**Grant Agreement No.: 101015857**
**Research and Innovation action**
**Call Topic: ICT-52-2020: 5G PPP - Smart Connectivity beyond 5G**

Secured autonomic traffic management for a Tera of SDN flows

D5.1: Testbed Setup and Prototype Integration Report

| Deliverable type | R (Report) |
|---|---|
| Dissemination level | PU |
| Due date | 31.12.2021 |
| Submission date | 31.12.2021 |
| Lead editor | Carlos Natalino (CHAL) |
| Authors | Paolo Monti (CHAL), Lluis Gifre, Laia Nadal, Javier Vilchez, Francisco Vázquez (CTTC), Alberto Mozo (UPM), Antonio Pastor (TID), Min Xie (TNOR), Sergio González Javier Moreno (ATOS) |
| Reviewers | Daniel King (ODC), Håkon Lønsethagen (Telenor) |
| Quality check team | Adrian Farrel (ODC), Daniel King (ODC) |
| Work package | WP5 |

*Abstract*

This deliverable reports the ongoing efforts *i)* towards the definition of the scenarios for the experimental activities, and *ii)* to introduce and explain the tools chosen to coordinate the prototype development and integration process. The scenarios defined in this deliverable specify which features of the TeraFlow OS will be most relevant, and which partners' facilities will be used to set up and test the functions. The set of tools is accompanied by a set of processes and guidelines to be followed during the development process. Finally, installation procedures for the current TeraFlow OS version are provided. As an annex, an updated inventory of the partners facilities initially included in MS5.1 is provided.

[End of abstract]

## Disclaimer

This report contains material which is the copyright of certain TeraFlow Consortium Parties and may not be reproduced or copied without permission.

All TeraFlow Consortium Parties have agreed to publication of this report, the content of which is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License[1].

Neither the TeraFlow Consortium Parties nor the European Commission warrant that the information contained in the Deliverable is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using the information.

CC BY-NC-ND 3.0 License – 2021 - 2023 TeraFlow Consortium Parties

## Acknowledgment

The research conducted by TeraFlow receives funding from the European Commission H2020 programme under Grant Agreement No 101015857. The European Commission has no responsibility for the content of this document.

## Revision History

| Revision | Date | Responsible | Comment |
|---|---|---|---|
| **0.1** | 16.07.2021 | Editor | Initial structure of document |
| **0.2** | 10.09.2021 | Editor | Refine the structure of the document |
| **0.3** | 29.10.2021 | Editor | Inclusion of pointers to the contributors of the document |
| **0.4.1** | 22.11.2021 | ATOS | Inclusion of the ATOS contribution to Section 3 |
| **0.4.2** | 26.11.2021 | CTTC | Inclusion of the CTTC contribution to Sections 2 and 3 |
| **0.4.3** | 29.11.2021 | UPM | Inclusion of the UPM contribution to Section 2 |
| **0.4.4** | 01.12.2021 | TID | Review of the Section 2.3 |
| **0.5** | 02.12.2021 | Editor | Review of the partners contribution |
| **0.6** | 07.12.2021 | CHAL | Internal review |
| **0.6.1** | 20.12.2021 | Håkon Lønsethagen | Review |
| **0.6.2** | 21.12.2021 | Daniel King | Review |
| **0.7** | 22.12.2021 | Editor | Address reviewer's comments |
| **0.8** | 25.12.2021 | Adrian Farrel | Quality review |
| **1.0** | 29.12.2021 | Editor | Final version ready for submission |

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

# EXECUTIVE SUMMARY

This deliverable summarizes the activities of WP5 during the first year of the TeraFlow project. The objective of this document is to describe the ongoing efforts towards *i)* the definition of the scenarios for the experimental activities, the prototype features deemed relevant in each one of the scenarios, and the testbeds that the partners have been setting up to test the prototype functionalities; and *ii)* the introduction and explanation of the tools chosen to coordinate the prototype development and integration process.

The document starts with an introductory section that highlights the purpose of this deliverable, its relationship with other deliverables, and a detailed description of the structure of this document. The second section presents an overview of the TeraFlow OS architecture. The third section presents the scenarios in which the TeraFlow OS prototype will be tested, validated, and benchmarked. More specifically, we have three scenarios the "Autonomous network Beyond 5G" scenario using facilities at Telefónica, Infinera, and Ubitech; the "Automotive" scenario using facilities at CTTC, NEC, and Telenor; and finally, the "CyberSecurity" scenario using facilities at Telefónica, UPM, and Chalmers. The fourth section of this document introduces the solutions adopted for the development process of the TeraFlow OS prototype, with special attention to the Continuous Integration/Continuous Delivery (CI/CD) pipeline implemented. The section also describes the installation procedures that need to be followed by a user who wants to develop further or test the TeraFlow OS prototype.

Finally, this document concludes with several remarks and an Annex describing the premise facilities available at the project partners.

© 2021 - 2023 TeraFlow Consortium Parties 

# Table of contents

Executive Summary ..... 3
List of Figures ..... 6
List of Tables ..... 7
Abbreviations ..... 8
1. Introduction ..... 10
1.1. Purpose ..... 10
1.2. Relationship to other Deliverables ..... 10
1.3. Structure ..... 10
2. Architecture Overview ..... 12
3. Scenarios ..... 13
3.1. Scenario 1: Autonomous Network Beyond 5G ..... 13
3.1.1. TeraFlow OS Prototype Features ..... 14
3.1.2. Testbed Setup ..... 15
3.2. Scenario 2: Automotive ..... 16
3.2.1. TeraFlow OS Prototype Features ..... 17
3.2.2. Testbed Setup ..... 18
3.3. Scenario 3: Cybersecurity ..... 18
3.3.1. TeraFlow OS Prototype Features ..... 20
3.3.2. Testbed Setup ..... 21
4. Preliminary Integration Results ..... 22
4.1. Reference Component Architecture ..... 22
4.1.1. Exposed Ports per Component ..... 24
4.1.2. Health RPC ..... 25
4.1.3. Metrics Exporter ..... 25
4.2. CI/CD Pipeline ..... 27
4.2.1. Introduction to CI/CD ..... 27
4.2.2. Tools ..... 27
4.2.3. TeraFlow Methodology for CI/CD ..... 32
4.2.4. Good Practices and Hints ..... 34
4.3. Installation Procedures ..... 37
5. Conclusions and Next Steps ..... 39
6. Annex 1: Inventory of Facilities ..... 40
6.1. CTTC ..... 40
6.2. Telefónica ..... 41

© 2021 - 2023 TeraFlow Consortium Parties 

6.2.1. The Mouseworld Laboratory ........ 41
6.2.2. The Future Network Laboratory ........ 43
6.3. Infinera ........ 44
6.4. SIAE Microelettronica ........ 44
6.5. NEC Laboratories ........ 46
6.6. ATOS ........ 47
6.7. Telenor ........ 48
6.8. Chalmers University of Technology ........ 49
6.9. UBITECH ........ 50
References ........ 52

© 2021 - 2023 TeraFlow Consortium Parties 

# List of Figures

Figure 1 TeraFlow OS general architecture ..... 12
Figure 2 Autonomous Network Beyond 5G scenario ..... 14
Figure 3 TeraFlow components used in the autonomous network beyond 5G use case ..... 15
Figure 4 Automotive scenario ..... 17
Figure 5 TeraFlow components involved in the automotive use case ..... 18
Figure 6 Cybersecurity scenario and threats ..... 19
Figure 7 Distributed Cybersecurity Component architecture ..... 20
Figure 8 TeraFlow components used in the cybersecurity use case ..... 21
Figure 9 TeraFlow OS general component architecture ..... 23
Figure 10 gRPC Health service data model ..... 25
Figure 11 Screenshot of Prometheus UI ..... 26
Figure 12 Dashboards of the TeraFlow organization and repository in GitLab ..... 29
Figure 13 Functional Architecture in TeraFlow to enable CI/CD pipeline ..... 32
Figure 14 Overall overview of the TeraFlow CI/CD Infrastructure ..... 34
Figure 15 GitFlow graph representing an exemplary structure involving the five types of branches ..... 35
Figure 16 TeraFlow GitLab repository structure ..... 37
Figure 17 Adrenaline testbed ..... 41
Figure 18 The Mouseworld facility ..... 42
Figure 19 Testing setup at the Espoo/Finland Lab ..... 44
Figure 20 Architecture including an SDN controlled Microwave network ..... 45
Figure 21 An example of L3VPN with MW links ..... 45
Figure 22 Distributed ledger and its components ..... 46
Figure 23 The ATOS telecom testbed ..... 47
Figure 24 Telenor's testbed setting ..... 48
Figure 25 Setup with the Volta Platform software ..... 48
Figure 26 Transport network slices example ..... 49
Figure 27 Overall architecture of the clusters ..... 50
Figure 28 UBITECH's testbed ..... 51

# List of Tables

Table 1 - Ports exposed per component ........ 25
Table 2 - Git-based repository tools comparison ........ 28
Table 3 - CI/CD frameworks side-by-side comparison ........ 31
Table 4 - Configuration variables supported by deploy_in_kubernetes.sh ........ 38

© 2021 - 2023 TeraFlow Consortium Parties

# Abbreviations

| | |
|---|---|
| **5G-PPP** | 5G Infrastructure Public Private Partnership |
| **API** | Application Programming Interface |
| **B5G** | Beyond 5G |
| **CCAM** | Cooperative, Connected and Automated Mobility |
| **CD** | Continuous Delivery |
| **CI** | Continuous Integration |
| **CO** | Central Office |
| **DC** | Data Centre |
| **DL** | Deep Learning |
| **DLT** | Distributed Ledger Technology |
| **DPI** | Deep Packet Inspection |
| **E2E** | End-to-End |
| **gRPC** | gRPC Remote Procedure Call |
| **HTTP** | Hypertext Transfer Protocol |
| **IETF** | Internet Engineering Task Force |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IT** | Information Technology |
| **JSON** | JavaScript Object Notation |
| **L2** | Layer 2 |
| **L2NM** | Layer 2 Network YANG Model |
| **L2VPN** | Layer 2 Virtual Private Network |
| **L3** | Layer 3 |
| **L3NM** | Layer 3 Network YANG Model |
| **L3VPN** | Layer 3 Virtual Private Network |
| **MANO** | Management and Orchestration |
| **MEC** | Multi-access Edge Computing |
| **ML** | Machine Learning |
| **MW** | Microwave |
| **NBI** | North-Bound Interface |
| **NFV** | Network Function Virtualization |

© 2021 - 2023 TeraFlow Consortium Parties

| | |
|---|---|
| **OLS** | Open Line System |
| **ONF** | Open Networking Foundation |
| **OPM** | Optical Performance Monitoring |
| **OS** | Operating System |
| **OSM** | Open Source MANO |
| **OSS/BSS** | Operation Support System/Business Support System |
| **OTA** | Over-the-Air |
| **PDL** | Permissioned Distributed Ledger |
| **RPC** | Remote Procedure Call |
| **SaaS** | Software-as-a-Service |
| **SBI** | South-Bound Interface |
| **SDN** | Software-Defined Networking |
| **SDO** | Standards Defining Organization |
| **SLO** | Service Level Objective |
| **TAPI** | Transport API |
| **TLS** | Transport Layer Security |
| **TR** | Technical Reference |
| **VNF** | Virtualized Network Functions |
| **VPN** | Virtual Private Network |
| **XML** | eXtensible Markup Language |
| **YAML** | YAML Yet Another Markup Language |

# 1. Introduction

TeraFlow delivers the next generation open-source cloud native Software-Defined Networking (SDN) controller providing efficient, reliable, secure, and scalable control for B5G networks. In this context, ensuring the ability of the TeraFlow Operating System (OS) to interact with existing equipment and leverage existing protocols to control such devices is paramount. WP5 is responsible for performing the TeraFlow OS integration, followed by experimentation, validation, and evaluation using a range of benchmark indicators. The project leverages the infrastructure available at the partners' premises to build testbed setups, realizing three scenarios described in this deliverable. Moreover, given the extensive software development efforts within the project and the highly distributed nature of the prototype implementation process, it is important to leverage state-of-the-art techniques, processes and tools to ensure the stability and good integration of the different components developed.

## 1.1. Purpose

The purpose of deliverable D5.1 is threefold. The first objective of the deliverable is to present the scenarios identified by the project as representative for the Beyond 5G (B5G) paradigm, identify the challenges posed by each of the scenarios, and explain how the TeraFlow OS is tackling them. Furthermore, the deliverable describes the specific features that the TeraFlow OS prototype must have to fulfil the scenario requirements. Finally, the deliverable describes how the scenarios relate to the use cases defined in D2.1. The second objective of the deliverable is to describe the reference component architecture used by the partners as a baseline for their implementation work. Furthermore, the deliverable presents the Continuous Integration/Continuous Delivery (CI/CD) pipeline adopted for code development and integration. The deliverable also provides information and pointers about the installation procedures to deploy the most recent version of the TeraFlow OS. The third objective of the deliverable is to present an inventory of the equipment available at the partners' premises. It also details how this equipment will be used for the experimentation and performance evaluation in years 2 and 3 of the project.

## 1.2. Relationship to other Deliverables

D5.1 takes inputs from MS2.1, where the use cases were initially defined, and a preliminary architecture of the TeraFlow OS was proposed. It also takes input from the MS5.1, where the partners' premises were first reported. Moreover, the TeraFlow OS components used to tackle the technical challenges in each one of the scenarios reported in Section 3 are reported in detail in D3.1 and D4.1.

## 1.3. Structure

This deliverable is structured as follows. Section 2 presents an overview of the architecture defined for the TeraFlow OS. Section 3 describes the scenarios used by the TeraFlow project to validate and evaluate the TeraFlow OS, and highlights the challenges presented by those scenarios explaining how the project plans to tackle them. For each scenario, the most important TeraFlow OS features are highlighted, and a list of the premises to be used to realize the scenario is specified.

Section 4 reports the preliminary TeraFlow OS integration results. First, a reference component architecture that the partners can follow to develop their own TeraFlow OS components is set out.

Then, the CI/CD pipeline developed for the TeraFlow OS development activities is described. Finally, the installation procedure for installing the TeraFlow OS is detailed.

The deliverable finishes with Section 5, where a few concluding remarks and next steps are presented. Annex 1 (Section 6) contains the description of the equipment that will be used for the experimental activities at the partners' premises.

# 2. Architecture Overview

The TeraFlow OS architecture is presented in detail in D2.1. Here, we briefly describe its main characteristics. Figure 1 shows the proposed architecture, consisting of a set of feature-dedicated micro-services implemented following the cloud native architecture concept. The micro-services can be classified into two groups: *core components* able to interact through a set of internal data models, and *netApps* consuming a set of features provided by the core components via standardized APIs. Thanks to the development of ad-hoc Application Programming Interfaces (APIs), the TeraFlow OS can also interact with state-of-the-art existing open-source Network Function Virtualization (NFV) and Multi-access Edge Computing (MEC) orchestrators, as well as Operation Support System/Business Support System (OSS/BSS). For a complete description of the TeraFlow OS architecture and micro-services, please refer to D2.1.

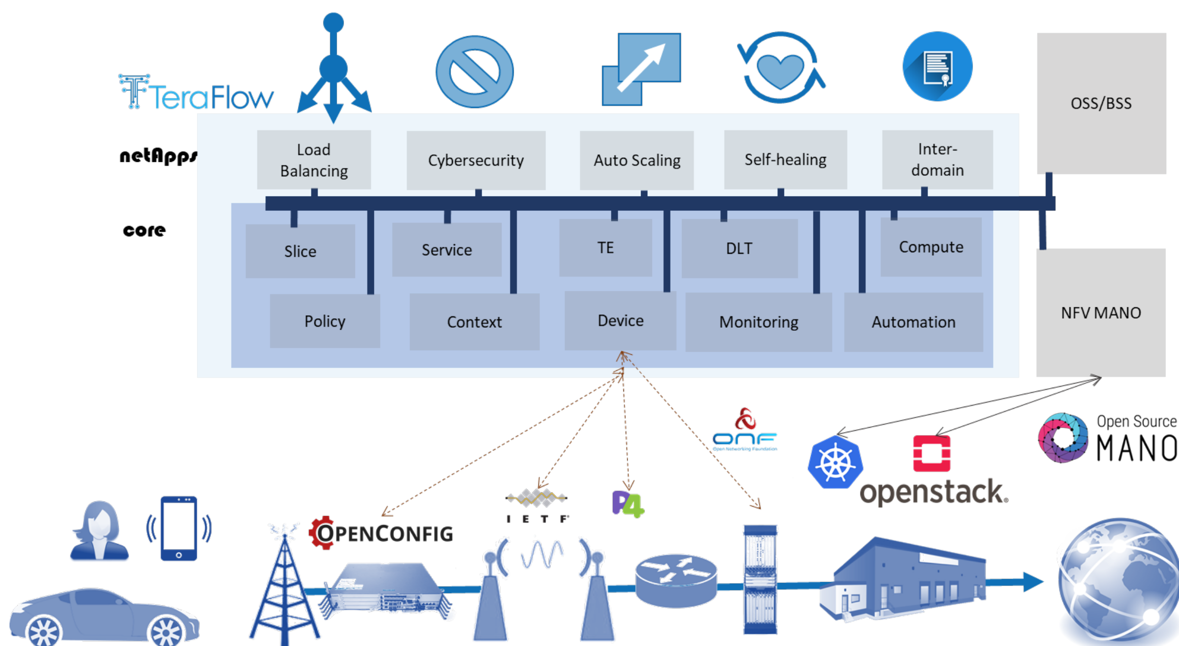



Figure 1 TeraFlow OS general architecture

The development of the TeraFlow OS relies on a CI/CD pipeline allowing partners to continuously develop, test, and integrate their components with other components developed by other partners. More details on the CI/CD pipeline are provided in Section 4.2.

