFMG. The types of parameters that will be supported are based on the Ethanol messages, and are subject to the availability of such parameters in the Raspberry's WiFi driver. A lengthier description of Ethanol is presented in the description of the UFMG testbed.

Advanticsys nodes are very limited in CPU and memory, and as such they cannot be controlled remotely using a SDN-like interface. Although one could use very simple control protocols such as 0MQ, those devices are very restricted in memory and in processing, and every byte available counts. As a consequence, UFMG decided not to provide a control mechanism for these nodes, in order to maximize the space for the user's application. Users will have to program the devices with their own code, which may be able to control some of the wireless parameters if the programmer implements that functionality.

**IoT control in UFES:** At UFES testbed, in the orchestration phase, the experimenter can dynamically configure the cameras by interacting with the Experiment Orchestrator, using a REST API (more details in the description of the testbed). The Experiment Orchestrator sends these commands to the SDN Controller, which, in turn, communicates with the Camera Agent. Beyond the possibility of dynamically changing parameters that were already exposed in the provisioning phase (image resolution and image acquisition frequency), the experimenter can also retrieve status information about the cameras. These parameters provide a wide range of experimentation capabilities, since they change the network traffic matrix, and also affect the precision of robot navigation based on computer vision in experiment 2.2 of FUTEBOL. In addition, the UFES testbed also provides indirect access to a mobile robot, since the
experimenter can control the robot movement through a set of processing functions in the datacenter that generate commands that are wirelessly transmitted to the robot.

**PC-based Wireless Control Units**

The UFMG FUTEBOL testbed provides mini PCs with common wireless capabilities for such devices, namely Bluetooth and WiFi. Those resources are made available to experimenters. The UFMG testbed can provide simple and enhanced WiFi devices. Various wireless tools are available to the experimenters, like hostapd, wpa_supplicant, iw, and iw tools, empowering the experimenter to manually create a scenario with access points and wireless stations, or create a wireless mesh, ad hoc or wifi direct scenario. Enhanced devices are Linux environments on mini PCs that use Ethanol.

The UFES testbed provides physical servers with Linux OS and hostapd daemon installed, which enables to use Linux servers as WiFi APs. The parameters exposed to the users are: the transmission power, and the channel. In the orchestration phase, the experimenter can dynamically configure the WiFi APs by interacting with the Experiment Orchestrator, using a REST API (more details in the description of the testbed). The Experiment Orchestrator sends these commands to the SDN Controller, which, in turn, communicates with the AP. Beyond the possibility of dynamically changing parameters that were already exposed in the provisioning phase to configure the WiFi cell (transmission power, and channel), the experimenter can also retrieve status information about the APs, and to force the AP that is currently associated with the robot to send a beacon message with a channel switch advertisement (CSA) in order to change the robot’s channel and to perform a handover. These parameters enable the experimenter to configure WiFi cells and to dynamically control the handover according to the robot's position. More details about the mobility solution are presented in the testbed description.

**LTE Shared Access**

Due to the legal restrictions regarding the spectrum use of commercial LTE equipment, the federated access to the LSA testbed is restricted with a limited command interface. The command interface is implemented as a software component that communicates with its counterpart connected to each piece of LTE hardware over a secure TCP connection.

The LSA Testbed will provide access to a VM with interfaces to set various parameters for the experiments. During the initial setup, the experimenter can set the following testbed configuration in the RSpec:

- Set the virtual locations of the reserved LTE basestations for LSA purposes, so that the experiment can involve scenarios from anywhere in the world, although physically the LTE hardware is located in Finland. As in real life scenarios, the location of basestations will remain the same during an experiment.
- Create simulated LTE basestations and set their virtual locations in addition to the real LTE equipment to allow experimenting with larger scenarios.
- Select the frequency of each LTE basestation from a set of available frequencies. The frequencies of LTE equipment are restricted to the spectrum license of the LSA testbed, given by The Finnish Communications Regulatory Authority - FICORA. The simulated basestations can simulate operating in any frequency.
- Select the behavioral mode of the LSA controller, for example if the basestations should change their frequency when evacuating a band or just lock the basestation for the duration of possible interference with an incumbent user.

During the actual experiment or LSA scenario, it is possible to:
Add simulated incumbent users (primary users of the spectrum) with a specific map location, duration of spectrum use, interference protection zone and operating frequency, which will affect the behaviour of the LSA system.

- Get reports of the performance and behavior of the LSA testbed, including evacuation time of LTE equipment that would interfere with primary users.

**Optical/Packet Orchestration**

The convergent optical/packet orchestration is implemented by extending the Openflow protocol to support the control of optical physical layer based on the ITU-T G 694.1 recommendation, as shown in Figure 3. In terms of convergent network architecture, it means that the southbound interface is a common and uniform interface based in OpenFlow protocol from a large variety of devices. The devices are SDN enabled and support the extended OpenFlow messages (e.g. `flowmod`) for physical layer parameters configuration. For instance, in the Bristol testbed, an experimenter can allocate the virtual network at packet level using VLAN-ID 57, and assign it to the frequency 193.8 THz (WDM) = 1546.92 nm wavelength in the optical domain.

The orchestration is carried out through the northbound API of the SDN controller running as a VNF. Thus, an experimenter can send commands (using a REST API) to the controller to re-plan e.g. the optical `lightpath` during the execution of an experiment. On the other hand, if the experimenter wishes to map multiple VLANs to be transported by the same `lambda` or requires any traffic engineering implemented by combinations from L2 to L4, these requirements are feasible by using the programmability of OpenFlow protocol.

**NFV and service orchestration**

Two orchestration mechanisms for virtual functions are being developed in FUTEBOL: one, based on virtual machines, which uses OpenStack for orchestration; and another that is based on containers and uses LXD’s and Docker’s REST APIs. The reason for the two is that they target different objectives and have different implementation requirements.

The OpenStack orchestrator aims to achieve (i) **VM placement**: deciding the location of a VNF in a physical server according to resource availability and experiment requirements; (ii) **horizontal scaling**: incrementing or decrementing the number of instances of a specific VNF in order to cope with a specific experiment need; and (iii) **SFC**: deciding on how to steer the traffic flows across an ordered set of VNFs that compose the service. To implement SFC and horizontal scaling functionalities according to the OpenStack mechanisms, the user will interact with the Orchestrator using TOSCA templates².

Because the OpenStack Orchestrator runs on top of the testbed’s infrastructure, it requires major changes to the existing testbed’s CF. Because of this, not all testbeds will implement it, unless they specifically require some of its features. Also, migrating VMs limits the orchestration within a testbed, which does not satisfy the requirements of all experimenters.

To support migration among testbeds, the migration functionality is deployed by virtualized functions and services using containers and the previously mentioned COPA. The container migration can be deployed in two ways: live migration and offline migration. The offline migration consists in stopping the services running in the container and moving the container between virtual machines, then restarting the container in another place. Live migration, in the other hand, consists in making a snapshot of the container, while it is running in the origin server, and the migration is started. After migration is

---

² [docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html](http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html)
completed, the snapshot is restored in destination VM, and the container characteristics that were in the source VM are maintained in the destination. Currently, COPA assumes that both origin and destination container pools are connected to the same layer-2 network, which is generally the case for single testbed experiments and can be achieved in some cases for inter-testbed experiments as well. Thus, it reconnects the container after migration to the layer-2 network keeping IP address configurations untouched and relies on the network to reestablish active connections. In the future, we might include the functionality so that COPA can actively control the (re-)establishment of connection paths within the network.

Some experimenters may need to have access to basic container management functionalities (creation, deployment, migration, etc.) in case they want to implement their own service orchestration routines. Thus, the COPA tool offers a REST API that provides the same functionalities that can be accessed over the GUI. The API also offers information on resource consumption (CPU, memory, and networking) of running containers. Based on this API, experimenters will be able write and run their own orchestration scripts and manage services during the experiment lifetime. For example, this approach is being used in Experiment 2.1 to jointly orchestrate video streaming services and radio processing functions.

Figure 7 shows an example of how this REST API can be used by an experimenter. After the provisioning of the resources for the experiment, the experimenter might write an Orchestration Script using his/her preferred programming language or any service-oriented tool. The calls to the REST API can be performed over the same SSH tunnels that the experimenter uses to access the COPA GUI. Upon receiving generic container management calls over the REST API, COPA will then proceed with the Platform Specific calls to perform the operations within the container platform in use for the experiment. If the experimenter wishes, it is also possible to upload the Orchestration Script into one of the VMs involved in the experiment and run it from there.

Some of the operations available through the REST API are briefly presented below:

- **Create** (Pool, Container Name, Image): creates a new container in a pool based on an image from the catalog;
- **Start** (Pool, Container): starts a previously created container;
- **Stop** (Pool, Container): stops a container that has been previously started;
- **Freeze** (Pool, Container): pauses the execution of a container that has been started, saving its state;
2.3 Experimentation and convergence control

A fundamental concept on the CF design is the converged experimentation slice that allows the experimenters to allocate converged resources. These resources can be either wireless or optical networking equipment, computing resources, virtual functions, etc.