The source code of the TeraFlow OS SDN Controller is available online at the TeraFlow OS SDN Controller GitLab repository [2]. Two main branches have been created. The *master branch* contains the latest code release, while the *develop branch* contains the code with the latest features still under active testing, integration, and validation. MS3.2 and MS4.1 provide a comprehensive description of the code developed so far.

An integral part of the TeraFlow OS SDN Controller is the service model it exposes, and that is assumed by the relevant TeraFlow Services North-Bound Interface (NBI). The “Transport Network Slice” service concept and its service model are at the core of these APIs and the Services NBI, enabling the associated service provisioning, configuration, monitoring, and SLA management. Scenario 1 described in Section 3.1 will elaborate on these artifacts and provide further motivation and description of the TeraFlow OS SDN Controller and architecture.

# 3. Scenarios

This section presents an overview of three scenarios identified by the project to represent some of the challenges posed by B5G networks. For each one of the scenarios, we briefly describe the main technical challenges, and the features that will be required (i.e., from the TeraFlow OS point of view) to overcome such challenges, the TeraFlow OS component that will be developed to provide these features, and, finally, the use cases of interest that will be investigated to validate and benchmark the performance of the TeraFlow OS prototype.

## 3.1. Scenario 1: Autonomous Network Beyond 5G

With 5G networks comes the opportunity to deploy new services in the network in an automated manner. Network operators can migrate to 5G based on templates for services and network slices hard-coded into their systems. In this case, each service and/or network slice selects its deployment type from a list of predefined specifications, defining specific network resources and having requirements or constraints.

However, this approach does not scale for B5G scenarios, where the network should adapt to the end users' needs in a dynamic and on-demand manner. This means that the network (operated by the network slice controller) should compute a deployment plan (considering relevant and needed network service functions) together with a service provisioning and configuration plan. This needs to be done dynamically and intelligently to match the requested service, provide adaptation capabilities during the service operation, and relate the requested services to the specific underlying network resources that are offered and available. If most of the services will require resources from different domains, these network resources need to be orchestrated to provide multi-layer and multi-domain services. Network automation is the only way to deal with such adaptive environments. SDN promised the capability to program the network, and there are tools to do it. However, each tool has its own APIs, their associated data models may vary and be proprietary, so integration is a costly and time-consuming process.

The TeraFlow OS SDN Controller aims at mitigating the challenges just mentioned by supporting a set of operator-driven use cases and workflows. There are two main objectives: to equip the TeraFlow OS with enough function to deal with the programmability of network elements, and to develop agents in the devices to cope with the north bound and south bound interface requirements. Figure 2 provides an example of the envisioned scenario where a set of multiple integrated network elements are used to support the autonomous provisioning and subsequent configuration and management of a transport network slice service (and any associated service elements) consisting of multiple Virtual Private Network (VPN) services such as Layer 2 (L2VPN) and/or Layer 3 (L3VPN) services. The extent of these logical transport networks can be on-net, or reach to off-net regions (and their destination end-points). In the latter case, an inter-operator inter-domain component is needed.

The optical network domain can be managed using the OpenConfig terminal device data models and/or the Open Networking Foundation (ONF) Transport API (TAPI). The microwave transport network follows the ONF Technical Reference (TR) 352 and/or Internet Engineering Task Force (IETF) Network Topology data models. Physical and virtual L3 routers can be controlled using OpenConfig data models. In this setting, the TeraFlow SDN Controller should be able to handle this multi-layer network and provide requested transport network slices.

As mentioned above, and as illustrated in Figure 2, the End-to-End (E2E) Transport Network Slice (as a Service) Abstraction is integral to the Services NBI. The details of the TeraFlow "Transport Network Slice" service and service elements are still being defined and evolved. For associated and ongoing work within Standards Defining Organizations (SDOs) we highlight the efforts within IETF and what they call the "IETF Network Slice".

> *"An IETF network slice is a logical network topology connecting a number of end-points using a set of shared or dedicated network resources that are used to satisfy specific Service Level Objectives (SLOs). An IETF network slice combines the connectivity resource requirements and associated network behaviors such as bandwidth, latency, jitter, and network functions with other resource behaviors such as compute and storage availability*." [3]

See the IETF's definition of the IETF Network Slice [3], and further elaborations in "Instantiation of IETF Network Slices in Service Providers Networks" [4]. Moreover, see Section 6 of D3.1 for more information on the IETF Network Slice. Also, see below, regarding how the Transport Network Slice service model can be mapped to underlaying service and network models, e.g., provider provisioned VPN service and network models.

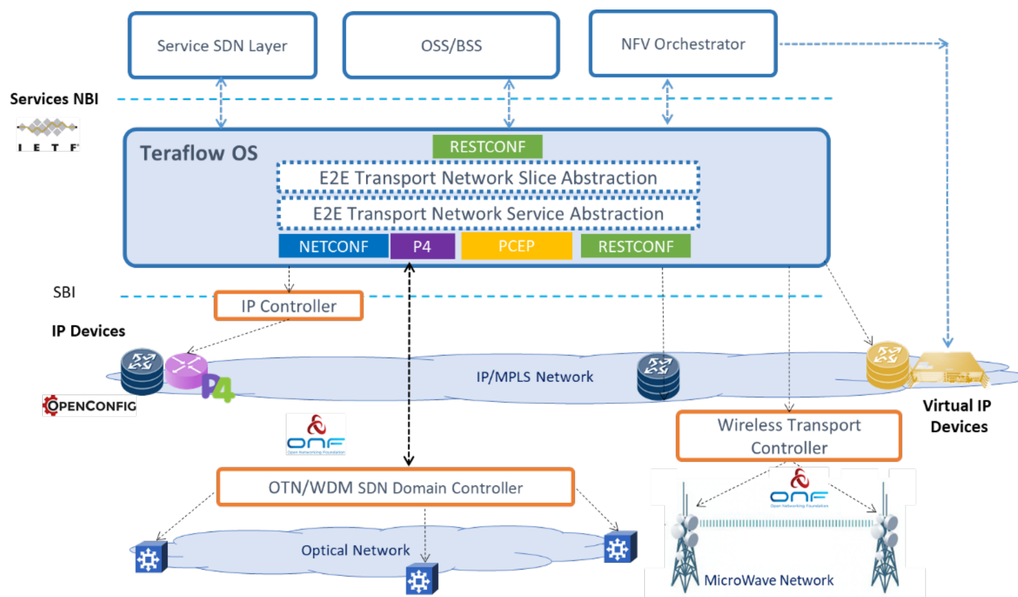



*Figure 2 Autonomous Network Beyond 5G scenario*

### 3.1.1. TeraFlow OS Prototype Features

Figure 3 shows the TeraFlow OS architecture, highlighting the most relevant components for this scenario. The Context Management component keeps track of the configurations applied in the different network resources. The Device component provides the interface with the network equipment controlled by TeraFlow OS. The Automation component performs zero-touch automated network management of workflows based on events that other components can trigger. The Service component provisions the appropriate Transport Network Services. Finally, the Slice Management component manages the life cycle of the Transport Network Slices requested through the NBI and is supported by means of L2VPN and L3VPN services. Use cases of interest for testing the validity of these components are: *Inventory*, *Topology*, *Service*, *Transport Network Slicing*, and *Automation*. More details about these use cases are provided in D2.1.

© 2021 - 2023 TeraFlow Consortium Parties 

A complete description of the components mentioned in this section is provided in D3.1. The same deliverable also provides details about the *NBI* and the *South-Bound Interface (SBI).* In short, for the NBI, we leverage Layer 2 Network YANG Model (L2NM) [5] and Layer 3 Network YANG Model (L3NM) [6] data models to provision the transport network slices supported by L2VPN and L3VPN services, respectively. For the SBI we plan to use/validate the ONF Transport API for the Open Line System (OLS), P4 to control the packet processing pipelines in the P4-enabled packet devices, OpenConfig to control the packet routers and the terminal devices, and both ONF TR-352 and IETF Network Topology data models for the microwave transport network.

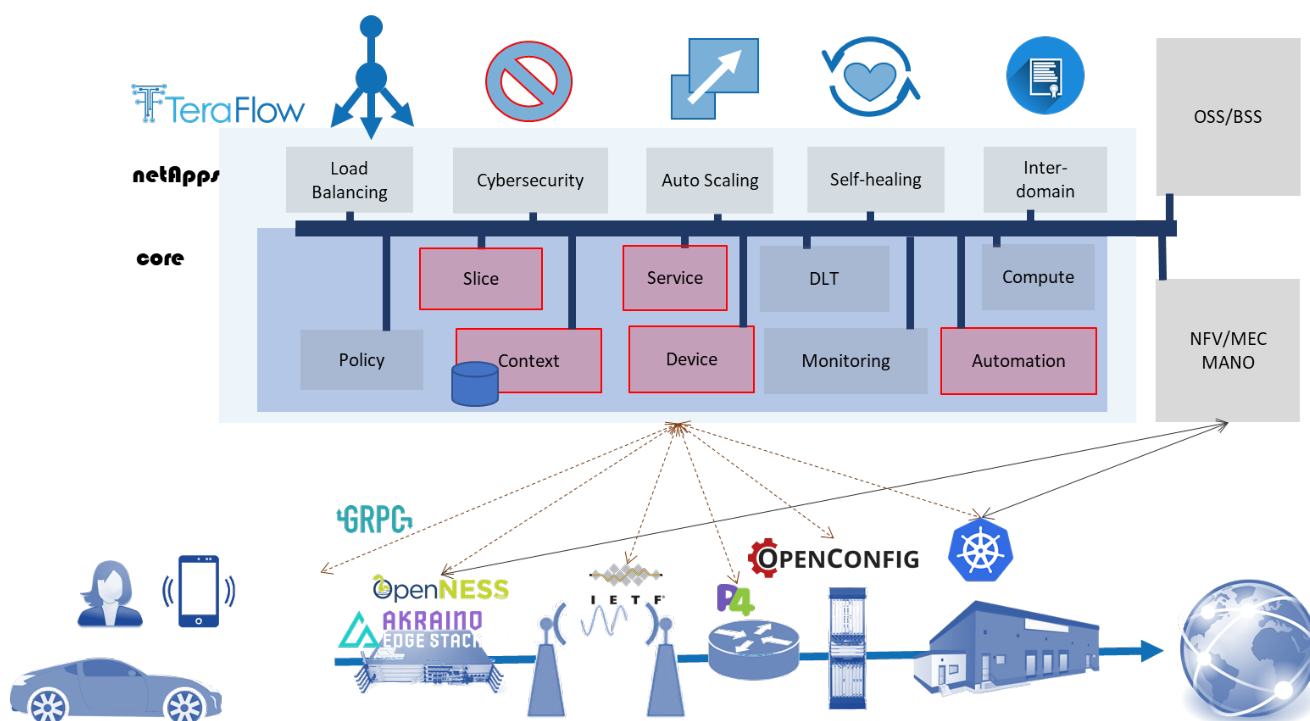



*Figure 3 TeraFlow components used in the autonomous network beyond 5G use case*

### 3.1.2. Testbed Setup

The testbed envisioned to test the use cases belonging to this scenario involves the following partners and facilities (more details on the specific facilities are available in Annex 1):

- **CTTC** contributes with the ADRENALINE testbed® (Annex 1, Section 6.1), providing an SDN/NFV packet/optical transport network and edge/core cloud infrastructure for 5G and Internet of Things (IoT) services. CTTC also provides its Kubernetes-based infrastructure to execute the TeraFlow OS instance, the TAPI-enabled Open Line System (OLS) controller, and the underlying optical transport network infrastructure.
- **Telefónica** contributes with the Future Network Lab (Annex 1, Section 6.2.2), providing access to different Internet Protocol (IP) and optical devices. The devices are controlled through the OpenConfig Driver deployed in the TeraFlow OS Device component.
- **SIAE** contributes with their SDN controller and Microwave (MW) link equipment (Annex 1, Section 6.4). The controller will oversee the MW equipment through the ONF TR-352 Driver deployed in the TeraFlow OS Device component.

© 2021 - 2023 TeraFlow Consortium Parties 

- **Infinera** contributes with its DRX-30 IP routers (Annex 1, Section 6.3) which support Netconf/OpenConfig data models. The devices will be controlled through the OpenConfig Driver deployed in the TeraFlow OS Device component.
- **Ubitech** contributes with a 32-port 400 GbE Intel Tofino-2 P4 switch (Annex 1, Section 6.9) acting as a high-performance network fabric. The device will be controlled through the P4 Driver deployed in the TeraFlow OS Device component.

The different partner premises will be connected by means of secure VPN tunnels forming a distributed testbed where the use case on Autonomous Network Beyond 5G will be assessed.

## 3.2. Scenario 2: Automotive

The automotive industry is evolving towards a vision where cars are becoming autonomous and connected wirelessly to cooperate for safer and more efficient driving. Today, most of the safety and efficiency features in vehicles are supported by on-board sensors, which are limited to visual line-of-sight. Connectivity offers a good complement to the on-board sensors by extending the vision and detection range even when visual line-of-sight is not available, while deploying Cooperative, Connected and Automated Mobility (CCAM) services.

To extend sensor capabilities in a CCAM scenario, each connected car subscribes to the topics of interest and gets the information when an update is available. Thus, it is expected that many flows requiring low-capacity and, optionally, low-latency (for mission-critical applications) will be generated. The scenarios are even more challenging in B5G networks. The main idea is to take full advantage of the sensors deployed in a vehicle to collect and analyse telematics and driver behavioural data to ensure the vehicle's performance, efficiency, and safety. This translates in 25 gigabytes of data sent to the cloud every hour for every vehicle.

The huge scale and diversity of CCAM services imposes three main requirements for transport services: low-latency, high-capacity, and massive flow management. The generation of large numbers of flows or huge aggregated volumes of data from the edge of the network to the core to make use of the cloud services will pose a challenge to the network. To overcome this scalability issue, a key solution element is to also deploy computing and storage resources at the edge of the network (i.e., MEC) and distribute the functionalities between the MEC and the core cloud, located in the core network. Moreover, it will be important to target flow management schemes that lower complexity and aim to keep real-time state information to a level that is cost effective and manageable, considering both technical and business aspects.

When looking at the deployment of CCAM services over a distributed edge and cloud infrastructure, several challenges need to be overcome. First, we need unified management of computing, storage, and networking resources. In this respect, the TeraFlow OS SDN Controller will be able to deploy integrated services (i.e., to provision cloud and edge computing resources, and connectivity between them) and optimize the cloud and network resources (i.e., packet/optical) in a concurrent way. Second, we need to address the multi-domain aspects of the problem, where resources need to be assigned in each domain and then combined in an end-to-end fashion. In this respect, the TeraFlow OS SDN Controller will deploy several per-domain instances and compose them to create end-to-end connectivity services. Finally, the different domains involved might belong to different network operators. This calls for methods to keep private the internal network details. In this respect, the TeraFlow OS SDN Controller will be equipped with a Distributed Ledger Technology (DLT) component,

based on blockchain technologies, to preserve the confidentiality of the data exchanged between the per-domain TeraFlow OS instances.

Figure 4 provides an example of the envisioned CCAM scenario. At the infrastructure layer, the scenario comprises several packet and optical transport networks for the metro and the core segments providing connectivity to the distributed cloud and edge computing infrastructure. CCAM services can be deployed in micro-DCs at the edge nodes (e.g., cell sites, street cabinets, lampposts), small-DCs (e.g., in a central office) for low/moderate-computation capacity and low response time, and core-DCs in the core network for high-computational capacity and moderate response time. Transport and cloud infrastructures are administratively partitioned into different domains, each controlled by a TeraFlow OS SDN Controller instance. In addition to selected uplink-heavy and latency-sensitive scenarios, the intention is to focus on a broadcasting application, such as Over-the-Air (OTA) updates, which are software improvements that a car company sends wirelessly to vehicles. Testing and experimentation will be necessary to address the role division and extent of the Transport Network Slice and related IETF Network Slice end-points regarding and interaction with adjacent access and service edge (SDN) control domains in the Automotive case.

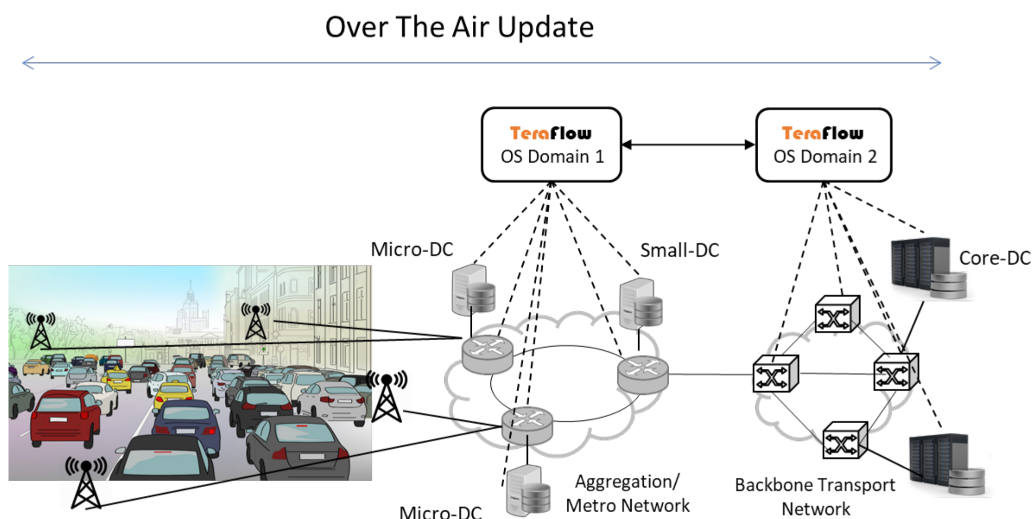



*Figure 4 Automotive scenario*

## 3.2.1. TeraFlow OS Prototype Features

The Automotive scenario complements the Autonomous Network B5G scenario by incorporating multi-domain and DLT functionalities. For this reason, the components involved in the Automotive scenario are the same as described in Section 3.1 with the addition of the DLT component (i.e., as a gateway to interconnect different TeraFlow OS instances) and the Inter-domain network application (i.e., to orchestrate the interdomain interactions between TeraFlow OS instances). Use cases and corresponding components of interest for testing the validity of the components in the Automotive scenario are: *Distributed ledger and smart contracts*, and *Inter-domain* (see Figure 5). More details about these use cases are provided in D2.1.