To allocate this kind of heterogenous resources for a converged experiment, experimenters must reserve them through the control framework, either through allocating resources in a testbed that contains heterogenous resources, or by stitching different testbeds together (as described in deliverable 3.1).

We illustrate this with a particular example of reserving a converged slice from experiment 3.2, as illustrated in Figure 8. In this experiment, the resources available for the experimenters are divided into wireless and optical.

The wireless IoT nodes collect environment information with different kinds of sensors, such as temperature, sound and humidity, as well as the conditions of the wireless channels, and can be configured by the experimenter to send the data they collect or retransmit the data they receive from other nodes. Connected with the wireless nodes is an analog wireless/optical converter, the Radio over Fiber equipment. It receives all the wireless signals in its bandwidth and modulates the lightwave sent through the optical fiber with them. Optical switches are connected in the optical fiber, allowing the experimenter to choose where the lightwave signal is sent and in which virtual baseband unit it will be processed after conversion. The lightwave signal is then converted back to an electrical signal as the same radio frequency wave received before the wireless/optical conversion. The RF signal is demodulated and digitized, then it can be processed by the virtual baseband unit.
These two control units can be used together to create new convergent resources. These resources enable the centralization of processing and can be very useful in experiments involving technologies such as C-RAN and Radio over Fiber, as shown in the figure above. Also, it will be used to compare the performance of Digitized Radio over Fiber with Analog Radio over Fiber.

Besides the reservation of different resources, orchestration of components is necessary for a meaningful experiment. Due to the heterogeneous nature of resources, the control mechanisms for them have different northbound and southbound interfaces. Table 2 summarizes the different southbound and northbound APIs present in the FUTEBOL CF. Currently, experimenters that wish to employ a converged orchestrator in their experiment can use the following southbound and northbound APIs to control the FUTEBOL resources:

**Table 2: Summary of northbound and southbound APIs**

<table>
<thead>
<tr>
<th></th>
<th>SDR</th>
<th>IoT</th>
<th>WiFi</th>
<th>Containers</th>
<th>Packet/ Optical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Northbound</strong></td>
<td></td>
<td>REST (UFES)</td>
<td>REST (UFES)</td>
<td>REST</td>
<td>REST of ODL (Bristol)</td>
</tr>
<tr>
<td><strong>Southbound</strong></td>
<td>0MQ</td>
<td>Bash over SSH (UFRGS)</td>
<td>Ethanol</td>
<td>LXD/Docker</td>
<td>OpenFlow</td>
</tr>
<tr>
<td></td>
<td>Ethanol (UFMG)</td>
<td></td>
<td>OpenFlow agents (UFES)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example of such orchestration can be found in experiment 2.1. There the SDR controller is used as basis for the sampling rate of an LTE-PHY transmission, based on the measured application rate, to try to adjust the bandwidth to the required wireless throughput. This SDR controller is integrated to a Ryu SDN controller that controls an OpenFlow enabled switch the change the CIR given to the flow, to guarantee its required QoS.

Also in this experiment, COPA is used to migrate application servers closer to customers, based on the measured load. Currently, we are working on integrating this with the migration of baseband processing, making it a converged orchestrator for virtual functions, SDR processing and SDN switches.
3 FUTEBOL CF DEPLOYMENT IN TCD TESTBED

3.1 Testbed description

In Trinity College Dublin, the Iris Testbed provides the resources for flexible radio experimentation, through virtualization technologies and Software Defined Radio. The testbed is comprised of 16 ceiling mounted USRPs, mounted with SBX daughterboards, giving them a frequency range between 40 MHz and 4 GHz.

By pairing the USRP frontends with processing resources in the form of virtual machines, the testbed users can run their own radio applications over the air. These virtual machines come with pre-installed software, to make it easier for the experimenters to start running their experiments. The testbed architecture is depicted on Figure 9, with pictures of the ceiling mounted USRPs and the server rack used for the processing.

Of the total 16 USRPs, fourteen are of the N210 model and 2 are of the X310 model. The X310s were recently added to support the sampling rates required for standards-compliant LTE. This means that it was possible to add disk images with open-source LTE implementations pre-installed, such as srsLTE. Besides LTE software, spectrum visualization software was also recently added, to help experimenters view the state of the experiment, detect errors and visualize results.

Below is a summary description of the different disk images available to the user and a short description of their utility:

- **Plain UHD**: This is useful for advanced users that want to either run a particular version of a library or that want to develop their own applications without the use of a framework.
- **IRIS**: This IRIS image is useful for experimenters that use the legacy IRIS SDR Framework. This framework provides a set of signal processing algorithms and fine-grain control over the number of threads and control blocks used.
● **GNU Radio:** GNU radio is the most popular SDR framework available. It provides a large amount of signal processing blocks and an easy to use graphical interface. It is ideal for beginners.

● **Fosphor:** The Fosphor image can be used by experimenters to view the spectrum and a waterfall diagram. This information is crucial to debug problems and visualize the transmissions.

● **srsLTET:** The srsLTET software provides a standards-compliant implementation of an LTE UE and an LTE eNodeB. Note that it currently can only be used on the USRPs X310, due to the sampling rate of LTE not being compatible with the N210’s master clock.

Besides the five servers exclusively dedicated to USRPs, there are two servers in the Iris testbed that are dedicated to run pure virtual machines; i.e., without any USRP attached. These can be used for multiple functionalities, such as application traffic generation, controllers, or core network functionalities. A summary of the disk images currently available for pure VMs can be found below:

● **Plain Linux:** Ubuntu Linux distribution for general use. Useful for application traffic generators or other uses.

● **LXC+Docker:** This image comes with pre-installed LXC, Docker and CRIU for NFV deployment and migration.

● **Open Air CN:** This disk image has pre-installed the required components for and LTE core network, namely, an HSS, an MME and SP-GW. An experienced user can configure the core network to work with a USRP with srsLTE working as an eNB.

### 3.2 Aggregate Manager implementation

The Cloud Based Testbed Manager (CBTM) is an aggregate manager developed by TCD to reserve USRPs plus VMs for wireless experimentation. It was originally developed in the Fed4FIRE project, with improvements for stability, maintainability and extra functionality developed under FUTEBOL.

The overall architecture of the CBTM is illustrated in Figure 10. There, it can be seen that there are four main components to the CBTM system: the CBTM AM, the CBTM PHP, the CBTM Coordinator and the CBTM Core.

The CBTM AM is responsible for interaction between the SFA clients (such as jFed) when reserving and provisioning experiments. This means that that the CBTM AM is responsible for implementing the SFA API and interacting with the other components to actually allocate and provision experiments, namely the CBTM coordinator and core. This piece of software was developed by extending the GCF reference aggregate manager\(^3\) to support the required functionalities.

Besides the CBTM AM, the CBTM PHP is a website built in PHP for administration and management of internal users. Internal users can use this website to make reservations, similarly to jFed, and the testbed administrator can view the reservations made and delete them if necessary.

Both the CBTM AM and CBTM PHP will translate their requests to an internal protocol and interact with the CBTM Coordinator to reserve and provision experiments. To check whether resources are available, make reservations, and delete reservations, the CBTM Coordinator interacts with a reservation database. New reservations are added to the database, if there are available resources during the desired timeslot. Once reservations expire, a daemon deletes them.

Once reservations are valid, the CBTM Coordinator interacts with the CBTM Core, which is responsible for the actual provisioning of resources. To do this, the CBTM Core uses Libvirt\(^4\), a virtualization library, to create virtual machines in the data processing servers. The VMs are created over a KVM hypervisor and have a passsthrough Ethernet interface, granting the user exclusive access to the USRP.

---

\(^3\) [https://github.com/GENI-NSF/geni-tools](https://github.com/GENI-NSF/geni-tools)

\(^4\) [https://libvirt.org/](https://libvirt.org/)
The user’s SSH keys are added to the VM, which means that once the provisioning is finished, the user can remotely login and perform his or her experiments.

3.3 NFV Manager implementation

In TCD, the previously described CBTM is used to deploy Virtual Machines, with pre-installed baseband processing software. This allows experimenters to deploy virtual network functions capable of performing baseband processing.

As the CBTM was developed to provision resources as opposed to orchestrate them, we will also deploy COPA, which will enable virtual function migration. To enable COPA, it is necessary to prepare containers with a set of predefined virtual functions.

We are currently preparing the following list of containers that will be available for experimenters to deploy and orchestrate:

- **UHD Interface**: This container will be responsible for transmitting IQ samples to the USRP hardware. It will contain socket to transmit/receive IQ samples to/from a networking interface, thus decoupling the baseband processing from the hardware interface. This container should always be directly connected to the USRP to perform then, thus it should not be migrated.
- **GNU Radio OFDM**: This container will modulate an OFDM waveform via the GNU Radio examples. This will include both a transmitter and a receiver.
- **srsLTE PHY**: This container will provide the Physical Downstream Shared Channel (PDSCH) of LTE from the srsLTE examples. This will include both a transmitter and a receiver.
- **srsLTE full-stack eNB**: This container will contain a full-stack implementation of the srsLTE eNB. As this piece of software has very strict latency constraints we are still verifying the feasibility of supporting this container.
- **OpenAir Interface Core Network**: This container will contain the elements necessary to implement a LTE core network (MME, SPGW and HSS).

3.4 Validation of the experiment control and service orchestration

In TCD we have started the implementation of SDR control and orchestration. For experiment 2.1, we have changed the srsLTE PDSCH example to support changing the sampling rate dynamically, without restarting the program. This allows for quickly changing the IQ fronthauling rate dynamically according to the network needs. To do this, the SDR application uses a 0MQ socket to report the application rate.
to the controller. The controller, based on this information, decides the appropriate system bandwidth (and accordingly the sampling rate).

We are also in the process of deploying COPA in our testbed. We are working on preparing containers to be deployed with this system for baseband processing and video traffic applications. This will enable us to orchestrate these two different applications having a network-wide view.
4 FUTEBOL CF DEPLOYMENT IN UNIVBRIS TESTBED

4.1 Testbed description

The federated FUTEBOL testbed at Bristol University is managed by the High Performance Networks Group. This is composed by the following equipment shown in Figure 11:
- 3 x ADVA FSP 3000 DWDM ROADM nodes (OpenFlow-enabled).
- 4 x Ethernet switches (OpenFlow-enabled) (NEC IP8800).
- 1 Polatis fibre switch 192 x 192 (Openflow-enabled).
- Physical servers for running virtual machines, experiment orchestrator, VTAM, OFAM, Flowvisor, and VNFs e.g. for video streaming.

4.2 Aggregate Manager implementation

FOAM is an open source AM for the management of OpenFlow resources. It was initially developed to handle OpenFlow packet-switched networks. FOAM was extended at the University of Bristol to manage OpenFlow-enabled optical devices as a development work within the Fed4FIRE project. To provide experimentation slices of optical resources, optical FOAM communicates with an optical flowvisor, a tool used for slicing OpenFlow network...
resources including both packet and optical devices. Extended FOAM will be used for the management of SDN-enabled optical devices in FUTEBOL optical testbeds.

4.3 NFV Manager implementation

The NFV Manager provisions and manages computing slices dedicated to run VNFs and service tools. It comprises the functionalities that are used to manage the interaction of a VNF with computing and network resources under its authority, as well as their virtualization-specific tasks in the testbed management. It translates the requests received from the federation into specific commands for each testbed resource.

To implement the new functionalities required by the NFV Manager which includes the NFV catalogue, existing tools for management of virtualization servers such as VTAM have to be extended. The second option that we consider to implement the NFV Manager is as an extension of the Cloud AM that uses OpenStack. Our implementation strategy is to apply the principle of the separation of concerns. By separation we mean that the current testbed deployment will be used to support the network orchestration functionalities, whereas for the NFV functionalities we will contribute as a tester deployment of Openstack AM under development by UFES testbed.
4.4 Validation of the experiment control and service orchestration

The Bristol testbed relies on a SDN-enabled Virtual Infrastructure (VI) that supports experiment control and service orchestration. This testbed can be used to demonstrate the network orchestration capabilities of the FUTEBOL CF. VNFs can be deployed on top of VIs to take advantage of the flexible and simplified composition and management of VNFs.

Figure 12 shows the convergent orchestrator (CO) that is proposed for multi-technology network transport scenarios including optical transport network and packet switched networks available in the Bristol testbed. The CO is an element that belongs to the experiment control layer that interacts with the controllers of physical resources (e.g. Open Daylight) and uses the orchestration functionalities.

This CO has been proposed as a solution for end-to-end network orchestration. This includes a VI composition that is able to select the physical resources in the different domains and layers (i.e. computing, L2 and optical resources) to provide the required end-to-end IT and network services. Furthermore, we will provide a real-time network replanning mechanism that can also be used to optimise the usage of physical resources.
A network experimenter can use the orchestration and virtualization platform to create service, for instance, a video streaming service. The VI will be provided as a response from the AM of Bristol federated testbed. This VI includes virtual machines (VMs) to host the VNFs, a virtual network to isolate the experimenter resources (e.g. VLAN tag) and a set of virtual links that are used to connect the VMs and the virtual switches.

The virtual topology visualization and the network programmability are abstracted as a set of resources within the SDN controller. For instance, the experimenter can program the flows, which are then directed by the controller to the physical switches. The current testbed has virtual links that are mapped onto end-to-end physical paths that span SDN-enabled packet and optical domains. It allows us to show the infrastructure replanning so that it can be done by the network experimenter to improve service delivery to end-users. Infrastructure replanning will be used to reconfigure the optical lightpaths using SDN technology based on optical network. Experimenters can test how replanning changes the mapping of virtual links into shorter and less congested lightpaths using SDN controller over the optical network. This will lead to better connectivity service over the VI and better performance observed by end-users.

### 4.4.1 Convergent slice allocation

The experimenter can specify a set of VMs, L2 connectivity with VLAN tag and Optical Devices ports with a lambda, according to the RSPEC below.

```
<rspec generated="2017-01-16T23:51:46.709Z"
    generated_by="jFed RSpec Editor" type="manifest" xmlns="">
    <openflow:sliver description="OF request example" email=""/>
    <openflow:controller type="primary" url="tcp:137.222.204.71:6633"/>
    <openflow:group name="optical">
        <openflow:datapath
            component_id="urn:publicid:IDN+openflow:ofam:univbris+datapath+00:00:00:00:0a:21:00:0a"
            component_manager_id="urn:publicid:IDN+openflow:ofam:univbris+authority+cm"
            dpid="00:00:00:00:0a:21:00:0a">
            <openflow:port name="MOD-5-08" num="1008"/>
            <openflow:port name="MOD-5-14" num="1014"/>
            <openflow:port name="OL-1" num="1"/>
            <openflow:port name="OL-2" num="2"/>
        </openflow:datapath>
        <openflow:datapath
            component_id="urn:publicid:IDN+openflow:ofam:univbris+datapath+00:00:00:00:0b:21:00:0a"
            component_manager_id="urn:publicid:IDN+openflow:ofam:univbris+authority+cm"
            dpid="00:00:00:00:0b:21:00:0a">
            <openflow:port name="OL-1" num="1"/>
            <openflow:port name="OL-3" num="3"/>
        </openflow:datapath>
    </openflow:group>
</rspec>
```
5 FUTEBOL CF DEPLOYMENT IN VTT TESTBED

5.1 Testbed description

VTT’s LSA testbed is located at the VTT premises in Oulu, Finland. The testbed enables experimenters to try out various scenarios related to Licensed Shared Access (LSA) spectrum sharing with commercial LTE equipment. The location of the basestations and other spectrum users can be simulated to create larger and more interesting scenarios. By request, it is possible to connect additional LTE resources from the 5G Test Network Finland to the federated LSA testbed, including macrocell basestations attached to outdoor masts.

5.2 Aggregate Manager implementation

VTT’s LSA testbed allows the experimenter to reserve commercial LTE hardware through Fed4FIRE. The aggregate manager implementation in the VTT testbed is based on the reference GENI GCF aggregate manager, which facilitates querying and reservation of the resources available in the testbed. The reference implementation has been extended by VTT with interfaces to:

- Set the basestation frequencies of the LTE equipment and map the location to model scenarios around the world. The frequencies of the equipment are limited by the Finnish Communications Regulatory Authority (FICORA). During the experiment the location of the basestations will remain fixed.
- Set the time and map the location for the primary use of the spectrum and to select the protection algorithm.

The aggregate manager requires the use of the GENI v3 API.

Initially, experimenters will be provided direct access via SSH to the virtual machine with access to the LTE equipment. Access will be provided on a first come, first served basis for a single user of the testbed, thus subsequent users have to wait for the testbed to become available. The maximum duration of a reservation might be restricted. Eventually the testbed users will be provided with access to multiple nested KVM/Qemu virtual machines through the CBTM control framework.

The virtual machines have a Ubuntu 14.04 LTS linux installation with a MySQL database as the LSA repository for setting the frequency and location of the basestation and the time and location of the primary user.

5.3 NFV Manager implementation

Not provided.

5.4 Validation of the experiment control and service orchestration

The network experimenter can specify a virtual machine containing the LSA repository with a request RSpec and will receive the IP address of the VM in the manifest RSpec for accessing it through SSH.

The following example shows the request RSpec when the testbed experimenter wishes to instantiate a node with two basestations in two different locations and with different frequencies. The first basestation
corresponds to the LTE equipment in the testbed, whereas the second one is a simulated basestation. The experimenter also sets the location and time for the primary user of the spectrum.

```xml
<rspec>
  <node>
    <sliver_type name="vm"/>
    <services>
      <login authentication="ssh-keys" hostname="futebol.willab.fi" port="22" username="?"/>
    </services>
    <basestation virtual="false">
      <frequency value="123"/>
      <location name="oulu"/>
    </basestation>
    <basestation virtual="true">
      <frequency value="456"/>
      <location name="mordor"/>
    </basestation>
    <user>
      <location name="oulu"/>
      <time value="1m"/>
    </user>
    <interface>
      <ip address="193.166.161.185"/>
    </interface>
  </services>
  <host name="futebol.willab.fi"/>
</node>
</rspec>
```

Due to the legal restrictions regarding the spectrum use, the federated control of the commercial LTE equipment of the LSA testbed is restricted with a limited command interface, having commands to lock and unlock the basestation, and select the operating frequency of the basestation. The command interface is implemented as a software component that communicates with its counterpart connected to each piece of LTE hardware over a secure TCP connection.