This scenario will also be used to validate a number of standard protocols defined in different SDOs. In particular, we are interested in testing the IETF drafts for network slices [7] and network slice [8] data model used to request Transport Network Slices, and the ETSI Permissioned Distributed Ledger (PDL) drafts used in the interconnection the DLT components.

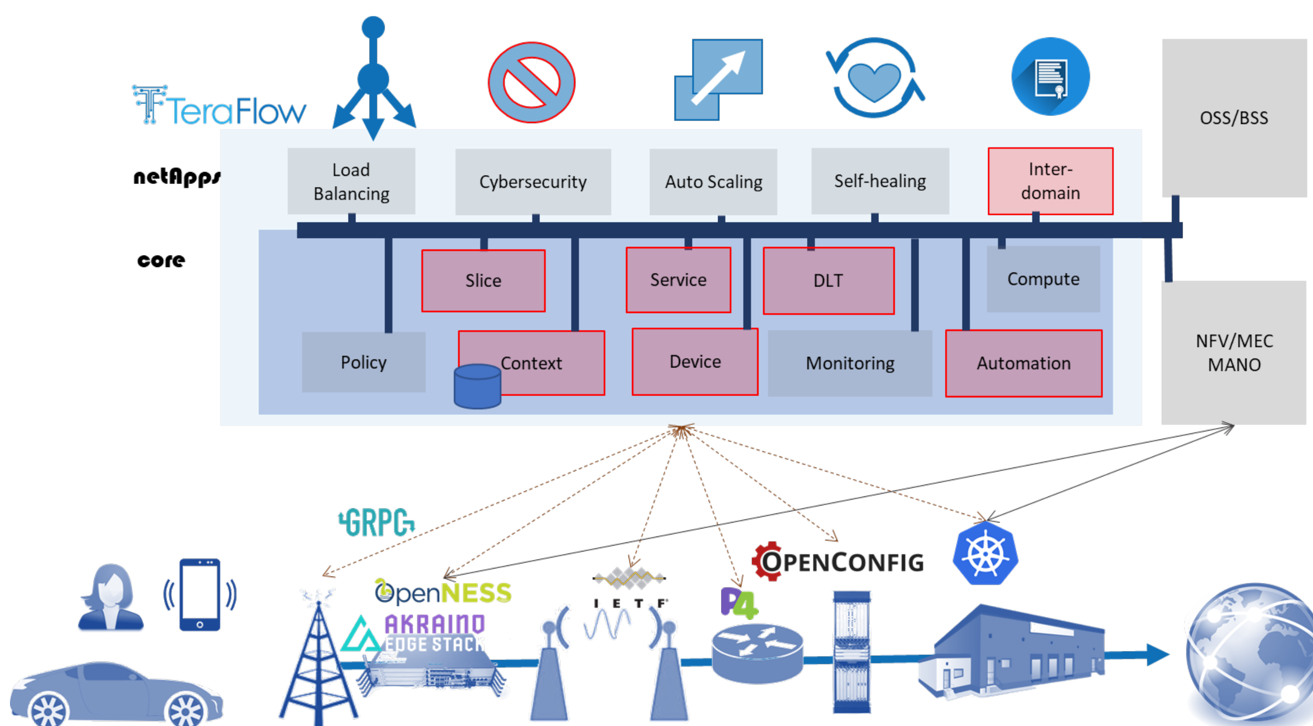



Figure 5 TeraFlow components involved in the automotive use case

### 3.2.2. Testbed Setup

The testbed envisioned to test the use cases belonging to this scenario involves the following partners and facilities (more details on the specific facilities are available in Annex 1):

- **CTTC** contributes with the ADRENALINE testbed® (Annex 1, Section 6.1) providing an SDN/NFV packet/optical transport network and edge/core cloud infrastructure for 5G and IoT services. For the validation of this scenario, we will take advantage of the TAPI-enabled OLS controller and of the underlying optical transport network infrastructure.
- **NEC** contributes with the blockchain infrastructure and runtime (Annex 1, Section 6.5) providing the means to interconnect different instances of the TeraFlow OS for the different domains.
- **Telenor** contributes with a server and a few whiteboxes (Annex 1, Section 6.7) providing the means to deploy the virtual routers that will be controlled for the Automotive use case.

The different partner premises will be connected by means of secure VPN tunnels forming a distributed testbed where the Automotive scenario will be assessed. The setup will comprise two domains controlled by two different instances of TeraFlow OS SDN Controller.

## 3.3. Scenario 3: Cybersecurity

Nowadays, when an operator moves towards an automated environment, security becomes a key feature since network operations are done by software components operating without human intervention or oversight. Moreover, the pervasive softwarisation of network and infrastructure components is further increasing their attack surface. Indeed, security must undergo a similar technological evolution to enable the resilience of SDN controllers, the automation of security policies

© 2021 - 2023 TeraFlow Consortium Parties 

over the network, the use of Machine Learning (ML) to detect and identify attacks, the utilization of DLT to assure configuration and forensic capacity, and the deployment of NFV security functions.

In this context, the very same tools can be used for attacks, such as malicious VNFs, or weaponized Artificial Intelligence (AI). Therefore, it is crucial to provide a combination of innovative solutions that are scalable in a production environment and resilient to sophisticated attacks in a common framework that integrates different security technologies to detect, identify, and mitigate both traditional and new generations of attacks across different technology domains, e.g., optical and IP layers.

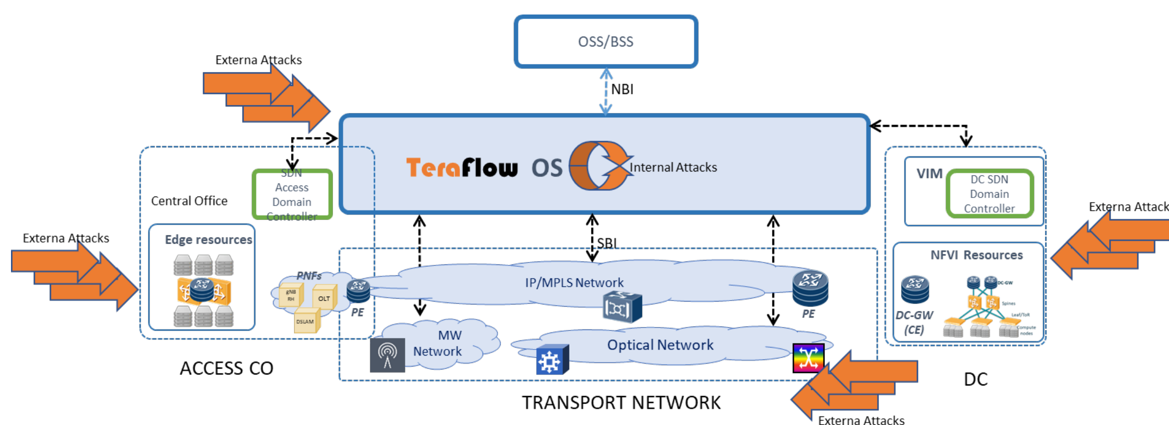



*Figure 6 Cybersecurity scenario and threats*

Figure 6 depicts an example of the envisioned Cybersecurity scenario and of the threats in the context of an automated network. At the data plane, attacks may be targeting the IP or the optical layers. Attacks exploiting the IP layer traverse or are targeted at the devices located in the access segment (e.g., edge DCs), in the core network, or in core DCs. In this case, per-packet inspection is necessary to detect and identify attacks. However, inspecting packets is a demanding operation. Executing such a process at a central packet inspector instance is impractical due to the need to transport the packets from the remote site (e.g., CO, DC) to the central location, incurring significant traffic and computing loads. Designing distributed packet inspection becomes necessary for efficient and effective attack detection at the IP layer. Moreover, it is necessary to coordinate the distributed packet inspectors, which means that a central entity is still necessary, but only for consolidating and coordinating the network's security status.

Attacks at the data plane can also exploit the optical layer. In this case, malicious access to premises hosting optical equipment may lead to disruption of the traffic of entire fibre links, to the perturbation of the quality of the transmission of certain portions of the spectrum, or even to unwarranted access to the data being transmitted. Designing accurate and fast optical attack detection, identification, and mitigation mechanisms becomes critical to avoid or minimize data losses and breaches.

At the control plane, the SDN controller and the ML models that support its operations may also be the target of malicious attacks. ML models can be induced to report false errors and make mispredictions by carefully tailoring the data fed to the model (i.e., a process known as adversarial attacks). The control plane must ensure that ML models used are not exposed or vulnerable to these kinds of attack.

The TeraFlow OS aims at mitigating these challenges by developing a cybersecurity network application which leverages the functionalities of the TeraFlow OS core components to realize a complete security assessment loop for the data plane. For the IP layer, Figure 7 illustrates the

architecture of the solution envisioned. Distributed modules perform packet inspection at the remote sites and report the security statuses to a centralized module. The centralized module receives security statuses from multiple remote sites and can consolidate these statuses for a more complete view, enabling it to detect, e.g., coordinated attacks targeting multiple remote sites. Once a complete picture of the attack is built, the attack mitigator is responsible for determining actions to mitigate the attack.

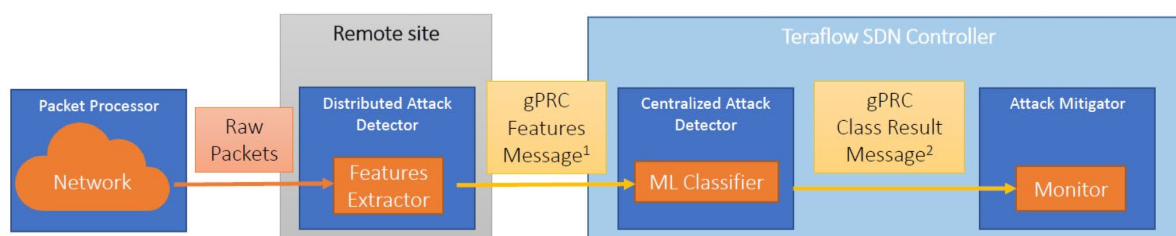



*Figure 7 Distributed Cybersecurity Component architecture*

For the optical layer, an attack detector contains the logic of the security assessment loop. It leverages the monitoring capabilities of the TeraFlow OS to periodically obtain Optical Performance Monitoring (OPM) data for each optical service. Then, an ML-based module provides attack detection inference (possibly with attack identification). The decoupling of the attack detection and inference modules brings several benefits, such as evolving the ML model without modifying the attack detector. Finally, when an attack is detected, the attack mitigator is triggered and is responsible for tracing a mitigation plan and executing it by coordinating with other components (e.g., automation). The study of mitigation strategies and how to evaluate their result is left for the next year of the project.

### 3.3.1. TeraFlow OS Prototype Features

The Cybersecurity scenario will enable the validation of several components, use cases, NBI and SBI interfaces, and protocols. The main components involved in this scenario are highlighted in Figure 8. The key component is the Cybersecurity netApp. This component contains two modules, i.e., centralized attack detector and attack mitigator, that are deployed at different containers to take advantage of the scalability and reliability features of cloud native applications. In addition, the Automation and the Service components are used for (re)configuration and provisioning tasks necessary to mitigate detected attacks. In addition to the modules deployed within the Cybersecurity netApp, an external distributed attack detector located at remote sites interacts with the TeraFlow OS SDN Controller. Note that the distributed attack detector is an external component running outside of the TeraFlow OS, it is not depicted in the figure. We refer to D4.1 for a complete description of the Cybersecurity netApp and its modules. Use cases of interest for testing the validity of these components and apps are *monitoring*, *automation*, and *service*. More details about these use cases are provided in D2.1.

© 2021 - 2023 TeraFlow Consortium Parties 

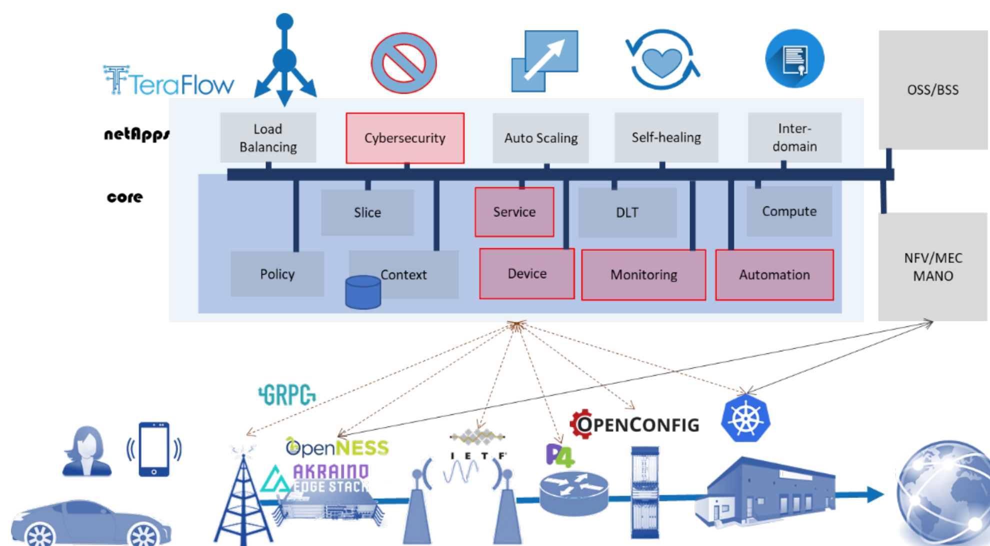



*Figure 8 TeraFlow components used in the cybersecurity use case*

## 3.3.2. Testbed Setup

The testbed envisioned to test the use cases belonging to this scenario involves the following partners and facilities (more details on the specific facilities are available in Annex 1):

- **Telefónica** will contribute with the Mouseworld and the Future Network Lab (Annex 1, Section 6.2). The first of these provides all the infrastructure to generate, capture, and process network traffic profiles and malicious traffic in order to train and evaluate ML models. The second provides access to different IP devices controlled through the OpenConfig Driver or the IETF interface deployed in the TeraFlow OS Device component for mitigation actions.
- **UPM** will contribute with the Machine Learning Lab deployed over the Mouseworld Lab, providing specific virtualized software components to deploy distributed/centralized attack detectors, including the trained models.
- **CHALMERS** will contribute with the C3SE infrastructure (Annex 1, Section 6.8), providing the infrastructure over which a Kubernetes cluster will be deployed. A custom workstation will host the Kubernetes controller and interact with workers deployed over the C3SE infrastructure. An instance of TeraFlow OS will be deployed over this cluster and used to validate the optical physical layer attack detection and mitigation features of the cybersecurity scenario. The infrastructure will also host emulated optical components that will enable the testing of the optical physical layer malicious activities.

The different partner's premises will be connected utilizing secure VPN tunnels, realizing a complete setup that can validate all the functionalities of the TeraFlow OS Cybersecurity netApp, comprising both IP and optical layers.

© 2021 - 2023 TeraFlow Consortium Parties 

# 4. Preliminary Integration Results

The previous section described the scenarios used to test, experiment with, and validate the TeraFlow OS SDN Controller. In this section, we present preliminary integration results. We first introduce the reference component architecture used to develop the TeraFlow OS Components. Then, we provide a few details of the CI/CD pipeline designed to automate the software validation and release tasks. Finally, we describe how a user can download and deploy an instance of the TeraFlow OS Controller on its own Kubernetes cluster.

## 4.1. Reference Component Architecture

The TeraFlow OS is implemented following the cloud native architecture concept. This means that the software is divided into microservices that execute specific tasks and collaborate through messages to achieve the end goal for which it is designed. A microservice can delegate operations to other microservices. They communicate using network protocols as follows:

- Services (usually one per microservice) are defined for each contextual set of operations that need to be defined, e.g., all operations related to monitoring are defined within the *monitoring* service.
- Remote Procedure Call (RPC) methods represent the operations that can be performed, e.g., *including* a device to the monitoring loop at the monitoring service.
- Messages define the information received and returned for each RPC method.

In the TeraFlow OS, the communication between components is based on gRPC Remote Procedure Calls (gRPC) [9]. We have defined different RPC methods and messages and grouped them by microservice. Deliverables D3.1 and D4.1 of this project describe the architectural details of each component, the interfaces they expose, and some preliminary experimental results validating their standalone functions.

The general architecture of the TeraFlow OS components using the default project language, i.e., Python 3, is depicted in Figure 9. The figure shows the set of folders and files available within the TeraFlow OS source code folder, referred to as ROOT in the figure. The figure also shows the structure of files and folders that should be created to implement a hypothetical component, i.e., “<component>”, that follows the reference architecture using Python 3. However, since components are independent and communicate among themselves through standard gRPC messages, several languages could be used for their implementation. Therefore, variations in programming language and project structure might be introduced when needed, depending on the expertise of the partner responsible for implementing a component. If a different programming language is used, only the .gitlab-ci.yml and Dockerfile files remain, while all other folders and files might change.