During the experimentation, the user can create incumbent users with a specific time of spectrum use, operating frequency and interference protection zone. The LSA system will automatically evacuate the LTE basestations that would interfere with the incumbent users.
6 FUTEBOL CF DEPLOYMENT IN UFRGS TESTBED

6.1 Testbed description

The UFRGS testbed is maintained by the Computer Networks Research Group. The testbed is focused on devices for wireless research, Internet of things, sensor networks and cognitive radio. The testbed access is provided mainly via the Fed4FIRE credentials and tools. All users with valid Fed4FIRE credentials are able to book and control the resources in UFRGS’s testbed. Also, the devices and the experiments are controlled by the CBTM control framework. The types of experiments and technologies available in the testbed are:

- Wireless and optical Integration with Radio Over Fiber;
- Wireless experimentation (WiFi, 3G, 4G, 5G) including new protocols and technologies over programmable radio platforms (i.e., Software-Defined Radio);
- Software-Defined Networking and Network Functions Virtualization with OpenFlow and container-based or traditional virtualization platforms;
- QoS/QoE for multimedia services for mobile and fixed networks;
- New architectures for cloud, fog networking, and Internet of Things with sensor devices based on Arduino and Raspberry Pi platforms.

Federated equipment includes:

**Virtualization and control server**

- DELL PowerEdge R530, with Intel Xeon 8 cores 5-2609 1.7 GHz, 8GB RDIMM, 1TB SATA Hot-plug Hard Drive, 4 ethernet 1 gigabit NIC, running Ubuntu 16.04.2 LTS.
- 6x Dell Utratop Brix with a Core i7-5500U,2.4Ghz TURBO 3Ghz - 3MB, 4GB RAM and 120Gb SSD, running Ubuntu 16.04.2 LTS.

**Wireless**

- 6x Ni USRP-2901, 70 MHz to 6 GHz, with 2X2 MIMO for two distinct antennas:
  - Dual Band vertical antenna (2.4 - 2.48 GHz and 4.9 - 5.9 GHz).
  - Ettus VERT900 vertical antenna (824 to 960 MHz, 1710 to 1990 MHz).

**IoT Devices**

- 16x Raspberry Pi 3 Model B, with a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, with 1GB RAM, one gigabit ethernet port, one wireless LAN and Bluetooth Low Energy (BLE) on board.
- 16x Arduino Uno, with 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, and an ICSP header.
- Xbee-Pro S2C (2.4GHz) wireless transceivers associated with a Arduino Uno

**Mobile**

- 2x Motorola Moto Z with Qualcomm Snapdragon 820, running Android 6.0.
- 1x Asus Zenfone 2, with Intel Atom Z3580, running Android 6.0.
- 2x Samsung Galaxy S7, with Qualcomm Snapdragon 820, running Android 6.0.
6.2 Aggregate Manager implementation

The AM authenticates experimenters, reporting which resources are available in the testbed, interacting with CBTM in order to allocate and provide resources for the users. The Aggregate Manager uses the GENI v3 API. The GENI Aggregate Manager API allows advertising and allocating the resources to slices in the form of slivers. The experimenter gets the GENI certificate and slice credentials, renewing them when necessary.

Regarding extensions to the default implementation of CBTM, UFRGS testbed overloads the support to a new sliver of Raspberry Pi and Arduinos devices. The Raspberry Pi can be allocated stand alone or connected with an Arduino. If the Raspberry is allocated in the stand alone mode the CBTM will create a user with the experimenter credentials and give full access to the resources. When allocated with an Arduino, the raspberry will serve as a gateway to programming the the Arduino platform. Also, the experimenter can select an Arduino with a specific shield connected. Currently, only xbee shields are available on the UFRGS testbed.

6.3 NFV Manager implementation

The NFV manager at UFRGS is responsible for managing containers on behalf of experimenters, and provides services and network functions from a catalog available in the testbed. This allows the experimenter to deploy services and applications on demand during the experiment, and perform operations such as service migration between different testbeds. We have developed the COPA tool and integrated it with the CBTM control framework. The experimenter inserts an RSPEC in jFed specifying the parameters of one or more pools of containers, requesting other types of resources within the same experimentation slice (wireless nodes, IoT devices, etc). Also, the experimenter may request pools of containers in different testbeds as well.

To instantiate a pool of containers, CBTM parses the RSPEC received and instantiates the resources, allocating the pool of containers as a VM. It is just a regular virtual machine with the tools installed to instantiate containers (i.e., LXD or Docker). The experimenter is not granted direct access to these pools of container virtual machines (e.g., through ssh). Thus, all of the container pools are controlled via the COPA web interface.

The role of COPA is to control the pools of containers, including the controlling of service lifecycle (creation, migration, removal, etc), using one COPA instance per experiment, connecting COPA and pools of containers via a REST API. The experimenter interacts with a web interface to choose a set of services from a catalog, and then COPA is informed about the services the experimenter wants and does the placement of containers in each pool. Figure 13 displays the high-level organization of the components just described.
Figure 13: Container management based on the integration of CBTM and COPA
6.4 Validation of the experiment control and service orchestration

The validation of the orchestration and control in UFRGS can be presented using the results from Experiment 2.1. This experiment is focused on the orchestration of wireless and optical network components as well as services, as illustrated in high-level in Figure 14. Using a container system, an adaptable video service (DASH) can be provided from one of the testbeds involved in the experiment. Video clients connected at Wireless Nodes are then initiated from the wireless edge networks. An orchestrator function implemented within the COPA tool can keep track of network traffic between all containers involved. Once the traffic surpasses a certain threshold, in which the traffic between testbeds gets considerably larger than the traffic internal to each testbed, the orchestrator may decide to migrate the video server container in the direction of the testbed with the majority of the clients. With the COPA tool, migration can be performed live, meaning minimal down-time is observed, and connections for the video clients are kept alive throughout the whole process.

Another use of COPA will take place in Experiment 3.1 during Stage 1 of the experiment: The first stage of Experiment 3.1 focuses on the orchestration of computing routines. Using a container system (Docker), two different containers are created to host the algorithms to process speech and sign language commands inputs from an IoT smart light application. Within different testbeds, VMs are instantiated with a Docker platform able to receive and execute the containers. The testbeds (UnivBris, UFRGS, and UFMG) are connected through the RNP backhaul containing optical links, whereas the IoT application was allocated by the control framework placed within the Brazilian institutions using wireless links. The IoT wireless nodes send their speech/sign language across optical links to be served by VMs placed inside UnivBris. All the infrastructure is monitored through the gathering of QoS metrics, such as throughput, jitter, and delay. Whenever a bad QoS is detected, the containers in use have to migrate from the UnivBris to another testbed in Brazil to get closer to the IoT application, mainly due to intolerable delay.
7 FUTEBOL CF DEPLOYMENT IN UFMG TESTBED

7.1 Testbed description

The UFMG testbed was designed to allow experimentation in a number of wireless technologies related to the SDN and/or IoT concepts. Thus, UFMG provides a rich set of resources, from VMs to resource-constrained wireless devices. It also supports a number of wireless technologies, such as WiFi and Bluetooth, and many others using programmable radios (USRPs). Further, the testbed supports SDN by using a physical OpenFlow switch. Below is a list of the available resources:

- Eight Dell Alienware Alpha R2 Mini Gaming PCs (Mini PCs), supporting WiFi, Bluetooth and USRPs;
- Sixteen Advanticsys MTM-CM5000-MSP IoT Nodes;
- Sixteen Raspberry Pi 3 Model B;
- One OpenFlow switch Pica8 model P-3297.

7.2 Aggregate Manager implementation

UFMG employs CBTM as its aggregate manager. Many extensions are being implemented in order to support resources that are not cloud-based (e.g. sensor nodes, SDN switches) or accessed via I/O ports on the physical machine.

In order to control resource-constrained devices such as the Advanticsys nodes, CBTM will be extended to communicate with software that controls the allocation and access to those nodes. This software is in charge of the exclusive access to the nodes, by only allowing one experimenter to access the USB of the Advanticsys node. It also wipes out the memory of the node after each allocation in order to avoid a user to inspect the node’s memory to infer the activities of the previous user.

A second extension is being implemented so that CBTM interacts with OpenFlow switches. CBTM will slice the OF switches using a divisor (e.g. Flowvisor, OpenVirteX). Depending on the needs of the user, either CBTM will provide an [IP, port] tuple so that the user can control the SDN slice using his preferred SDN controller, or it will instantiate a layer-2 learning switch over the network slice. The latter is supported in order to allow novice users to easily create a virtual network.
7.3 NFV Manager implementation

The UFMG testbed will provide network functions only for the Mini PCs, which can be used for experiments using USRPs, WiFi and Bluetooth, besides being used as a generic compute node. The following services are supported, in the form of different slivers. Each differ mainly in which hardware is available for the experimenter.

- **Basic Linux image with GnuRadio:** This image will have a Linux installed, together with GNU Radio. It can be used for USRP experiments.
- **Linux image with LTE and WiFi:** This Linux image is associated with the built-in WiFi card of the Mini PCs, as well as the USRP. It will also come pre-installed with a 4G stack. Users can configure the WiFi interface using typical Linux tools, such as `ifconfig`, `iwconfig` and `wpa_supplicant`. This sliver is used in stage 3 of experiment 2.1.
- **Linux image for Mote programming:** This sliver will be used in conjunction with the Advanticsys motes. It provides the programming environment necessary to build TinyOS programs, send them to the Advanticsys motes and receive the information that is provided by them. This image will come pre-installed with custom software that only allows the reserved motes to be programmed by the user. Further, this image will also contain the source code for TinyDB, which is a mote software that allows experimenters to obtain sensor readings using a SQL-like language. This tool is intended for novice experimenters.
- **Linux image with Ethanol:** This image will contain Ethanol, a SDN southbound interface for controlling WiFi radios, which can be used for WiFi orchestration.

All other types of computing devices in the testbed will not support different services. Those are the Raspberry Pi and the Advanticsys motes, which are IoT devices with processing and storage restrictions. In that sense, we expect that the software running on those devices should be simple and easy to install, and for that reason we leave for the experimenter to install his/her own software on the devices.

Again, it is worth stating that novice experimenters or experimenters desiring to use IoT devices on an application-level experiment can use the Advanticsys motes with TinyDB to collect data using a simple SQL-like interface. Since the Raspberry Pi nodes will not have any sensor pre-installed, they will be used mostly for experiments that want to test their protocols or applications with low computing capacity, and as such we expect them to be medium or advanced users. Since Raspberry Pis run a PC-like Linux, they should not encounter much difficulty operating the nodes.

7.4 Validation of the experiment control and service orchestration

Ethanol is a software-defined wireless network architecture to control IEEE 802.11 devices. At the controller, Ethanol runs as a POX module. So the controller can profit from Ethernet control capabilities provided by the POX implementation of OpenFlow, and wireless control capabilities provided by Ethanol. The Ethanol controller module is implemented using object-oriented programming, and therefore there are classes that abstract the physical access point, its radios, and the active SSIDs (or as we call the virtual access point). At the access points, we have altered hostapd – a Linux implementation of IEEE 802.11 authentication and access point services. Using Ethanol’s hostapd modified version, we can actuate on hostapd parameters and access various wireless metrics. It is important to highlight that the availability of some metrics depends on the wireless NIC, so there are some properties that may or may not be available to the experimenter, depending on the actual hardware. IEEE 802.11 proposes management and information services that can be used to extract information from the wireless stations, but this implementation is optional (actually only newer Apple devices provide such implementation). Therefore we also provide an agent that runs in the wireless station that allows access to wireless metrics.
and command of some actions (e.g. force a handover to a specified access point). For WiFi orchestration, the control can be performed using Ethanol as in the Raspberry Pis. Bluetooth control will be performed using command line tools from Linux.

In order to validate the ETHANOL orchestration interface for WiFi devices, we performed an experiment that adjusts the QoS of wireless flows. QoS is an important feature for enterprise networks, which require flow prioritization when there are network-intensive tasks running concurrently with more delay-driven services. For example, a VM migration among servers should have lower priority than the wireless clients accessing a video. Thus, the access points should have the means to prioritize those flows. The Ethanol agent provides an interface to configure these parameters, thus allowing the controller to provide per-flow programmability of QoS parameters.

For this experiment, a WiFi node in the testbed was configured to act as an Ethanol AP. This node is connected to two wired clients and a server, and one client is associated to the wireless link. This configuration emulates an enterprise network, where wireless and wired clients want to access an Intranet resource. We ensure flow prioritization using HTB scheduling, so that the throughput of the flows should be proportional to their assigned allocations.

We set up three queues at the Ethanol switch, with rates Q1=6, Q2=3 and Q3=1. When a new flow is detected (PacketIn events), we assign it to the proper queue based on its source address. All clients connect to the server using TCP. All the configuration was performed using Ethanol messages.

At first, one Ethernet client E1 and one wireless client W1 are active, and associated to queues Q1 and Q2 respectively. As expected, E1 received 3/9 of the capacity and W1 received 6/9 of the capacity. At 120 seconds, E2, which is associated to Q3, starts transmitting. The bandwidth allocations are now 6/10, 3/10 and 1/10 of the link capacity, as expected. Thus, Ethanol provides per-flow QoS programmability. The figure below shows the throughput over time.

![Figure 15: Ensuring the throughput in the wireless link using Ethanol](image)

Next, we used the Ethanol messages to destroy the queue. As a consequence, the throughput became unstable due to the lack of packet prioritization. This is shown in Figure 16.
Figure 16: Network throughput without packet prioritization
8 FUTEBOL CF DEPLOYMENT INUFES TESTBED

8.1 Testbed description

The UFES testbed aims to evaluate the impact of SDN technologies in applications with low end-to-end latency and high bandwidth requirements. We exploit the impact of a converged network control framework in cyber-physical systems running critical and real-time applications such as remotely controlled robotics. Moreover, we integrate diverse techniques to dynamically provide stringent network requirements by bridging communication and cloud computing ecosystems.

To this end, we enable experiments of real-time remote control of robots in Intelligent Spaces. An Intelligent Space can be defined as a physical space equipped with a network of sensors, capable of obtaining information about the world and a network of actuators that allows user interaction and task execution. Sensors, actuators, and computing services must be managed by an infrastructure capable of analyzing information obtained by sensors and making decisions.

The UFES testbed, shown in Figure 17, is composed by the following main components: an Intelligent Space, an edge data center, and a fiber optics connection between them. The Intelligent Space is an indoor room that is composed by: four cameras, four access points (APs), a mobile robot unit, and a SDN switch. The edge data center is composed by a SDN switch and a set of commodity servers.

The mission of the robot can be defined as reaching a destination or following a specific trajectory. The robot platform contains only the necessary components for wireless communication and execution of control commands. The cameras gather images from the Intelligent Space and send them via a SDN network to a cloud platform, which is responsible to process the images, calculate robot localization, and generate control commands. The commands are sent back to the robot via a convergent SDN network that controls both packet, optical, and wireless networking resources.

The UFES testbed enables experiments in the following areas:

- Software-Defined Networking;
- Cloud robotics;
- Mobile Edge Computing;
- Network Functions Virtualization; and
- Wireless Mobility.

Figure 17: UFES testbed
8.2 Aggregate Manager implementation

We are developing our own Control and Management Framework, called O2CMF. It intends to support the federation of cloud computing resources, and packet-switched, optical and wireless networking. Figure 18 shows how O2CMF is structured.

The Federation Layer acts as a mediator between requests from the federation (Fed4FIRE) and the Orchestrator, which interacts with the controllers of physical resources. This layer directly accesses the database that stores information about the reserved resources. That information is used to provide SSH access to VMs. Since the testbed offers computing and SDN resources, there are two Aggregate Managers (AMs): one for Cloud and other for SDN. To deal with SFA calls we used a reference implementation of the SFA interfaces developed by the GENI project, the GENI Control Framework (GCF). In each AM, in addition to GCF, there is a component, called Dispatcher, responsible for translating SFA calls into orchestration calls (via a RESTful API). The Orchestration Layer allows the integrated manipulation of resources and promotes the decoupling between the mechanisms that enable federation and the technologies that manage the resources. The Resource Controllers Layer contains the platforms responsible for the management of physical resources.

OpenStack was chosen as the cloud operating system, because it has extensive support by the open source cloud community, and a strong ecosystem of companies that rely on it to run their business. OpenStack is a set of interrelated software tools, capable to build and manage cloud computing platforms, controlling large pools of compute, storage and networking resources throughout a datacenter. Users can manage the OpenStack cloud through a web-based dashboard, command-line tools, APIs, a SDK, or via RESTful web services. For more information, please check the OpenStack website: https://www.openstack.org.
It is important to note that, while cloud computing resources are managed by OpenStack, all the other
resources in UFES testbed (e.g., SDN switches, and Intelligent Space) are managed in a uniform manner
by the SDN Controller. To achieve this uniform communication the southbound API is the OpenFlow
protocol. For devices that do not natively support OpenFlow, we will develop an OpenFlow Agent for
each device that translates the OpenFlow messages to the specific messages of each particular device.

At the moment, the implementation status of O2CMF is the following: the Cloud AM is in the last stages
of development and the Openstack infrastructure is deployed; the SDN AM and the SDN resource
control mechanisms are in the initial stages of development; and the Orchestration Layer is in the design
stage.

The user can request the provisioning of three main blocks of resources at UFES testbed:
- VM and LAN connectivity;
- Intelligent Space; and
- SDN switches.

The following subsections will give more details on how O2CMF provisions each one of these types of
resources. Note that RSPECs definitions aim to give an example of how we plan to provide this
resources, but final RSPECs may be modified.