© 2021 - 2023 TeraFlow Consortium Parties

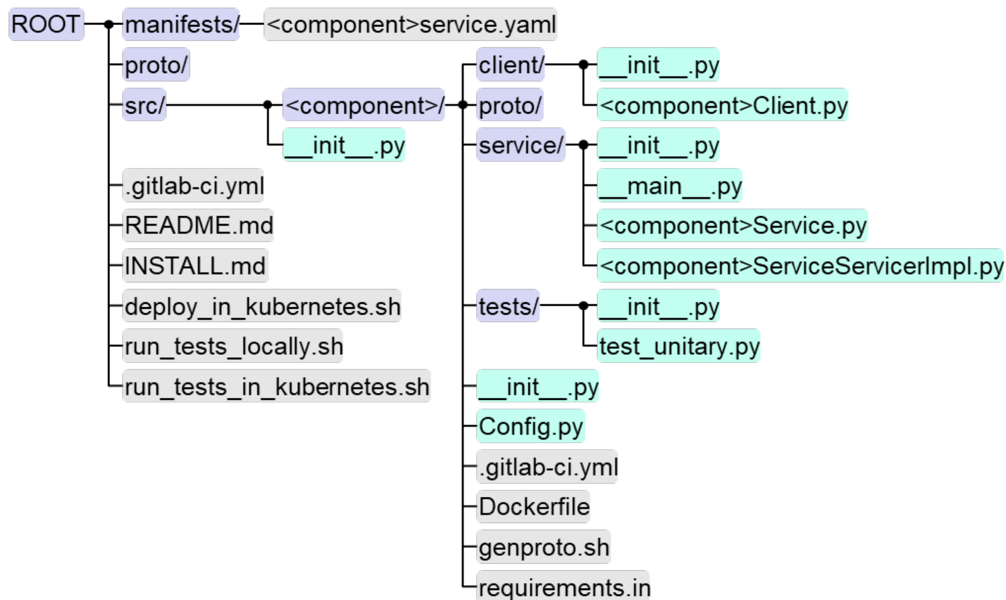



*Figure 9 TeraFlow OS general component architecture*

Within the ROOT folder of the repository, there are 3 folders and some files:

- The "manifests/" folder contains the Kubernetes manifest YAML [10] files (one per component) with the descriptors and details used to deploy the components in Kubernetes. The general recommendation is that all the components are deployed in the same namespace. However, note that these descriptors are namespace-independent, meaning that the same manifest can be used to deploy the services in different Kubernetes namespaces within the same Kubernetes cluster, but modifications to the names of components might be needed in such a deployment case.
- The "proto/" folder contains the definition of the network protocols used by the components to communicate with each other. All the files contained in this folder are defined using the gRPC language. These files are used to generate the protocol code (i.e., in a specific programming language).
- The "src/" folder contains the source code of all components (more details about this are provided later in the section).
- The ".gitlab-ci.yml" file is the root GitLab CI/CD pipeline definition file (check Section 4.2 for more details about the CI/CD pipeline). It imports the per-component ".gitlab-ci.yml" files.
- The "README.md" file provides basic reference information about the project, relevant links, and installation instruction pointers.
- The "INSTALL.md" file provides details to users on how to install the TeraFlow OS in their local Kubernetes environments.
- The "deploy_in_kubernetes.sh" script automates the deployment of the TeraFlow OS components and defines a few standard configuration variables.
- The "run_tests_locally.sh" script runs the unitary tests locally in the development environment. Developers should run this script and ensure all tests have passed before committing their contributions to the repository to prevent incompatibilities with existing components.
- The "run_tests_in_kubernetes.sh" script complements the script "run_tests_locally.sh" by running integration and functional tests in pre-production Kubernetes environments.

© 2021 - 2023 TeraFlow Consortium Parties 

For each component in the folder "src/", the reference files are organized as follows:

- The "client/" folder contains Python implementations of the clients that other components can use to communicate with this component. Note that multiple implementations might exist per component when the same client is implemented in different languages.
- The "proto/" folder contains the subset of gRPC-generated Python files used by the component. Multiple implementations might exist depending on the programming languages used.
- The "service/" folder contains the Python implementation of the component. This folder might include multiple files and folders. However, the minimal definition of a service includes:
    - The "__main__.py" file is the entry point to the component. It starts the gRPC service, and the Prometheus end-point used to collect per-component metrics. More details about Prometheus are available in Section 4.1.3.
    - The "<component>Service.py" file implements the gRPC server for the component. The service is designed to include, by default, a gRPC Health RPC that is used by Kubernetes to verify the health of the service and restart it when needed.
    - The "<component> ServiceServicerImpl.py" file implements the behaviour for each RPC method supported by the component. Each method has its performance monitored by means of Prometheus client libraries and measured values are exposed through a Prometheus HTTP metrics end-point.
- The "tests/" folder contains the unitary tests that the service should pass to confirm its correct implementation and behaviour. This folder might include multiple test files. Ideally, the tests should cover 100% of the component code, and this can be enforced by CI rules. More details about this are given in Sections 4.2.3 and 4.2.4.1. Testing is implemented based on the Python "pytest" library, and coverage is tracked using the "coverage" library.
- The "Config.py" file contains configuration constants for the component including, among others, the network ports to which the component should listen.
- The ".gitlab-ci.yml" file contains the component's GitLab CI/CD pipeline definition. This file is imported by the root ".gitlab-ci.yml" file, and allow different components to define their own CI/CD procedures. An example of the "gitlab-ci.yml" file is available as a reference in the project repository and can be used as the starting point.
- The "Dockerfile" file defines the recipe for creating the Docker image from the source code of the component.
- The "genproto.sh" script contains the commands required to gather the gRPC-compliant network protocol files from the root "proto/" folder and generate the code in the component's "proto/" folder for the required programming languages.
- The "requirements.in" file contains the set of dependencies, i.e., Python packages, that the component has.

### 4.1.1. Exposed Ports per Component

In our deployment it is essential to keep track of the exposed ports per component. As a reference for the developers, they are listed in Table 1.

| Component | Exposed ports |
|---|---|
| Context | 1010 (gRPC), 8080 (HTTP), 9192 (metrics) |
| Device | 2020 (gRPC), 9192 (metrics) |
| Service | 3030 (gRPC), 9192 (metrics) |
| Monitoring | 7070 (gRPC), 9192 (metrics) |
| Slice | 4040 (gRPC), 9192 (metrics) |
| Automation | 5050 (gRPC), 9192 (metrics) |
| Policy | 6060 (gRPC), 9192 (metrics) |
| DLT | 8080 (gRPC), 9192 (metrics) |
| Compute | 9090 (gRPC), 9192 (metrics) |
| Cybersecurity | 10000 (gRPC), 9192 (metrics) |
| Inter-domain | 10010 (gRPC), 9192 (metrics) |

*Table 1 - Ports exposed per component*

### 4.1.2. Health RPC

In order to properly monitor component status, we use the standard gRPC Health service, which allows the status of each component to be checked (i.e., UNKNOWN, SERVING, NOT_SERVING, SERVICE_UNKNOWN). Kubernetes uses this service to check the health of each pod and triggers appropriate actions when a pod is not in the expected state. Figure 10 shows the data model defined by the gRPC Health service.

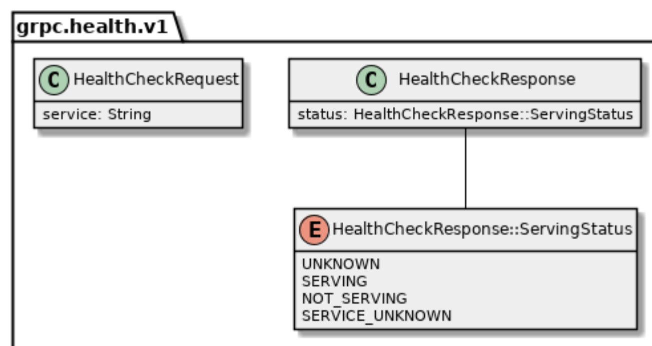



*Figure 10 gRPC Health service data model*

### 4.1.3. Metrics Exporter

Prometheus is an open-source software tool widely used for alerts, performance, and event monitoring. It is a time series database, and can record real-time metrics using an HTTP synchronous model. It provides a language for flexible queries and real-time alerting, as depicted in Figure 11. It allows monitoring from multiple data sources: *i)* Openstack, *ii)* Kubernetes (e.g., process CPU consumption), and *iii)* dedicated metrics exporters. Metrics exporters can be conveniently auto discovered inside Kubernetes, thus integrating the monitoring of all the components in a single place.

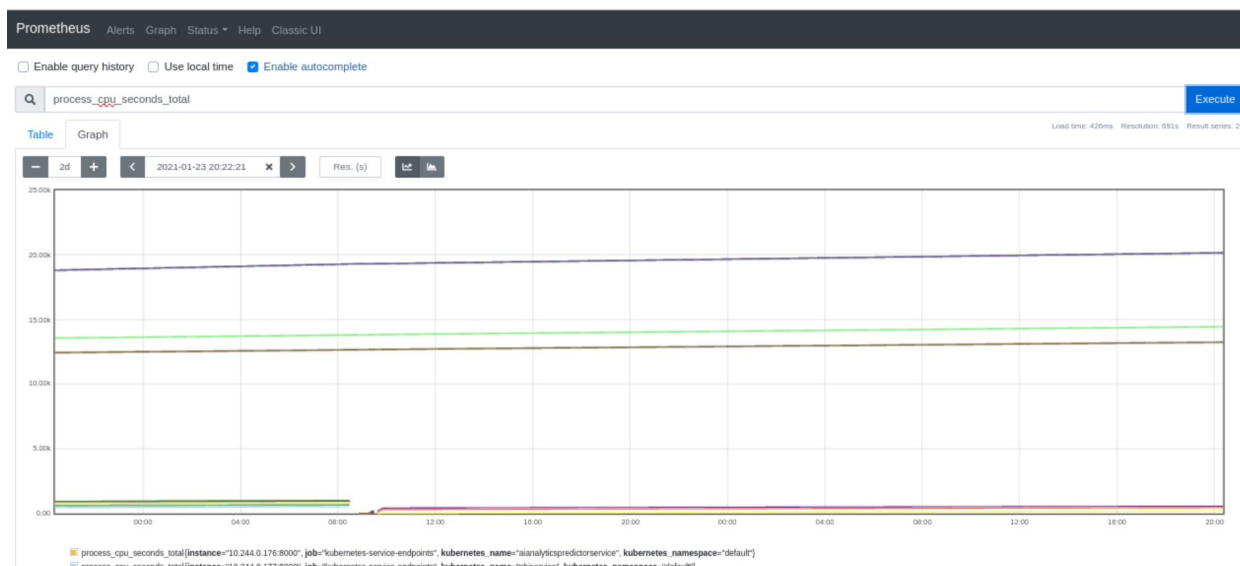



*Figure 11 Screenshot of Prometheus UI*

To have metrics exported and collected from all TeraFlow OS components, Code 1 (see in the following) provides an example of how to include and use the Prometheus client library [11].

The type of metrics that can be stored in Prometheus are:

- Counter: counters are numerical metrics that can only be incremented and are reset when the process restarts.
- Gauge: gauges are numerical metrics that can have their value set, incremented, or decremented.
- Summary: a summary is a vector that can be used to store observations, and that can be further processed to obtain averages and other statistics.
- Histogram: a histogram categorizes the observed events based on predefined ranges (referred to as buckets). This enables the calculation of probability distributions and more advanced statistics over the observed values.

```
from prometheus_client import start_http_server, Summary
import random
import time

# Create a metric to track time spent and requests made.
REQUEST_TIME = Summary('example_request_processing_seconds','Time spent processing request')

# Decorate function with metric.
@REQUEST_TIME.time()
def process_request(t):
    """A dummy function that takes some time."""
    time.sleep(t)

if __name__ == '__main__':
    # Start up the server to expose the metrics.
    start_http_server(9192)
    # Generate some requests.
    while True:
        process_request(random.random())
```

*Code 1 Example of a Prometheus client*

## 4.2. CI/CD Pipeline

Given the inherent software-based and distributed nature of the TeraFlow OS design, the TeraFlow project must provide a shared, reliable, and automated tool to accommodate the different software contributions of the consortium partners. Under this umbrella, the joint use of DevOps with the CI/CD pipeline methodology can help meet these requirements.

### 4.2.1. Introduction to CI/CD

For the sake of contextualization, some key definitions are presented below:

- *DevOps*: This term, coming from software *development* and Information Technology (IT) *operations*, is a way of performing actions and processes according to a set of ideas and practices targeted to empower the collaboration, communication, and integration among the software developers and IT/infrastructure engineers in a common IT system.
- *Continuous Integration (CI)*: This is a methodology based on the automation of tasks and processes related to software integration to minimise issues from the beginning of the integration process. It is based on the presence of a shared code repository where developers can host their code, submitting updates at short intervals to facilitate the identification of possible errors and quickly revert in the event of problems. The CI pipeline is mainly composed of two phases: build and test. Since Python, the default programming language used in TeraFlow OS, does not require building the source code, the build phase is responsible for building the containers for each component. Then, in the test phase, the unitary tests are run and their coverage is tracked. The process continues only if all the tests run successfully.
- *Continuous Delivery (CD)*: This can be seen as the natural continuation of CI but focused on the automation of the delivery of software releases. CD contributes to a basic pipeline, i.e., CI/CD, with at least three main phases: build, test, and deploy. However, the last phase, deployment to production, was historically performed manually, and is the main automation procedure that the continuous deployment process tackles.

In the rest of the section, we present the fundamental tools, the proposed methodology, and a best practice manual to properly enable the use of CI/CD methods within the TeraFlow consortium.

### 4.2.2. Tools

Within the TeraFlow project's scope and needs, we assume three main categories for the available tools needed for a proper CI/CD integration: *i)* source code management, *ii)* CI/CD frameworks, and *iii)* containerization tools.

#### 4.2.2.1. Source Code Management

Having source code management and version control is a must nowadays in a software development environment. It is especially necessary to manage code and changes made by several contributors. Plenty of tools can be found in the community, such as, CVS, Mercurial, and Subversion.

With respect to the source code versioning tool to be used during the development of the TeraFlow OS, a few aspects are important. First, the tool should allow for easy and simple distributed development due to the highly distributed development nature of the TeraFlow OS, i.e., each component has at least one partner as contributor, while several people from the partners may be

working on the project at the same time. The fact that we have decided to use a single repository for all the components makes this requirement even more important. Moreover, the tool should allow or facilitate the integration with CI/CD pipelines. Note that the code versioning tool is an important part of the CI/CD platform that, in fact, is the tool that triggers and tracks the CI/CD pipeline results. Therefore, Git has been established as the *de-facto* choice for code management and version control in the project.

### Git and Git Repositories

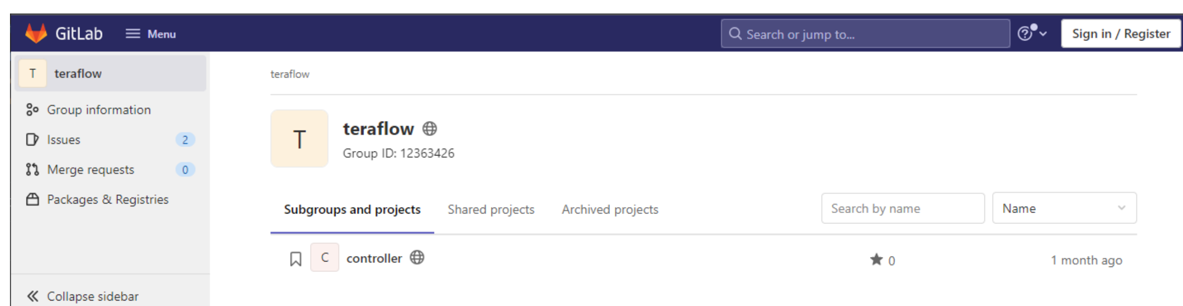
Git is the most widely used open-source software version control in the world. It manages and tracks the code modifications hosted in a common source code repository [12], and is especially useful in the development of collaborative and distributed environments. Additionally, multiple Git-based repository (or repo for short) distributions can be found, with GitHub, GitLab, and Bitbucket as probably the most representative Git management platforms/repositories.

With Git, each developer has a local (in the computer) and a remote (in the cloud) repository. This means that changes can be committed to the local repository without affecting the remote repository. The users can also create branches that are meant to concentrate changes Once the local code reaches the point at which it is ready to be run by the CI/CD pipeline, the local changes can be pushed to the remote repository. Another useful feature is the use of branches, which can be used to concentrate changes for a particular TeraFlow OS feature or component. This allows the same person to work on multiple features/components without one interfering with the other. Finally, changes made into one branch can be integrated into another branch. The branching scheme adopted for the development of TeraFlow OS is described in detail in Section 4.2.4.2. These features are relevant for the adoption of the CI/CD pipeline procedures planned for the development of the TeraFlow OS SDN Controller.