8.2.1 VM provisioning and LAN connectivity

The experimenter can specify a set of VMs and L2 connectivity between them, according to the RSPEC
of Figure 19. The Cloud AM of Figure 18 receives this request and executes the following steps:

- Verifies the availability of the requested resources;
- Registers the reservation of the requested resources;
- Maps each bridge to a virtual network in OpenStack;
- Creates the VMs according to the specification;
- Connects the VMs to the related virtual networks.

```xml
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request" generated_by="jFed RSpec
Editor" generated="2017-07-06T15:32:51.372-03:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1" xmlns:jfed-
command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1" xmlns:jfed-
ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vlan/1"
xm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3 http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="false"
    component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
    <sliver_type name="vm">
      <disk_image name="urn:publicid:IDN+futebol.inf.ufes.br+image+ubuntu16"/>
      <flavor name="urn:publicid:IDN+futebol.inf.ufes.br+flavor+small"/>
    </sliver_type>
    <interface client_id="node0:if0">
      <ip address="192.168.0.1" netmask="255.255.255.0" type="ipv4"/>
    </interface>
  </node>
</rspec>
```
Figure 19: RSPEC for VMs and LAN connectivity
Internally, OpenStack assigns a VLAN id to each virtual network and automatically perform tag/untag operations every time a packet leaves and reaches the VM network interfaces. This mechanism ensures connectivity and traffic isolation.

In OpenStack, the physical servers are called compute nodes and their internal networking structure is described by Figure 20. In this structure, VMs appear in the upper layer, connected to a Linux bridge, called qbr, that implements security groups using iptables. These bridges are connected to the integration bridge, br-int, which is an Open vSwitch (OVS) bridge that interconnects the VMs running on the compute node. The br-int is, then, connected to the br-eth OVS bridge, which provides connectivity to the physical network interface cards. Communication between compute nodes can be achieved with L2 switches, which do not need to support OpenFlow.

In the example of Figure 20, when virtual machine VM1 sends an Ethernet frame to the physical network, it must pass through nine devices inside of the host: TAP eth0, Linux bridge qbrxxx, virtual ethernet pair (qvbxxx, qvoxxx), OVS bridge br-int, virtual ethernet pair (int-br-eth, phy-br-eth), OVS bridge br-eth, and, finally, to one of the physical network interface cards. When a virtual private network needs to be created, OpenStack configures the OVS of server nodes, using VLAN tags. Thus, a tenant gets a subnet and range of private IPs that are only accessible for a specific VLAN, where all his VMs are bridged. In the example of Figure 20, the physical network supports VLAN IDs 101 and 102 and there are two private virtual networks, net01 and net02, which have VLAN ids of 1 and 2. Flow rules in br-int and br-eth execute VLAN translation between physical and private VLANs.

VM instances that are in the same virtual network and are hosted in the same server node, as VM1 and VM2 in Figure 20, communicate directly via br-int bridge. On the other hand, if two VM instances are hosted in the same server node, but are not in the same private network, as VM1 and VM3 in Figure 20, their communication is intermediated by an OpenStack Controller node, which acts as a L3 router. As another example, if two VMs are in the same virtual network, but are hosted in different servers, the communication will pass through all the bridge layers, up and down, but without the intermediation of the OpenStack Controller node, as the communication does not involves L3 routing.
8.2.2 Intelligent Space provisioning

The Intelligent Space is composed by: four cameras, four access points, and a mobile robot unit. The experimenter requests selects the resources of the Intelligent space, according to the RSPEC of Figure 21. At the moment, it is not possible to share the Intelligent Space between experimenters. So as long as there is a resource allocation from the Intelligent Space, everything else is unavailable for that time slot.

The SDN AM receives the request and executes the following steps:

- Verifies the availability of Intelligent Space;
- Registers the reservation of the requested resources;
- Resets the cameras, access points and the robot to the default configuration.
- Sets the IP address of cameras and access points (just the Ethernet interface) according to the RSPEC parameters.

```
<?xml version='1.0'?><rspec xmlns="http://www.geni.net/resources/rspec/3" type="request" generated_by="jFed RSpec Editor" generated="2017-07-06T15:32:51.372-03:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmns:jfed-ssh-5 http://docs.ocselected.org/openstack-manuals/kilo/networking-guide
```
<camera client_id="camera0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <resolution h="1024" w="720" />
  <acquisition fps="5"/>
  <interface client_id="camera0:if0">
    <ip address="192.168.0.2" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</camera>

<camera client_id="camera1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <resolution h="1024" w="720" />
  <acquisition fps="5"/>
  <interface client_id="camera1:if0">
    <ip address="192.168.0.3" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</camera>

<camera client_id="camera2" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <resolution h="1024" w="720" />
  <acquisition fps="5"/>
  <interface client_id="camera2:if0">
    <ip address="192.168.0.4" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</camera>

<camera client_id="camera3" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <resolution h="1024" w="720" />
  <acquisition fps="5"/>
  <interface client_id="camera3:if0">
    <ip address="192.168.0.5" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</camera>

<access_point client_id="access_point0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <power val="1" />
  <channel val="10" />
  <interface client_id="access_point0:if0">
    <ip address="192.168.0.6" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</access_point>

<access_point client_id="access_point1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <power val="1" />
  <channel val="10" />
8.2.3 SDN switches provisioning

The Intelligent Space and the edge data center contain SDN switches that can be configured and assigned to a specific controller, according to the RSPEC of Figure 22. The SDN AM receives this request and executes the following steps:

- Verifies the availability of the resources;
- Registers the reservation of the requested resources;
- Sets up a VM running the OpenFlow controller with specified IP address;

```xml
<robot client_id="robot0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
  <interface client_id="robot0:if0">
    <ip address="192.168.0.10" netmask="255.255.255.0" type="ipv4"/>
  </interface>
</robot>

<rspec generated="2017-01-16T23:51:46.709Z" generated_by="jFed RSpec Editor" type="manifest" xmlns">
  <openflow:sliver description="OF request example" email=".">
    <openflow:controller type="primary" url="tcp:192.168.0.254:6633"/>
    <openflow:group name="packet">
      <openflow:datapath component_id="urn:publicid:IDN++openflow:futebol.inf.ufes.br++datapath+05:00:00:00:00:00:01" component_manager_id="urn:publicid:IDN+openflow:futebol.inf.ufes.br+authority+cm" dpid="05:00:00:00:00:00:00:01">
        <openflow:port name="GE0/2" num="2" type="cam" camid="1"/>
        <openflow:port name="GE0/3" num="3" type="cam" camid="2"/>
    </openflow:datapath>
  </openflow:group>
</rspec>
```

Figure 21: RSPEC for Intelligent Space
8.3 NFV Manager implementation

The NFV Manager is an extension of the Cloud AM that uses OpenStack to provide VM images that were previously built. These images are offered as VNFs in the Service Catalog. The RSPEC for VNF provisioning is shown in Figure 23.

```xml
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request" generated_by="jFed RSpec Editor" generated="2017-07-06T15:32:51.372-03:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vlan/1"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3 http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="false"
component_manager_id="urn:publicid:IDN+futebol.inf.ufes.br+authority+cm">
    <sliver_type name="vnf">
      <disk_image
name="urn:publicid:IDN+futebol.inf.ufes.br+image+sdn_controller">
        <flavor name="urn:publicid:IDN+futebol.inf.ufes.br+flavor+small"/>
      </disk_image>
    </sliver_type>
    <interface client_id="node0:if0">
      <ip address="192.168.0.1" netmask="255.255.255.0" type="ipv4"/>
    </interface>
  </node>
</rspec>
```
The Service Catalog offers the following VNFs (more details in the next section):

- **Image Processing Services**: (Pattern Tracker, Frame Converter, and Pose Estimation): The images from cameras are processed by a sequence of image processing functions in order to calculate the position of the robot.
- **Visual Servoing Application**: It shows the image from cameras and allows the user to select a target position/trajectory.
- **Robot Position Control Service**: It sends commands to control the robot movement as a result of the current position of the robot, gathered from the Image Processing Applications, and the target position/trajectory, defined by the experimenter in the Visual Servoing Application. It also provides information about robot position that can be used to perform other tasks, such as handover management.
- **Camera Gateway and Agent**: It receives OpenFlow commands from the SDN Controller and translates them into native configuration commands to the camera.
- **Broker**: It enables the communication between the Image Processing Applications and the Robot Position Control Application using a publish/subscribe scheme.
- **Experiment Orchestrator**: It provides a single orchestration interface for the experimenter that enables the converged control of the experiment resources. It interacts with the VIM based on OpenStack, the SDN Controller, and other VNFs to gather information about the infrastructure and to perform orchestration decisions.
- **Experiment SDN Controller**: It is responsible for controlling the OpenFlow switches, cameras, and APs.
- **Measurement and Monitoring Application**: It monitors the resources and provides information about them.
- **Experiment Control Application**: It enables the experimenter to write scripts to execute an experiment using the orchestration API.

### 8.4 Validation of the experiment control and Service orchestration

This section describes how the user interacts with the resources provisioned via federation to execute his/her experiment.

#### 8.4.1 Showcase description

The UFES testbed allows a wide variety of experiments in the areas of Software-Defined Networking, Cloud robotics, Mobile Edge Computing, Network Function Virtualization, and Wireless Mobility. As a showcase of the testbed experimentation capabilities, we have designed an experiment that shows how a converged control framework can manage networking and cloud infrastructures to enable the real time, remote control of robots in an indoor scenario as a visual servoing application. Visual servoing, also known as vision-based robot control is a technique which uses feedback information extracted from a vision sensor (e.g., cameras) to control the motion of a robot. The combination of position and orientation is referred to as the pose of an object.
Figure 24 shows the software architecture used in the Intelligent Space in order to provide a visual servoing application. This application enables users to control the target position of a robot in a room by clicking on an image screen provided by the cameras. In this architecture, all the communication between the system components goes through a message broker (using RabbitMQ message broker software), and a SoA (Service oriented Architecture) approach is used. The system has basically three types of elements: services, gateways and applications. Services are entities that expose software computation to the other elements (e.g., Robot Position Control, Pattern Tracker, Frame Converter, and Pose Estimation). Gateways expose physical resources as a service, in other words, they enable users to control physical devices like cameras, robots, speakers and others, using well defined interfaces. Applications expose high-level services to users through a graphical interface.

The main components of the architecture are described as following:

- **Robot Position Control**: controls the robot position to match the target position selected by the user;
- **Pattern Tracker**: detects the pattern in the image and returns a vector of points in the image coordinate reference system;
- **Frame Converter**: converts image points to the global coordinate reference system;
- **Position Estimation**: estimates the robot pose using the pattern points in the global coordinate reference system;
- **Camera Gateway**: receives RAW images, and compresses them;
- **Robot Gateway**: receives and executes velocities commands in the robot platform.

In Figure 25, a diagram presents the message flow in the system to enable the Visual Servoing Application. The Robot Position Control Service intermediates the communication between modules and it is responsible to decide the next processing step. In the first step, the image from cameras is captured and converted by the Camera Gateway. Then, the Robot Position Control sends this information to parallel instances of the Pattern Tracker. The information is consolidated by the Frame Converter, and the Pose Estimation calculates the robot pose. Finally, the Robot Position Control sends a movement command to the Robot Gateway. After a time window, the process starts again and new images are captured and processed.
8.4.2 Experiment Control Application

The experimenter writes his/her experiment script in the Experiment Control Application according to his/her algorithm based on calls to the REST API provided by the Orchestrator.

The experiment starts with a provisioning phase using jFed tool, which comprises the following steps:

- The experimenter requests the Intelligent Space (see Figure 21);
- The experimenter requests the switch ports in the OpenFlow switch that are connected to the cameras and APs in the Intelligent Space (see Figure 22);
- The experimenter requests the switch ports that interconnect the OpenFlow switch connected to the Intelligent Space and the OpenFlow switch connected to the edge datacenter (see Figure 22);
- The experimenter requests the necessary VNFs as pre-built VMs that are available in the Service Catalog (see Figure 23); and
- The Cloud AM and SDN AM from O2CMF at UFES testbed receive the requests and provision the resources, if available (see Figure 18).

After the provisioning phase, the experimenter can control its resources to perform the experiment by developing an application that can perform the following actions:

- To set wireless AP parameters: power and channel;
- To set camera parameters: fps, color system, and resolution;
- To retrieve the robot current position;
- To control the handover based on the robot location;
- To retrieve monitoring and measurement information from the resources;
- To perform vertical scaling of VNF components (modification of VM resources, such as CPU cores or memory).

8.4.3 Wireless Mobility

The APs are physical servers with Linux OS and hostapd daemon installed, which enables us to use Linux servers as APs. All the APs are configured with the same BSSID, creating a single global BSSID, but with different channels for each AP. Initially, the robot can select the AP for association, but, if the selected AP is not ideal with respect to the robot position, the SDN Controller can send a command to perform a handover. To this end, the SDN Controller performs the authentication of the robot on the destination AP and instructs the source AP to send a beacon message with a channel switch advertisement (CSA) to the robot. In this way, the robot performs the handover by changing only its operating channel and does not need to perform authentication and verification on the new AP.
In addition, when the robot connects with a new AP after the handover, the SDN controller has to change the route to forward the messages to the AP that is currently associated with the robot. The route to the correct AP is defined by a flow rule where the destination IP address matches the robot IP address and the action specifies the output port for the designated AP. This rule is modified when the robot registers to a new AP and the SDN Controller receives the event. This change is made using an OpenFlow Protocol FlowMod message, which modifies the output port for the new AP.

### 8.4.4 Experiment SDN Controller

The SDN Controller is responsible for managing all the SDN infrastructure of the experiment using a southbound interface based on OpenFlow, including non-native OpenFlow devices that depend on the development of OpenFlow Agents to receive OpenFlow messages. It is also responsible for providing northbound interfaces to interact with the Orchestrator.

When receiving the calls from the Orchestrator, the SDN Controller sends OpenFlow messages to the respective devices, if they natively support OpenFlow (e.g., OpenFlow switches and physical servers with OVS), or to the SDN Agents that enable the device to support OpenFlow. The following SDN Agents will be developed:

- **Cameras**: the SDN Agent will be a VM in the datacenter with Linux OS that receives OpenFlow commands using a OVS virtual switch and translates them into native commands to the cameras. The cameras receive IP addresses in the same network of the VMs hosted in the datacenter.
- **APs**: Since the APs are physical servers with Linux OS, the SDN Agent is an application that receives OpenFlow commands using a OVS virtual switch and translates them into native commands to a wireless network interface. The physical servers of the APs receive IP addresses in the same network of the VMs hosted in the datacenter.

### 8.4.5 Experiment Orchestrator

The experiment orchestrator is responsible for providing the high level northbound interfaces that enable the experimenter to access information about the infrastructure and to send commands to actuate on it.

To enable wireless mobility experiments, the Orchestrator interacts with the SDN Controller and provides a REST API that implements the following calls.

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /ap/&lt;id&gt;/status</td>
<td>Returns general information about the access point owning the given id.</td>
<td><code>{ &quot;id&quot;: 1, &quot;hostname&quot;: &quot;ap00&quot;, &quot;interfaces&quot;: { &quot;lan&quot;: { &quot;mac&quot;: &quot;00:00:00:00:00:00&quot;, &quot;ipv4&quot;: &quot;192.168.1.1&quot;, &quot;ipv6&quot;: &quot;abcd::ef1/64&quot;, }, &quot;wifi&quot;: { &quot;mac&quot;: &quot;00:00:00:00:00:00&quot;, &quot;ipv4&quot;: &quot;172.16.30.1&quot;, &quot;ipv6&quot;: &quot;abcd::ef1/64&quot;, } }</code></td>
</tr>
</tbody>
</table>
| "channel": 1,  
| "power": 10,  
| "uptime": 10102010,  
| "port": 1,  
| "active": true,  
| "devices": [ "1", "2", "3" ],  
| }
### GET /device/<id>/status

Returns general information about the device (that is, the robot) owning the given id.

<table>
<thead>
<tr>
<th>id</th>
<th>hostname</th>
<th>interfaces</th>
<th>channel</th>
<th>ap</th>
<th>uptime</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dev00</td>
<td>{ lan: { ipv4: &quot;192.168.1.1/24&quot;, ipv6: &quot;abcd::ef1/64&quot; } }</td>
<td>1</td>
<td>1</td>
<td>10102010</td>
<td>true</td>
</tr>
</tbody>
</table>

### PUT /ap/<id>/channel/10

Changes to the given value the wireless channel of the access point owning the informed id. Returns the id of the access point and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### PUT /device/<id>/channel/10

Changes to the given value the wireless channel of the device (that is, the robot) owning the informed id. Returns the access point’s id and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### PUT /ap/<id>/power/10

Changes to the given value the transmit power of the access point owning the informed id. Returns the access point’s id and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### GET /device/<id>/discovery

Returns the id of the access point which the device owning the given id is connected.

<table>
<thead>
<tr>
<th>id</th>
<th>ap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### PUT /switch/ap/<id>/device/1

Informs to the OpenFlow switch which access point the device is associated.

<table>
<thead>
<tr>
<th>id</th>
<th>ap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The experimenter is also able to vary the input network traffic from the cameras based on the following calls implemented by the Orchestrator, which interact with the SDN Controller to execute the commands.

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
<th>Return</th>
</tr>
</thead>
</table>

© FUTEbol Consortium 2016-2019
GET /camera/<id>/status

Returns general information about a camera owning the given id.

<table>
<thead>
<tr>
<th>id</th>
<th>hostname</th>
<th>interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev00</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;lan&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ipv4&quot;: &quot;192.168.1.1/24&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ipv6&quot;: &quot;abcd::ef1/64&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;fps&quot;: 10,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;color&quot;: &quot;rgb&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;resolution&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;height&quot;: 720,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;width&quot;: 1024</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;timestamp&quot;: 10102010,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;active&quot;: true,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

PUT /camera/<id>/fps/10

Changes to the given value the fps rate of the camera owning the informed id. Returns the camera’s id and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PUT /camera/<id>/resolution/720,1024

Changes to the given value the resolution of the camera owning the informed id. Returns the camera’s id and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PUT /camera/<id>/color/rgb

Changes to the given value the color system of the camera owning the informed id. Returns the camera’s id and a status code of the action (1-success/0-fail).