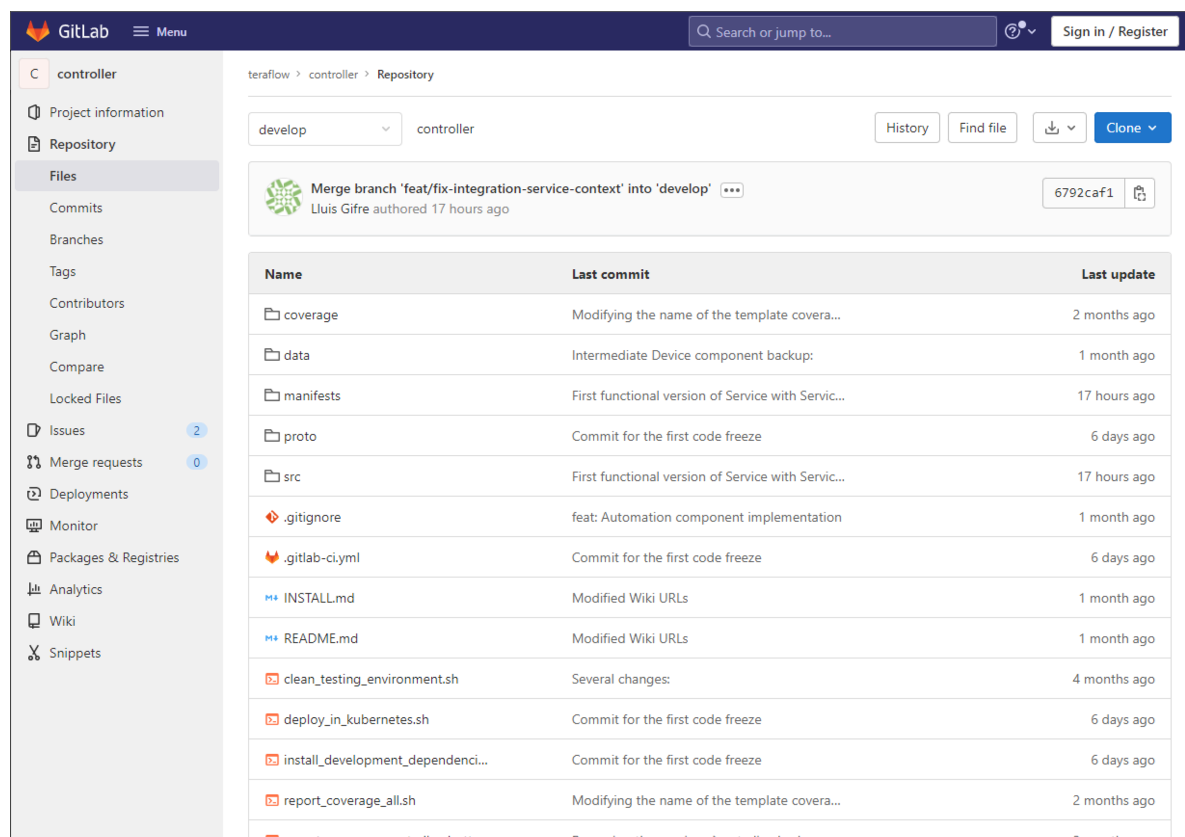
| | Free users | Max repo size | Max file size | Max API calls per hour and client |
|---|---|---|---|---|
| GitHub | No limit | 500 MB | Up to repo size | 5k |
| GitLab | No limit | 10 GB | Up to repo size | 36k |
| BitBucket | 5 | 1 GB | Up to repo size | 5k |

*Table 2 - Git-based repository tools comparison*

Table 2 presents a brief comparison of free offerings from GitHub, GitLab, and BitBucket. Having analysed the possible options for Git-based repositories and to properly meet the needs of the TeraFlow project, GitLab has been chosen as the repository to host and manage the project source code. A new environment has been created for the purposes of the TeraFlow organization under the name of *teraflow-h2020* (https://gitlab.com/teraflow-h2020), depicted in Figure 12. Within the organization, a repository named *controller* hosts the TeraFlow OS code.




*(a) Dashboard of the TeraFlow organization in GitLab*




*(b) Dashboard of the TeraFlow OS repository in GitLab*

*Figure 12 Dashboards of the TeraFlow organization and repository in GitLab*

### 4.2.2.2. CI/CD Frameworks

Projects framed around network management for 5G and beyond are pushing to align with state-of-the-art software engineering to embrace and leverage its benefits. Under this umbrella, the TeraFlow project aims to integrate CI/CD capabilities to streamline software development, integration, and deployment tasks with a higher level of resilience and cost-effectiveness.

There is a vast market for CI/CD open-source and proprietary tools to exploit the CI/CD methodology in software projects. There are multiple choices among which to select the most suitable option according to the project's needs. Some of the most popular tools are Jenkins, Buddy, TeamCity, Bamboo, and CircleCI, as well as tools provided by repository tools such as GitHub and GitLab. Under this project's scope, we will analyse two of the most widely used CI/CD frameworks: Jenkins and GitLab CI/CD. Note that the final decision of which tool to use is made after considering the containerization tools/platform in Section 4.2.2.3.

#### Jenkins

Jenkins [13] is a free, open-source tool for automating software development activities such as building, testing/deploying, and simplifying the continuous integration and continuous delivery procedures. Jenkins' core is entirely written in Java. It is released under the MIT license and is available for a huge variety of operating systems, including macOS, Windows, and the most utilized GNU/Linux distributions such as OpenSUSE, Ubuntu, and RedHat, either running in a standalone installation or in a Docker container.

The Jenkins ecosystem is very customizable, offering a broad set of plugins (>1000), including plugins for specific programming languages. Jenkins also offers parallelization tools for multi-building as well as a remote access API [14] in three different flavours: XML, Python, and JavaScript Object Notation (JSON) with JSONP support. Since Jenkins is an open-source tool that supports various plugins, it has a vast community of users with very detailed documentation.

The definition of the Jenkins pipeline [15] is contained in a text file called *Jenkinsfile*, written in a scripted pipeline syntax. The Jenkins pipelines define the entire building process, separated by stages. The stages define a subset of conceptual tasks or steps. Some common stages of a Jenkins pipeline are, for example, build, test, and deploy. Once a pipeline is defined, the *Jenkinsfile* is executed in a set of machines, commonly known as *Nodes*, integrated in the Jenkins environment.

The Jenkins team also hosts a subproject named JenkinsX [16] specialising in automating CI/CD pipelines in Kubernetes and cloud native deployments. JenkinsX is out-of-the-box integrated with Helm, Jenkins CI/CD server, Kubernetes, and other cloud-based tools providing an integrated cloud native solution to simply develop and deploy software reliably and agilely compared to traditional non-cloud-based alternatives.

For instance, Helm is an application package manager running on top of Kubernetes. It allows the application structure to be described through convenient *helm-charts* and to be managed with simple commands. Moreover, it has pre-built charts for several relevant applications.

**GitLab CI/CD**

Previously released as a standalone tool, GitLab CI/CD was integrated within the main GitLab project in 2015, forming a fully open DevOps platform packaged as a single application. As a result, GitLab acts as a single source of truth, providing source code management, version control, code documentation, role-based access control, CI/CD, code quality control and analytics, and packaging, among other features. GitLab offers automated DevOps features that eliminate the complexity of software delivery. After a commit, GitLab automatically sets up the pipeline and the integrations, automatically detects the code language, builds, tests and measures the code quality, scans potential vulnerabilities and licencing issues, offers real time monitoring, and automatically deploys the application. GitLab is entirely written in Go and Ruby and shared under MIT license.

GitLab CI/CD pipelines are defined in the *gitlab-ci.yml* [17] and comprise stages and jobs. Jobs define what to do, and stages specify when to do a specific job. Jobs can output files and directories: these outputs are called artifacts. As an example, a typical stage might be "*build*" while the job associated to this stage might be "*compile the code*", and the artifact might be "*the executable file*". The GitLab pipeline is based on runners, build instances that are used to execute the jobs over multiple machines and share the results to the Gitlab core. Runners can be shared or specific. Shared runners allow a single runner to handle jobs with similar requirements from multiple projects, while specific runners are created per project under specific requirements.

In addition, GitLab includes a Container Registry, offering a secure and private registry for Docker images completely integrated with GitLab. This feature allows developers to easily deploy Docker container images using GitLab CI/CD and other Docker-compliant tools.

GitLab can be used as an SaaS platform running in their servers or can be executed locally deployed as a Self-Managed platform. GitLab offers different pricing options [18] including a free plan that offers unlimited private repositories and contributors, up to 10GB of storage per repository, and 400 minutes of CI/CD pipeline execution per month (applicable only to shared runners). Gitlab provides excellent

documentation and has a great community offering priority support with the premium and ultimate subscription tiers.

### Comparison

Jenkins and GitLab CI/CD are two of the most widely adopted CI/CD solutions on the market. The main difference between them is that GitLab acts as a single source of truth since in addition to CI/CD, it offers source code management, version control, and code documentation (among other features) in a single application. GitLab also offers better integration with cloud native and container-based developments and supports SaaS deployments. A more detailed, side-by-side comparison of both frameworks is found in Table 3.

| | Jenkins | GitLab CI/CD |
|---|---|---|
| Licensing | Open Source | Open Source |
| Pricing | Free | Free<br>Premium ($19 per user/month)<br>Ultimate ($99 per user/month) |
| Deployment | On premise | SaaS/On premise |
| Official support | No | Yes |
| Official documentation | Yes in https://www.jenkins.io/doc/ | Yes in https://docs.gitlab.com |
| Source code management | Not included. Git is supported through third party plugins | Yes, it is included in GitLab and also provides integration with external SCMs such as GitHub |
| Application Performance Monitoring | Does not include the feature to analyse performance | Yes, shows performance metrics for the deployed apps |
| Application performance alert | Not included. You can get email notifications through third-party plugins | Yes, it allows creation of service-level alerts for any event occurring within the code |
| Code quality check | Not included. It can provide code quality check through third-party plugins | Yes, it includes code quality checks |
| Built-in container registry | Not included. There are third-party plugins for uploading/downloading images from repositories. Jenkins-X includes a container registry | Yes, it offers a secure and private registry for Docker images |
| Preview changes | Not included | Yes, it includes live preview of changes and can be used in each different branch |
| API | Yes | Yes |
| Auto DevOps | No | Yes |

*Table 3 - CI/CD frameworks side-by-side comparison*

#### 4.2.2.3. Containerization (Kubernetes/Docker)

GitLab offers out-of-the-box integration with Kubernetes, supporting integration with multiple Kubernetes clusters for different environments such as production, development, staging, etc. GitLab allows direct management of Kubernetes clusters, providing namespaces and automatically creating

© 2021 - 2023 TeraFlow Consortium Parties 

the required resources for the GitLab projects. Moreover, GitLab includes a built-in container registry that allows the use of GitLab as a container repository for Kubernetes.

Historically, the interaction between Kubernetes clusters and GitLab has been done through *kubectl* or *helm* tools directly in the jobs [19], but in the latest versions of GitLab a new integration method has been introduced [20]. This new method relies on the GitLab Kubernetes Agent ("*The Agent*"), simplifying the integration, increasing the security, and providing a safe, reliable, and future-proof integration solution between GitLab and Kubernetes clusters. The Agent provides a continuous and permanent connection to the Kubernetes cluster through websockets or gRPC, maintaining a minimal footprint on the cluster side, while allowing multiple Agents to run in the same cluster with different access levels. The Agent is still under development and will only be available for a selected group of customers and projects. The Agent will be publicly available in later releases and further developed with more features.

According to the analysis reported previously, given the wide catalogue of the functionality provided, the fully functional integration with Kubernetes, the easiness in defining CI/CD stages and jobs in a single configuration file, the ability to exploit the benefits of having the source-of-truth in a single framework, and taking into account the choice of GitLab to host the TeraFlow source code, GitLab CI/CD has been chosen to realise the CI/CD jobs within the TeraFlow project.

### 4.2.3. TeraFlow Methodology for CI/CD

In this subsection, we present the proposed methodology to develop, integrate, test, and release the artefacts produced by TeraFlow under a common framework by the following CI/CD principles. The methodology is presented in four complementary tiers: *i)* functional architecture and pipeline; *ii)* TeraFlow CI/CD infrastructure; *iii)* good practice and hints for CI/CD usage; and *iv)* release time plan.

#### 4.2.3.1. Functional Architecture and CI/CD Pipeline

As explained in the previous subsection, GitLab has been chosen as the CI/CD framework. It is composed of two main actors: *i)* the GitLab server, where its repository and tools can be deployed both in the gitlab.com facilities and in a local server; and *ii)* the GitLab Runner, that oversees running and managing all the pipelines events, build, test, package, and deploy, when triggered from the Gitlab server. The events are configured in the *gitlab-ci.yml* file. This approach of GitLab, based on having a single source-of-truth, makes it very easy to connect to the production infrastructure, in this case the Kubernetes cluster, where the stable versions of TeraFlow OS will be stored.

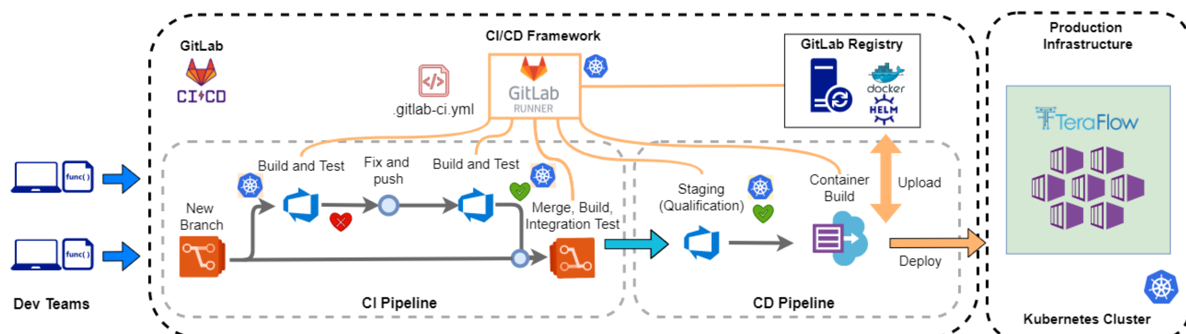



*Figure 13 Functional Architecture in TeraFlow to enable CI/CD pipeline*

Figure 13 shows an overview of the global pipeline over the CI/CD functional architecture. The pipeline is assumed to start when TeraFlow developers submit (i.e., push) their source code to the GitLab

repository. The CI pipeline is then launched, followed by the CD pipeline, until a stable version is reached in the production infrastructure.

### Continuous Integration Pipeline

The CI Pipeline is triggered when a development team starts to work on a new feature of a specific component. The pipeline should follow a sequence of events similar to the ones shown in Figure 13 for the CI Pipeline, described as follows:

1. A new branch is created for the new feature from the main branch of its component according to the branching recommendations (more details in subsection 4.2.4), e.g., *feat/mycomp-#x-new-functionality*.
2. When the new functionality is mature enough, a new commit is pushed to the remote branch and the GitLab server informs the Runner to trigger the build and test stages according to its configuration in the *.gitlab-ci.yml* file. Then, the Runner clones the code locally and commands the jobs. In parallel, artifacts are stored in the GitLab registry according to their tags in the config file. If the tests pass correctly, the Runner informs the GitLab server to proceed with a merge request action. Otherwise, the issues must be solved.
3. A new version of the source code is updated to fix the issues (new commit and/or push) and step 2 is executed again to build and test the new version of the code. If the tests point out no further issues, the CI pipeline can proceed.
4. Once all the features belonging to the same development branch have been committed, the source code is integrated, and any overlapping conflicts are fixed during the merge process. At that point, all the features are merged under the same development branch for that component.
5. Finally, the Runner commands the same stages (build, test), but adding some integration tests to oversee the interconnection of this component with the rest of them in the TeraFlow ecosystem. These tests are performed at the integration environment.

The CI lifecycle can be run several times to ensure the reliability of the new features before merging them into a common branch and triggering the CD pipeline.

### Continuous Delivery Pipeline

The CD pipeline is carried out when a development or a master branch merge request is executed successfully.

1. The GitLab Runner retrieves the latest artifacts from the GitLab registry corresponding to the given pipeline.
2. Before deploying a new release into the production environment, the GitLab Runner executes some actions, to be performed in the staging environment (qualification), related to the identification of potential migration issues given in the development branch. This phase is very close to the production stage.
3. Finally, when all the previous tests have passed, a new release is ready to be deployed in the production environment. The actions to be executed by the Runner are quite similar to the ones in the previous stage, but are applied to the master branch. In this final stage, the artifacts (docker containers and helm charts) are reuploaded to the GitLab container registry with a release tag associated to this version.

#### 4.2.3.2. TeraFlow CI/CD Infrastructure

Figure 14 presents the infrastructure currently in use to support the TeraFlow CI/CD architecture. It comprises three main actors/sites:

- **Development Environment**: This is where the code is programmed and locally tested by the developers. It works in a distributed fashion within the facilities of the TeraFlow partners, i.e., each developer has their own environment.
- **GitLab Server**: This can be hosted in the GitLab.com cloud or can be deployed in an internal local server intended only for TeraFlow project purposes. TeraFlow currently uses the cloud deployment provided by GitLab.com. The GitLab server contains: *i)* a clone of the TeraFlow OS source code; *ii)* the CI/CD configuration file (*.gitlab-ci.yml*); and *iii)* the GitLab registry where the artifacts of the CI/CD pipeline (replicas of docker containers and helm charts) are stored.
- **CTTC Facilities**: Two main environments are created to enable some of the key stages in the CI/CD pipeline. The environments are deployed in different servers with a Kubernetes cluster and a GitLab Runner to handle the CI/CD jobs in each one:
    - Integration and testing environment: TeraFlow OS components are initially tested for compatibility and interconnectivity/integration purposes.
    - Production environment: This hosts the components in a final release stage. This environment will be used as a testbed for TeraFlow OS pilots during years 2 and 3 of the project.