<table>
<thead>
<tr>
<th>id</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Additionally, the Orchestrator enables access to some information about the resources by interacting with the Monitoring and Measurement Application and providing a REST API that implements the following calls:

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
<th>Return</th>
</tr>
</thead>
</table>

© FUTEBOL Consortium 2016-2019
<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Description</th>
<th>Sample Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /device/&lt;id&gt;/position</td>
<td>Returns the position of the device (that is, the robot) owning the given id.</td>
<td>{ &quot;id&quot;: 1, &quot;position&quot;: { &quot;X&quot;: 10, &quot;Y&quot;: 10, } }</td>
</tr>
<tr>
<td>GET /device/&lt;id&gt;/error</td>
<td>Returns the position error of the device (that is, the robot) owning the given id. The error is defined as the distance between the current position and the target trajectory.</td>
<td>{ &quot;id&quot;: 1, &quot;error&quot;: 0.1, }</td>
</tr>
<tr>
<td>GET /vm/&lt;id&gt;/usage</td>
<td>Returns resource usage information about the VM owning the given id.</td>
<td>{ &quot;id&quot;: 1, &quot;cpu&quot;: 0.6, &quot;memory&quot;: 0.6, &quot;bandwidth&quot;: 700, }</td>
</tr>
<tr>
<td>GET /vm/&lt;id&gt;/serverusage</td>
<td>Returns resource usage information about the server that hosts the VM owning the given id.</td>
<td>{ &quot;id&quot;: 1, &quot;cpu&quot;: 0.6, &quot;memory&quot;: 0.6, &quot;bandwidth&quot;: 700, }</td>
</tr>
<tr>
<td>GET /vnf/&lt;id&gt;/usage</td>
<td>Returns resource usage information about the VNF owning the given id.</td>
<td>{ &quot;id&quot;: 1, &quot;cpu&quot;: 0.6, &quot;memory&quot;: 0.6, &quot;bandwidth&quot;: 700, }</td>
</tr>
<tr>
<td>GET /vnf/&lt;id&gt;/serverusage</td>
<td>Returns resource usage information about the server that hosts the VNF owning the given id.</td>
<td>{ &quot;id&quot;: 1, &quot;cpu&quot;: 0.6, &quot;memory&quot;: 0.6, &quot;bandwidth&quot;: 700, }</td>
</tr>
<tr>
<td>GET /vnf/&lt;id&gt;/responsetime</td>
<td>Returns response time information about the VNF owning the given id. Response time is defined as the total amount of time the VNF takes to respond to a service request.</td>
<td>{ &quot;id&quot;: 1, &quot;responsetime&quot;: 10, }</td>
</tr>
</tbody>
</table>
8.4.6 Service Orchestration

In addition to the previously described REST APIs, the experimenter will be able to interact with the Experiment Orchestrator to specify service orchestration functionalities using a script in TOSCA (Topology and Orchestration Specification for Cloud Applications) language. More specifically, UFES testbed will use the TOSCA NFV profile\(^6\) that specifies a NFV data model using TOSCA language. We plan to implement orchestration functionalities that enable horizontal scaling and SFC.

The horizontal scaling is the ability to increment (scale out) or decrement (scale in) the number of instances of a particular VNF. Moreover, the use of policies enables scaling in an automatic manner. Figure 26 shows an example of how to define a threshold based on the CPU utilization (cpu utilization greater than 50%), an alarm that monitors the defined threshold, and a policy that specifies an action when the threshold is reached (increment the number of instances), using TOSCA language\(^7\).

---

\(^6\) http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html
\(^7\) https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnfd/tosca-vnfd-alarm-scale.yaml
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
description: Demo example

metadata:
  template_name: sample-tosca-vnfd
topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            disk_size: 1 GB
            mem_size: 512 MB
            num_cpu: 2
          properties:
            image: cirros-0.3.5-x86_64-disk
            mgmt_driver: noop
            availability_zone: nova
            metadata: {metering.vnf: SG1}
    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
          node: VL1
        - virtualBinding:
          node: VDU1
    VDU2:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            disk_size: 1 GB
            mem_size: 512 MB
            num_cpu: 2
          properties:
            image: cirros-0.3.5-x86_64-disk
            mgmt_driver: noop
            availability_zone: nova
            metadata: {metering.vnf: SG1}
    CP2:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
anti_spoofing_protection: false
requirements:
  - virtualLink:
    node: VL1
  - virtualBinding:
    node: VDU2

VL1:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: net_mgmt
    vendor: Tacker

policies:
  - SP1:
    type: tosca.policies.tacker.Scaling
    properties:
      increment: 1
      cooldown: 120
      min_instances: 1
      max_instances: 3
      default_instances: 2
      targets: [VDU1, VDU2]

  - vdu_cpu_usage_monitoring_policy:
    type: tosca.policies.tacker.Alarming
    triggers:
      vdu_hcpu_usage_scaling_out:
        event_type:
          type: tosca.events.resource.utilization
          implementation: ceilometer
          metrics: cpu_util
        condition:
          threshold: 50
          constraint: utilization greater_than 50%
          period: 600
          evaluations: 1
          method: avg
          comparison_operator: gt
          metadata: SG1
          actions: [SP1]
An SFC is represented by a VNF-FG which describes how VNFs are interconnected and how the traffic flows through the graph. The following example specifies a VNF-FG using TOSCA language (https://github.com/openstack/tacker/blob/master/samples/tosca-templates/vnffgd/tosca-vnffgd-sample.yaml). The example of Figure 27 shows how to define a SFC that steers the HTTP traffic from a source to the sequence VNFD1 and VNFD2. The SFC functionality will be demonstrated in a specific NFV experiment, because the remote control of robots experiment would require deep changes in the application level, such as replacing the message broker by the SFC mechanism.

<table>
<thead>
<tr>
<th>tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>description: Sample VNFFG template</td>
</tr>
<tr>
<td>topology_template:</td>
</tr>
<tr>
<td>description: Sample VNFFG template</td>
</tr>
<tr>
<td>node_templates:</td>
</tr>
<tr>
<td>Forwarding_path1:</td>
</tr>
<tr>
<td>type: tosca.nodes.nfv.FP.Tacker</td>
</tr>
<tr>
<td>description: creates path (CP12-&gt;CP22)</td>
</tr>
<tr>
<td>properties:</td>
</tr>
<tr>
<td>id: 51</td>
</tr>
<tr>
<td>policy:</td>
</tr>
<tr>
<td>type: ACL</td>
</tr>
<tr>
<td>criteria:</td>
</tr>
<tr>
<td>- network_src_port_id: 640dfd77-c92b-45a3-b8fc-22712de480e1</td>
</tr>
<tr>
<td>- destination_port_range: 80-1024</td>
</tr>
<tr>
<td>- ip_proto: 6</td>
</tr>
<tr>
<td>- ip_dst_prefix: 192.168.1.2/24</td>
</tr>
<tr>
<td>path:</td>
</tr>
<tr>
<td>- forwarder: VNFD1</td>
</tr>
<tr>
<td>capability: CP12</td>
</tr>
<tr>
<td>- forwarder: VNFD2</td>
</tr>
<tr>
<td>capability: CP22</td>
</tr>
<tr>
<td>groups:</td>
</tr>
<tr>
<td>VNFFG1:</td>
</tr>
<tr>
<td>type: tosca.groups.nfv.VNFFG</td>
</tr>
<tr>
<td>description: HTTP to Corporate Net</td>
</tr>
<tr>
<td>properties:</td>
</tr>
<tr>
<td>vendor: tacker</td>
</tr>
<tr>
<td>version: 1.0</td>
</tr>
<tr>
<td>number_of_endpoints: 2</td>
</tr>
<tr>
<td>dependent_virtual_link: [VL12,VL22]</td>
</tr>
<tr>
<td>connection_point: [CP12,CP22]</td>
</tr>
<tr>
<td>constituent_vnfs: [VNFD1,VNFD2]</td>
</tr>
<tr>
<td>members: [Forwarding_path1]</td>
</tr>
</tbody>
</table>

*Figure 27: SFC Example*
9 Conclusions

The FUTEBOL control framework described in this deliverable provides the mechanisms for experimenters to provision and orchestrate their experiments.

This is done by first adding new testbeds to the existing federation frameworks to provision experiment slices. New SDR, IoT, WiFi and LTE Shared Access resources have been added to the federation, both in Brazil and Europe. To do this, extensions to existing CFs such as CBTM and VTAM were done. Also, a new framework based on OpenStack was developed to enable VNF and SFC functionality in testbeds that adopt it.

Besides the work of adding new testbeds and resources to the federation, new control orchestration mechanisms allow experimenters to quickly develop experiments where dynamic network orchestration is necessary.

To do this, a new container management system is also being currently developed in FUTEBOL: COPA. This container management system aims to make easier for experimenters to migrate virtual functions across their experiments, even multi-testbeds experiments.
References