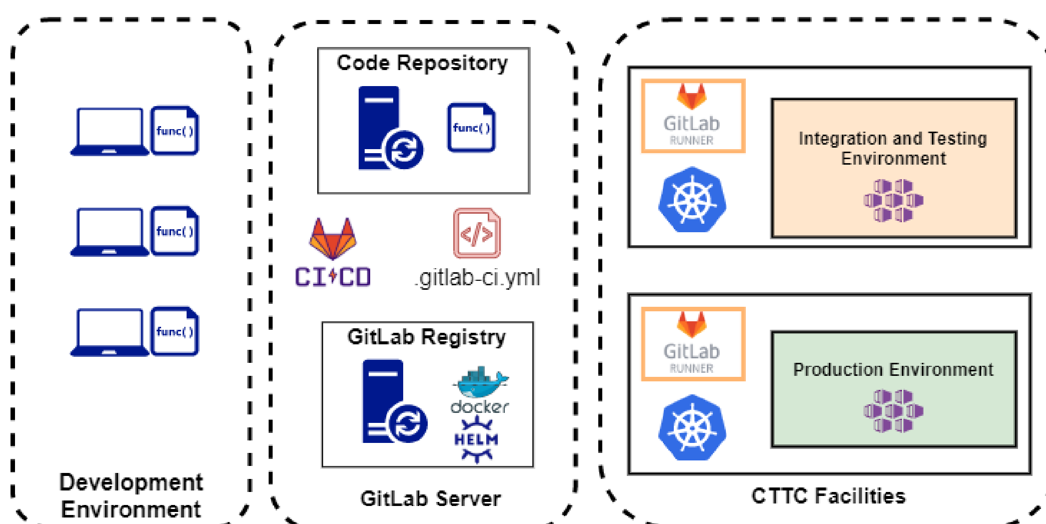



*Figure 14 Overall overview of the TeraFlow CI/CD Infrastructure*

### 4.2.4. Good Practices and Hints

In the TeraFlow project, it is essential to provide a common environment that allows component developers to work within a unified project vision. This section specifies a set of good practices and hints not only for development purposes, but also to properly apply the principles of the CI/CD methodology proposed in this document. Such recommendations are grouped into three main blocks: testing, GitFlow practices, and repository structure and usage.

#### 4.2.4.1. Tests Types

Proper test planning is essential in the development of software projects, although it is particularly valuable not to concentrate on a single type of test, but to spread the efforts in such a way that a good coverage is achieved and is not susceptible to failures.

Tests can be classified into three groups:

- **Unit Tests**: Responsible for testing isolated, small portions of code. Such tests should not be very large in size and end-to-end logic is not normally specified at this stage. An effective software architecture should support unit tests with almost no breaks. Their objective is to assert that, given some static state that the software is in, the expected output is obtained. Typically, they are present from the earliest stages of a CI/CD pipeline.
- **Integration Tests**: These are targeted at stressing the system to find any issue when a software unit (e.g., a microservice in the case of the TeraFlow OS) is communicating with other software units under the same scope. In other words, these tests check whether components can work together. It is recommended to use integration tests after each merge request.
- **Function Tests**: These tests try to find any problems in the fulfilment of an end-to-end function, e.g., a user story that describes how a service request is processed by the TeraFlow OS. They should be executed at the CI/CD pipeline stages that can provide an overall perspective of such function.

Although there are different practices and other types of test (such as acceptance testing, security testing, and static code analysis), they are out of scope for the basic set of recommendations to be considered in the core CI/CD pipeline in the creation of TeraFlow OS components.

#### 4.2.4.2. GitFlow Reference for Branch Naming and Merge Requests

GitFlow is a reference tool in the software development workflow that offers a set of good practices when using Git, showing how to structure the resources in source code version control [21]. According to such recommendations, we follow the structure represented in Figure 15.

*Figure 15 GitFlow graph representing an exemplary structure involving the five types of branches*

There are two main branches (*master* and *develop*) and three auxiliary ones (*hotfix*, *release*, and *feature*). These branches work as follows:

- **Master**: Contains every stable release of the project. Any commits (code pieces) uploaded to this branch must be ready to be included in production.
- **Hotfix**: Emerges from the master branch. It contains a production version with a bug that needs to be fixed urgently. Once the bug is fixed, the content of this branch is included in the master and develop branches. In addition, the fixed production version must be marked with a tag in the master branch (v1.1 in Figure 15).

- **Release**: This also comes from the master branch containing the code to be included in the next release. It is a step in the release of a new version of the code. All major and minor issues should have been solved at this point. Once ended, this branch must be connected to both the master and develop branches. According to the convention, the branch should be named as *release-**, where * represents the version of the release and corresponds with the tag in the master branch.
- **Develop**: Holds the development code for the next planned version of the project. It will include each of the new features to be developed. Note that the master and develop branches are only aligned in the first commit of the workflow.
- **Feature**: Emerges from the develop branch hosting the commits for a new feature. Once the new feature is completely implemented and tested, this feature branch is merged back into the develop branch. We recommend naming each new feature branch as follows: *feat/mycomp-#id-new-functionality*, where my *mycomp* represents the name of the TeraFlow OS component associated, *#id* is a unique feature number, and the name ends with the short description of the new functionality.

#### 4.2.4.3. GitLab Repository Structure

Figure 16 shows the proposed structure of the TeraFlow repository in GitLab to host the necessary source code, scripts, and configuration files required to perform the project's software-based work. This proposed structure is in line with the one described in Section 4.1. The *controller* root folder of the repository should contain at least the following folders and files:

- *proto*: Hosts the data model descriptions of the TeraFlow OS components, structured as Protocol Buffers, and the RPCs used to export their functionalities and features via the gRPC protocol. Each component will be defined in one single *.proto* file.
- *src*: This folder is used to host the source code of the microservices: in our case, the components. Each component will have its own folder with all the necessary information to perform its role. It is worth mentioning that the root "*uService_i"* shown in Figure 16 will be replaced by the corresponding name of the components. Such folders will have, at least, the following content:
    - *common* folder. Place to host common resources needed for the development of the components.
        - *logger.py*. Python class to log events in microservices.
    - *uService_i* folder.
        - Files specific for each microservice. Section 4.1 describes the reference architecture for each microservice, and the content of this folder.
- *.gitlab-ci.yml*: This configuration file contains all the information to define the global CI/CD pipeline and to command its corresponding stages and jobs.

Additionally, in each folder of the GitLab repository, a README.md file is provided to contain all the necessary documentation about content of the folder, such as a functional description of the source code, how to set the config files, or how to execute the scripts.

It should be noted that this suggested TeraFlow repository structure is an initial proposal that has already been used by the partners to produce the code freeze reported in MS3.2 and MS4.1. Nevertheless, throughout the project we will continue improving the processes, and the structure may be subject to various updates and modifications to meet the requirements during development and testing stages of the TeraFlow OS.

© 2021 - 2023 TeraFlow Consortium Parties

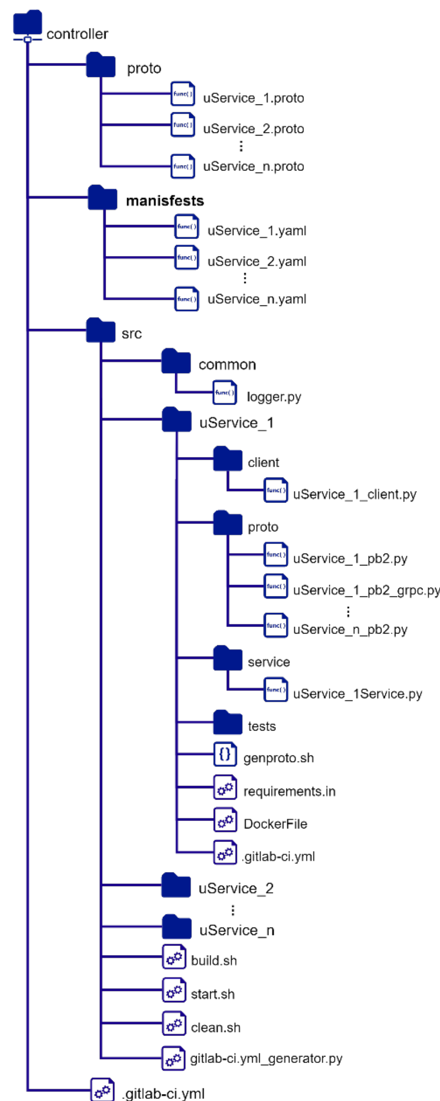



*Figure 16 TeraFlow GitLab repository structure*

#### 4.2.4.4. Release Time Plan

According to the methodology plan exposed in the Description of the Action, the TeraFlow project will produce three main software releases:

- v1 will be released at M12
- v2 will be released at M24
- v2.1 will be released at M30

The major software releases will deliver the main functions, while the minor release will provide bug fixes and possibly additional features required by the operating experiments.

## 4.3. Installation Procedures

Installation instructions to deploy the TeraFlow OS are found in the TeraFlow OS GitLab repository [2]. The project *README.md* and *INSTALL.md* files describe the procedures and provide appropriate pointers to the scripts and files to be used. Note that these resources will be continuously updated as the implementation of the TeraFlow OS progresses. For the latest information, please visit the

TeraFlow OS repository. These files assume that the user has the appropriate setup, i.e., computer with appropriate OS and Kubernetes installation.

The *INSTALL.md* file describes the installation procedure that mainly consists of cloning the GitLab repository, modifying the *deploy_in_kubernetes.sh* script present in the root folder of the repository, and running that script. The available configuration variables supported by this script are listed Table 4.

| Variable | Default Value | Description |
|---|---|---|
| REGISTRY_IMAGE | "" <empty string> | Defines the URL of the Docker registry where the images built for each TeraFlow OS component will be pushed. Empty string means that the internal Docker image base should be used instead of a registry. |
| COMPONENTS | "context device service compute monitoring centralizedattackdetector" | Contains the space-separated sequence of TeraFlow OS components the user wants to build and deploy in Kubernetes. |
| IMAGE_TAG | "tf-dev" | Defines the Docker image tag that will be used to tag the built images. |
| K8S_NAMESPACE | "tf-dev" | Specifies the Kubernetes namespace that will be created (or re-created) to deploy the TeraFlow OS. |

*Table 4 - Configuration variables supported by deploy_in_kubernetes.sh*

Note that this script is a basic skeleton provided to automate the deployment procedure in a development setup. It is expected that users will adapt it to accommodate the particularities of their environments. Details such as Docker registry login, proxy configurations, and Kubernetes roles and accesses, among others, are intentionally left open.

To facilitate the process of preparing a computer to receive a TeraFlow OS instance, be it for development or testing purposes, we have made a wiki page that hosts instructions [22]. Currently, the wiki has three articles, but these will be extended. One of the guides specifies the process to install Kubernetes on a computer, and another describes where the user can download a VM that already has all the fundamental Kubernetes components installed. To guide project partners and potential users on how to prepare a computer to host a development version of the TeraFlow OS, the Wiki page entitled “Installing Kubernetes on your Linux machine” describes how to deploy Kubernetes on a Linux machine in a way convenient to run the TeraFlow OS. If the user wants to use a pre-built virtual machine, the Wiki page “Using a pre-built virtual machine” describes how to download the VM and execute TeraFlow OS. Finally, a Wiki page listing a few useful Git and GitLab procedures is available for the project partners.

© 2021 - 2023 TeraFlow Consortium Parties 

# 5. Conclusions and Next Steps

This deliverable describes the ongoing activities of WP5 and provides a summary of the progress made by the partners over the first year of the project. The primary outcomes are: *i)* a detailed list and description of scenarios for the experimental activities to be carried out over the next years; *ii)* the documentation of the reference component architecture adopted for the development of the TeraFlow OS components; *iii)* the definition of the code development and integration pipeline made available and to be followed by the partners; and *iv)* a description of the infrastructure available at the partners' premises, and their relationship with the scenarios.

For the next steps, the preparation and improvement of testbed facilities will continue throughout the project. Now that an inventory of the equipment at each partner has been made, and the contribution of each partner to each scenario has been defined, a more detailed design will be prepared. The integration of the TeraFlow OS will also continue with the adoption and of the CI/CD platform, and any necessary enhancements to the platform will be made. Work will continue on preparing the next releases of the TeraFlow OS. Finally, starting from year 2 of the project, the scenario integration and demonstration activities will start, where the prototype of the TeraFlow OS will be deployed at the partners' premises to realize the testbed setup reported in this deliverable.

© 2021 - 2023 TeraFlow Consortium Parties 

# 6. Annex 1: Inventory of Facilities

This annex describes the testbed facilities available at the partners' premises. They can be used to develop and test some of the TeraFlow OS components during the project. The realization and performance assessment of each scenario described in Section 2 will rely on a subset of the facilities described in the following sections.

## 6.1. CTTC

The ADRENALINE testbed® [23] is an SDN/NFV packet/optical transport network and edge/core cloud infrastructure for 5G and IoT services. A representation of the testbed is provided in Figure 17.

Key elements include:

- ***SDN-enabled disaggregated Optical Transport Network (OTN).*** 1 photonic mesh network (PMN) with 4 nodes (2 ROADMs and 2 OXCs) and 5 bidirectional flexi/fixed-grid DWDM amplified optical links up to 150 km. 1 passive optical network (PON) with a 19-core 25Km multi core fibre (MCF) and a bundle of single-mode optical fibres (SMFs). 1 programmable SDN-enabled S-BVT transmitting multiple flows at variable data rates/reach. 1 OLS controller for PMN and PON. 1 Transport SDN controller (ONOS/ABNO) orchestrating the OLS controller and the SDN-enabled BVT.
- ***SDN-enabled Packet Transport Network (PTN)***. 4 PTN covering edge, metro, and intra-DC segments. 13 OpenFlow switches deployed on COTS platforms and using OvS. 5 OpenFlow switches that are equipped with tuneable 10 Gb/s XFPs as alien wavelengths. 4 SDN controllers based on OpenDaylight, one for each PTN, provide connectivity with the CTTC mobile and vehicular testbed and the 5GBarcelona infrastructure.
- ***Edge and cloud datacentres (DCs).*** 4 edge-DCs, 1 core-DC, and 2 vehicular MECs are deployed. 2 Edge-DCs deploy a container-based server (Kubernetes), and the other 2 deploy an OpenStack compute node. Vehicular MECs are integrated in OpenFlow switches deploying Kubernetes. The core-DC is composed of 3 HPC servers controlled using a single OpenStack controller.
- ***Transport SDN controller (T-SDN).*** A proprietary implementation based on the IETF ABNO architecture [1] for end-to-end connectivity services among virtual machines, containers, and end-points. The T-SDN controller acts as global orchestrator of multiple SDN controllers with a parent/child hierarchy. It orchestrates the 4 SDN controllers of the PTNs and the SDN controller of the OTN. TAPI is used as the northbound interface.
- ***NFV service platform (SP).*** An NFV service platform based on SONATA open source software. It deploys NFV network services and network slices for multi-tenancy. (i.e., verticals). The NFV SP orchestrates the 3 OpenStack controllers and 4 Kubernetes controllers. NFV network services composed of chained virtualized network functions (VNFs) deployed on VM and/or containers. A network slice is composed of one or more concatenated NFV network services that deploy a set of VNFs and/or cloud native network functions (CNFs).

© 2021 - 2023 TeraFlow Consortium Parties 

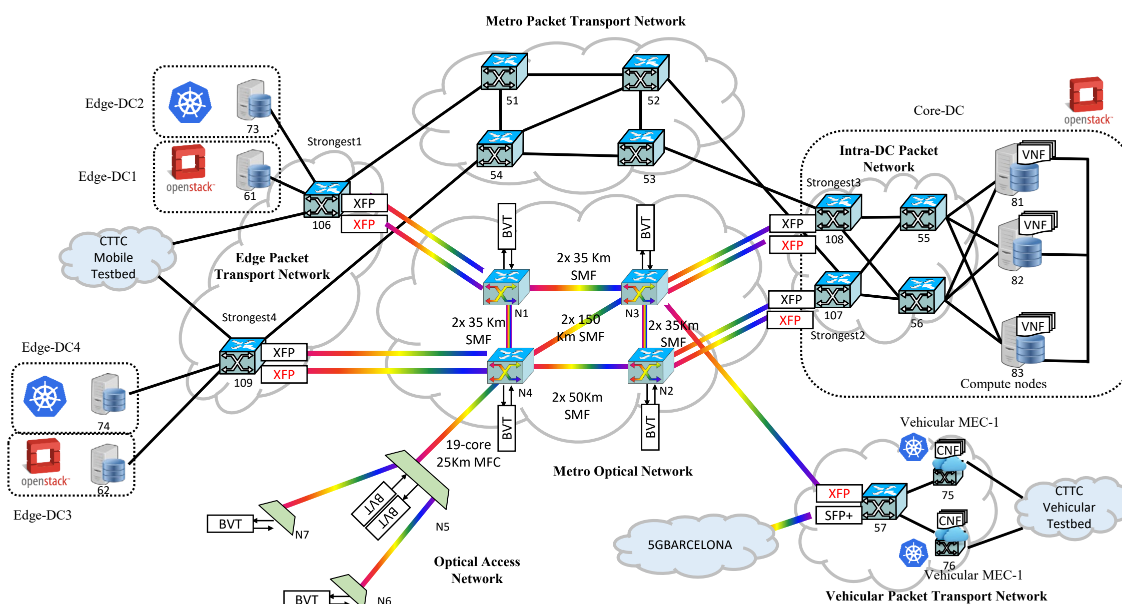



*Figure 17 Adrenaline testbed*

## 6.2. Telefónica

Two facilities will be used in the project: the Mouseworld Laboratory, and the Future Network Laboratory. Both are described in the following subsections.

### 6.2.1. The Mouseworld Laboratory

The Mouseworld Lab (Figure 18) follows the approach of a Digital Network Twin (DNT) focused on network traffic for different types of applications, including security. Mouseworld is the solution in charge of emulating a specific network configuration and generating the required traffic in a controlled environment to be used subsequently by advanced data processing components such as machine learning algorithms. The emulation environment setup in Telefónica premises network scenarios to be deployed in a controlled way using NFV-enabled architecture, under the management of an NFV Orchestrator (NFVO) and extending an ETSI NFV Management and Orchestration (MANO) stack as necessary. To this end, several resources are available:

- A set of computer servers for virtualization, based on Intel processors.
- Networks switches compatible with OpenFlow, and routers for external connectivity.
- Isolated Internet connectivity for security experiments.
- Commercial traffic generators (Keysight Breaking Point for security, and Spirent for network traffic load).
- Interconnection with a 5G lab (5TONIC).
- VPN and remote access.

The use of NFV technology allows deployment of different scenarios for different experiments and traffic independently, and provides a way to launch clients and servers and to collect the traffic generated by them even if they interact with clients and servers outside the Mouseworld, i.e., on the Internet.

© 2021 - 2023 TeraFlow Consortium Parties 

For each experiment, the Mouseworld Lab uses four modules interacting in a pipeline: Launcher, Data Collector, Feature Extraction, and Tagger.

- *The Launcher* coordinates each Mouseworld scenario and runs experiments that generate real network traffic that crosses not only the Mouseworld internal network but also the Internet. This component facilitates the generation of three basic types of regular traffic: web, video, and file hosting (e.g., Dropbox, OwnCloud) by running real clients installed in a batch of Linux virtual machines. In addition, the Launcher can also manage ad-hoc virtual machines and run sessions to generate network traffic of a collection of complementary Internet protocols using Ixia BreakingPoint, a commercial tool that allows to generate complex patterns of synthetic traffic.
- *The Data Collector and storage* module gathers all the packets generated by a single experiment.
- *The Feature Extraction* module groups the collected packets into flows based on the classic five-tuple of source and destination IP-address/port number and transport protocol, and calls an external module to obtain the set of features of each flow. This module uses a modified version of the Tstat tool in order to derive, from flow statistics, a set of standard Netflow features together with a set of highly informative features. The modified Tstat not only obtains flow features in forensic mode at the end of the connection as is done in the original Tstat, but also in real-time at different moments. This allows us to train and test machine learning components that can identify attacks or normal flows even when only a few packets of the flows have been transmitted.
- *The Tagger* automatically and without human intervention adds labels to each flow using external and log information output from the Launcher during the execution of each experiment. As flow tagging is highly dependent on the machine learning task, this component is simply a wrapper of the ad-hoc tagger that must be developed for each type of scenario. It is worth noting that labels are only generated for training and testing machine learning models, and labels are not needed to generate predictions when they are running in a production environment.

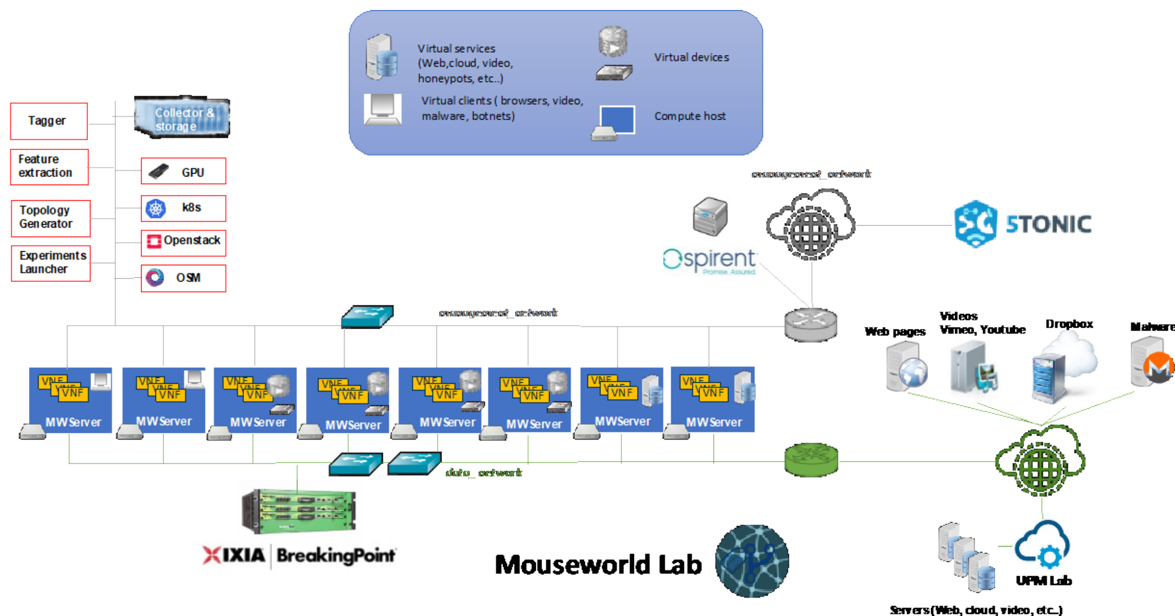



Figure 18 The Mouseworld facility

### 6.2.2. The Future Network Laboratory

The Future Network Laboratory, located in Distrito C, Sur 3, Madrid, Spain, is central for validation and certification of the NBI and SBI APIs required by the whole Telefónica group. The validation of APIs is divided into NBI and SBI specification validation. IP/Optical NBI specifications are related to the interfaces from the SDN controller to the OSS layer. The validation tools for the NBI are based on the RESTCONF protocol and YANG data models. IP/Optical SBI specifications are related to the interfaces from the SDN controller to the network layer. The SBI validation tools automate the devices' configuration tasks using NETCONF and OpenConfig.

The lab facility combines local and remote equipment, and it is composed of a combination of different devices, controllers, and software tools:

- The IP/Optical **testing software tools** used are the following:
  - *Postman* is a collaboration platform for API development and validation. A set of collections are defined to test the CRUD (Create, Read, Update, Delete) operations on the Network Controllers.
  - *Newman* is a command-line collection runner for Postman. It allows you to effortlessly run and test a Postman collection directly from the command line. It is built with extensibility in mind to easily integrate with your continuous integration servers and build systems.
  - *Swagger* is a collaboration platform used to create Open-API Specifications. It is used by the team to document the desired NBI behaviour on a web-based environment.
  - Python has a set of libraries to automate the NETCONF integration/testing process (PyTest & NCC-Client).
- **SDN IP controllers** with the *MUST* APIs support:
  - Cisco NSO.
  - Juniper/Anuta ATOM.
  - Nokia NSO.
  - Infinera Transcend Simphony.
- **SDN optical controllers** and OLS (optical line systems) devices:
  - Ciena: MCP and a hierarchical controller (MDSO) (both deployed in Germany) and 3 6500 optical nodes.
  - Huawei: 3 9800 optical nodes with a ring topology, and 3 OTN electrical nodes (new HW tested remotely in Brazil).
  - Infinera: 3 7300 optical nodes and an SDN controller.
  - ADVA: 3 FSP 3000 optical nodes and an SDN controller.
  - Nokia and ZTE are also tested remotely from the lab.
- **IP devices:**
  - Nokia 7750.
  - Cisco virtual-XR.
  - Juniper virtual MX and MX240.
  - Huawei NE-40 and ATN950.
  - Infinera DXR-30.
  - ADVA and IP-Infusion whiteboxes.
- **Open Terminals Optical devices** (transponders and muxponders):
  - Ciena: 2 waveserver nodes with 400G line interfaces.
  - Huawei: 2 DC908 nodes with 200G line interfaces (deployed in Germany).

- Nokia: 2 PSS-8 with 200G interfaces.
- ZTE: 2 ZXONE7000 with 200G line interfaces.
- Cisco: 2 NCS1004 with 400G line interfaces.
- ADVA: 2 types of Open terminals (Cloud connect and Flex).
- Cassini. 2 whiteboxes temporally loaned to CTTC.

## 6.3. Infinera

Infinera is currently testing OpenConfig implementation in the Espoo/Finland lab, using the two-node setup presented in Figure 19 where a DRX-30 and a Edgecore AS-7316 are connected back-to-back.

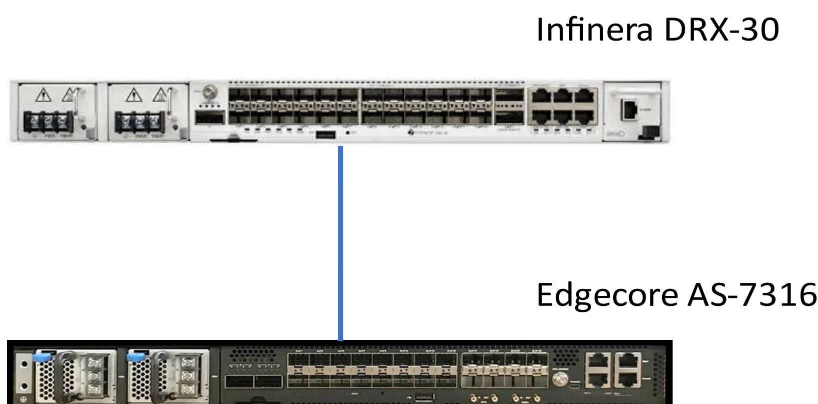



*Figure 19 Testing setup at the Espoo/Finland Lab*

While this setup is not sufficient for a comprehensive L3VPN functionality testing, the underlying idea is that the L3VPN functionality itself is being tested and regression tested in several larger setups, and in this context the purpose of testing is to ensure that the pre-existing function can be configured and monitored using OpenConfig. That is, the testing scope is newly introduced as a configuration/monitoring interface. So far, the assessment has been that, for OpenConfig testing, it is sufficient that BGP sessions are UP between the nodes and LSPs are functional, hence, the above setup has been sufficient up to this point.

Infinera plans to revisit the testing setup from time to time and introduce new routers into the setup, or introduce a completely new setup for testing, if that is needed during the course of the TeraFlow project, or for OpenConfig development and testing activities overall.

There are two testing streams that are ongoing: *i)* Testing the OpenConfig interface directly with XML syntax, and *ii)* Testing with Infinera SDN (where Infinera SDN is not part of TeraFlow project).

## 6.4. SIAE Microelettronica

The SIAE testbed will include at least two MW links managed by an SDN Controller implementing an SBI ONF MW model via NETCONF.

Expanding the scenarios for an autonomous network Beyond 5G, SIAE foresees that the TeraFlow OS will include specific modules to manage the different technologies of a transport network.

With specific focus on MW Networks, an SDN controller module is foreseen to be able to be orchestrated by a Hierarchical Controller together with Optical and IP SDN Controller modules to

provide a homogeneous vision of the underlying transport network, as illustrated in Figure 20. This architecture will provide an abstracted representation of network resources enabling integrated management based on the service requirements.

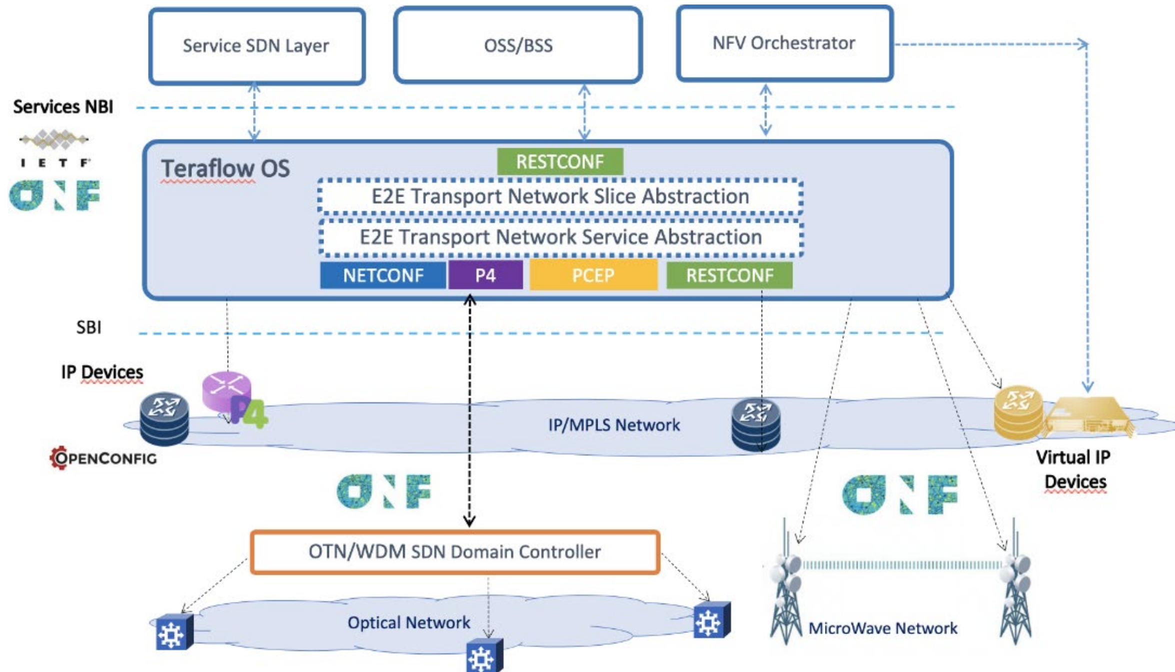



*Figure 20 Architecture including an SDN controlled Microwave network*

As regards the underlying transport network, a possible multi-technology scenario capable of supporting L3VPN services is presented in Figure 21, where the SIAE contribution is related to the MW radio links.

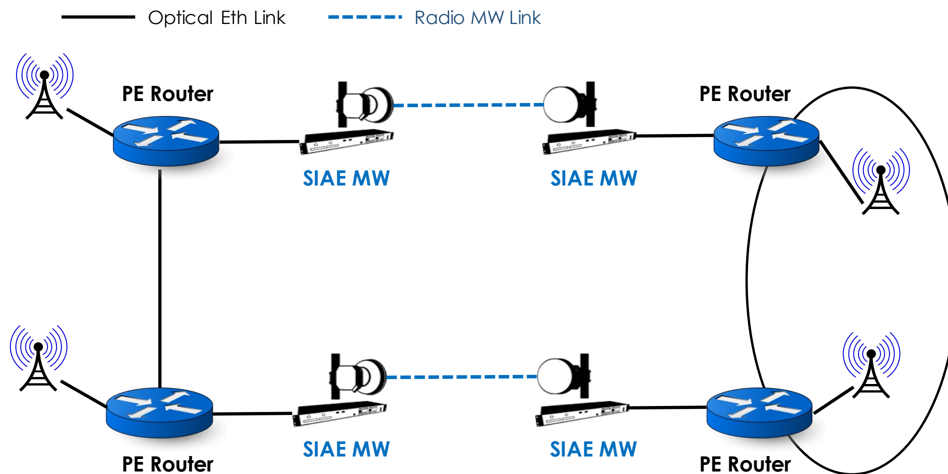



*Figure 21 An example of L3VPN with MW links*

As regards the TeraFlow OS, two future scenarios are currently under internal evaluation as regards the SDN MW Controller module:

- Using an open source SDN Controller (e.g., OpenDayLight) that is appropriately enriched with developments to meet the needs that will emerge from the definition of the use case.

© 2021 - 2023 TeraFlow Consortium Parties 

- Using the SIAE MW SDN Controller (not part of the TeraFlow project) to manage SIAE MW Equipment via NETCONF and interacting at the NBI using an IETF RESTCONF API with an HL Transport Controller capable of orchestrating the activity of all SDN controller modules for the various transport technologies.

## 6.5. NEC Laboratories

NEC will provide the DLT component and the necessary SDK/APIs to communicate with the blockchain. The network entities, their services, and the components of the TeraFlow OS will interact with the ledger through dedicated smart contracts.

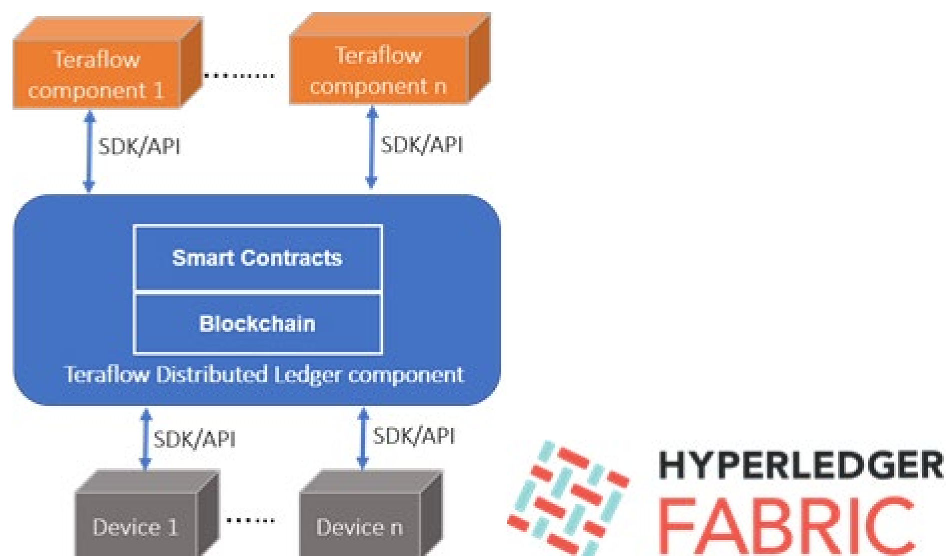



*Figure 22 Distributed ledger and its components*

For the implementation and evaluation of the DLT and its components, NEC will develop a customized architecture based on the Hyperledger Fabric test network (Figure 22).

A distributed application in the Hyperledger Fabric consists of two components:

- A smart contract, called *chaincode*, implementing the application logic.
- An *endorsement policy*, enabling the validation of transactions.

The entities involved in a Hyperledger Fabric blockchain consist of a set of network nodes with different roles:

- *Clients*, which submit to-be-executed transactions, support the execution phase, and transmit validated transactions to the ordering service.
- *Peers*, which are responsible for the execution and validation of transactions. Each peer maintains a blockchain ledger, which records transactions in the form of a hash chain, and a state, providing a succinct representation of the latest ledger state.
- *Ordering service nodes* (also known as *orderers*), which establish a total order among all endorsed transactions, thereby producing an ordered sequence of transactions arranged in blocks, according to a pluggable consensus protocol.

The Hyperledger Fabric architecture allows flexible trust assumptions to be expressed: all clients are untrusted (i.e., they are considered as potentially malicious), while peers are grouped into

© 2021 - 2023 TeraFlow Consortium Parties 

*organizations* such that mutual trust is assumed within each organization. This model is suitable for accommodating diverse, application-specific requirements.

## 6.6. ATOS

The ATOS Telecom testbed (Figure 23) is physically located in Madrid and has secure external access managed by an OpenVPN server located in a Leaseweb-based (public cloud) environment. Additionally, this environment is also configured to provide a communication channel with the ETSI Hub for Integration and Validation (HIVE) for NFV testing purposes in a collaborative multi-party scenario.

The physical ATOS testbed is supplemented by two OpenStack Points of Presence (PoPs): a Kubernetes Cluster, and a server that hosts the local repositories of the ATOS Research and Innovation (ARI) department.

PoP1 consists of two physical machines, one with the OpenStack controller and a compute node, and the other with an additional compute node. PoP2 is an all-in-one OpenStack instantiation with the controller and a single compute node on the same machine. Furthermore, the ATOS Kubernetes Cluster includes four Intel NUC miniPCs to host the pods.

ATOS Telecom's testbed connectivity is enabled by a single router intended to be used as a gateway for two L3 spine switches to handle the testbed's internal network.

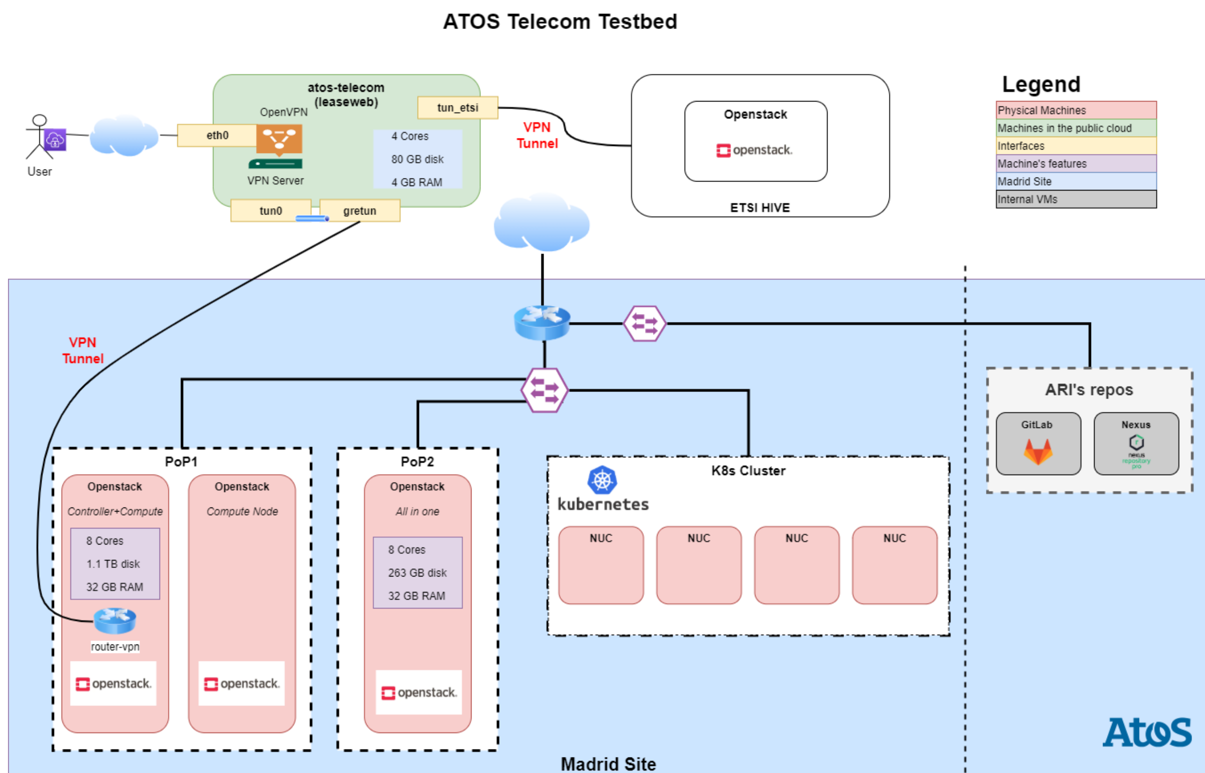



*Figure 23 The ATOS telecom testbed*

The ATOS testbed will be used only for internal development and testing purposes.

## 6.7. Telenor

Telenor's testbed will include one server and at least two whitebox switches, as shown in Figure 24. If budget allows, the testbed may be expanded into a 3-switch setup.

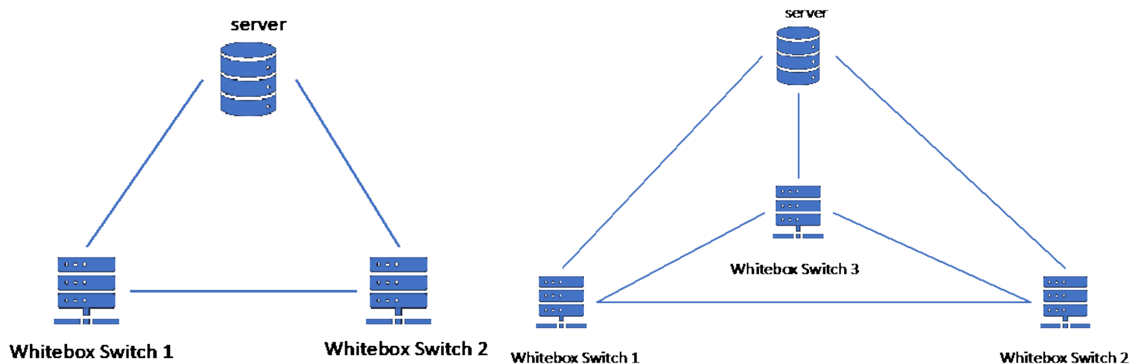



*Figure 24 Telenor's testbed setting*

The physical server will deploy the TeraFlow OS and router virtualization platform software (Figure 25)[2]. The router virtualization platform is responsible for virtualizing each whitebox switch into multiple virtual routers (2 virtual routers per switch in this case) and providing a control plane for each virtual router (see the green lines between VRx CP and V-Router x). An initial plan is to allocate 2 VMs for the Kubernetes cluster hosting a TeraFlow OS instance and 5 VMs for the router virtualization platform. If High Availability (HA) is required, additional VMs would be needed for the TeraFlow OS in the future.

The data plane traffic traverses the virtual routers (see the red lines between V-Routers).

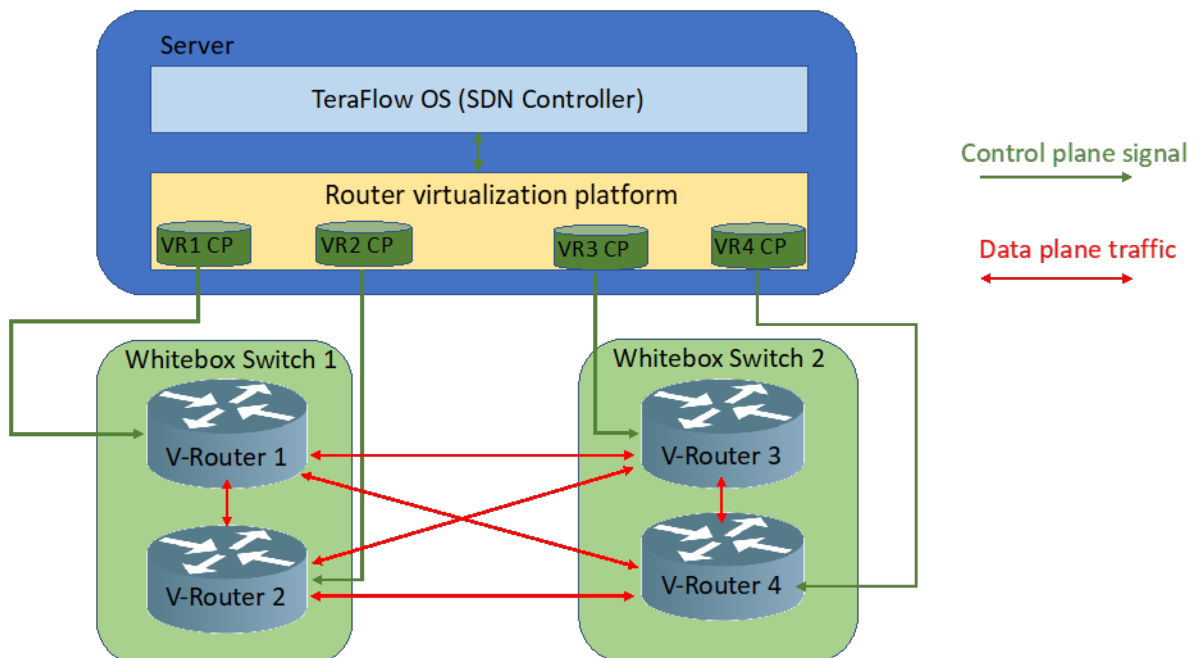



*Figure 25 Setup with the Volta Platform software*

[2] Initially, the VOLTA platform software was planned to be used. With VOLTA leaving the consortium, another partner will provide this virtualization platform software.

© 2021 - 2023 TeraFlow Consortium Parties 

Based on this setup, two virtual networks (VNs) can be created: VN1 with V-Routers 1 and 4 (purple), and VN2 with V-Routers 2 and 3 (orange) (Figure 26). These two VNs can be used to create two transport network slices (TN slices). Note that the inter-slice links between relevant V-Routers are drawn as dotted lines, to distinguish them from the intra-slice links (solid lines).

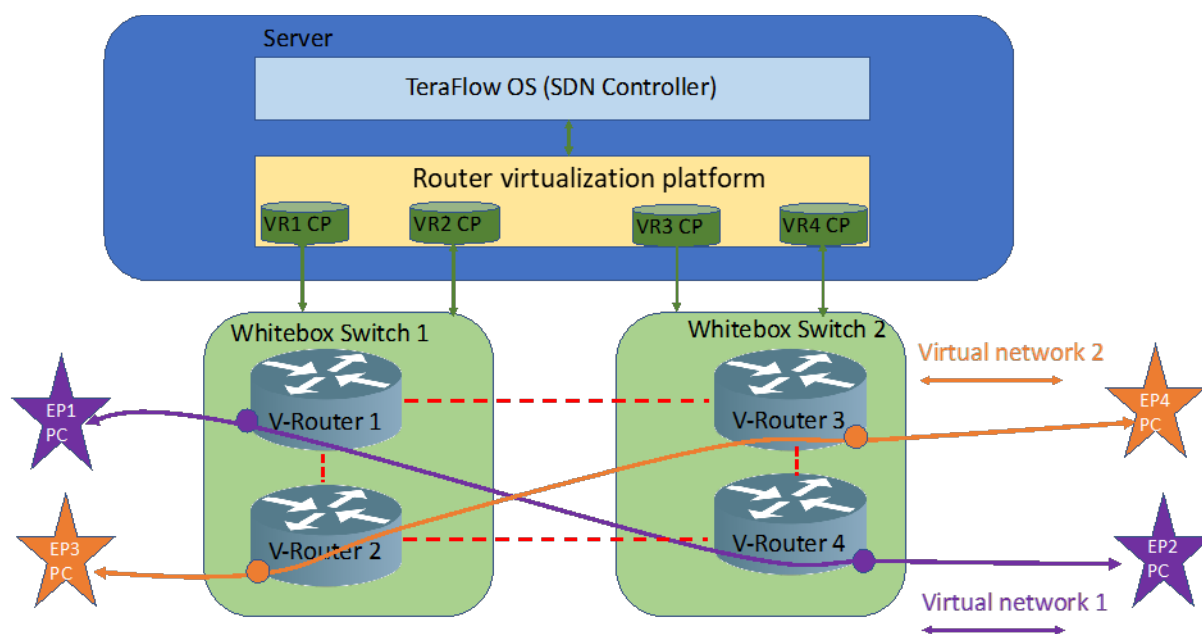



*Figure 26 Transport network slices example*

TNOR is currently evaluating this plan and will start the procurement process once the plan is approved and confirmed to be aligned with the allocated resource (budget).

## 6.8. Chalmers University of Technology

The Department of Electrical Engineering (E2) in Chalmers is a member of C3SE (Chalmers Centre for Computational Science and Engineering - https://www.c3se.chalmers.se/), Chalmers' infrastructure for demanding computing calculations and data storage. The facility is one of the nodes of the Swedish supercomputer Swedish National Infrastructure for Computing, SNIC. The system's maximum performance is expected to be around 300 teraflops, i.e., 300 trillion floating-point operations per second. This capacity is expected to increase throughout 2021 with the addition of a new node (Alvis).

C3SE currently has 3 clusters available to its members: Hebbe, Vera, and Alvis. Hebbe has 315 compute nodes, with a total of 6300 CPU cores, 26 TB of RAM, and 6 high-performance GPUs. Vera has 245 compute nodes, with a total of 7848 CPU cores, 28 TB of RAM, and 13 high-performance GPUs. Alvis is a cluster dedicated to Artificial Intelligence and Machine Learning research, with more than 200 high-performance GPUs.

Figure 27 shows the overall architecture adopted in each of the clusters. All the nodes in C3SE are interconnected by 56 Gbps interfaces, enabling low latency/high-speed parallel executions. These high-speed interfaces can be especially useful to run the TeraFlow micro-services. Each cluster runs its own scheduling engine. Jobs submitted through the login node are schedule at one or more nodes. All the nodes are based on Linux and are pre-installed with several commonly used software platforms. The system also has a container runtime engine compatible with Docker images, which allows users to run their own containers within the cluster. The C3SE infrastructure can be used in conjunction with other partners' infrastructure by establishing secure network tunnels (VPN).

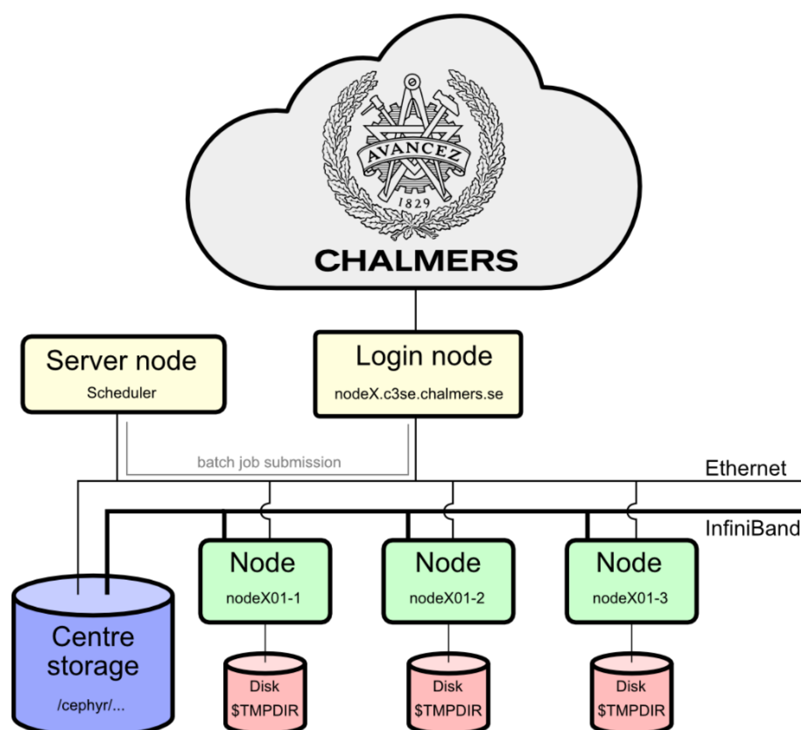



*Figure 27 Overall architecture of the clusters*

## 6.9. UBITECH

Through the research and technical activities within the TeraFlow project, UBITECH aims to form a state-of-the-art experimentation platform that will lead the ongoing 5G activities of the company, while forming a solid background for the upcoming 6G research and development initiatives. This experimentation platform is depicted in Figure 28.

The main objective of the platform is to carry out the P4 activities within the TeraFlow OS. These are among the key contributions of UBITECH in the TeraFlow project. In the rest of this section, we outline the main hardware and software components of this testbed.

Hardware:

- 2 servers based on the latest AMD Milan, each with a 64-core processor (128 hyper-threaded cores) clocked at 2.45 GHz, 256 MB of CPU cache, 256 GB of DDR4 main memory, and 2 TBs of SSD storage.
- Each server is equipped with:
    - A dual-port 100 GbE NVIDIA ConnectX-5 NIC on a PCIe gen3.0x16 slot
    - A Xilinx Alveo SN-1000 smart NIC on a PCIe gen3.0x16 slot, which comprises of the following edge networking and processing components:
        - A P4-compliant Xilinx XCU26 FPGA with 2x 4GB DDR4 memory and a dual-port 100 GbE network interface card; and
        - An embedded ARM Cortex-72 processor with 16 physical cores and 4GB of DDR4 memory.
- A 32-port 400 GbE Intel Tofino-2 P4 switch acting as a high-performance network fabric between the servers.

- An Amarisoft Callbox Classic 4G/5G system interconnected with the switch. This system allows injection of cellular traffic towards the servers and through the switch, mainly for experimentation purposes.

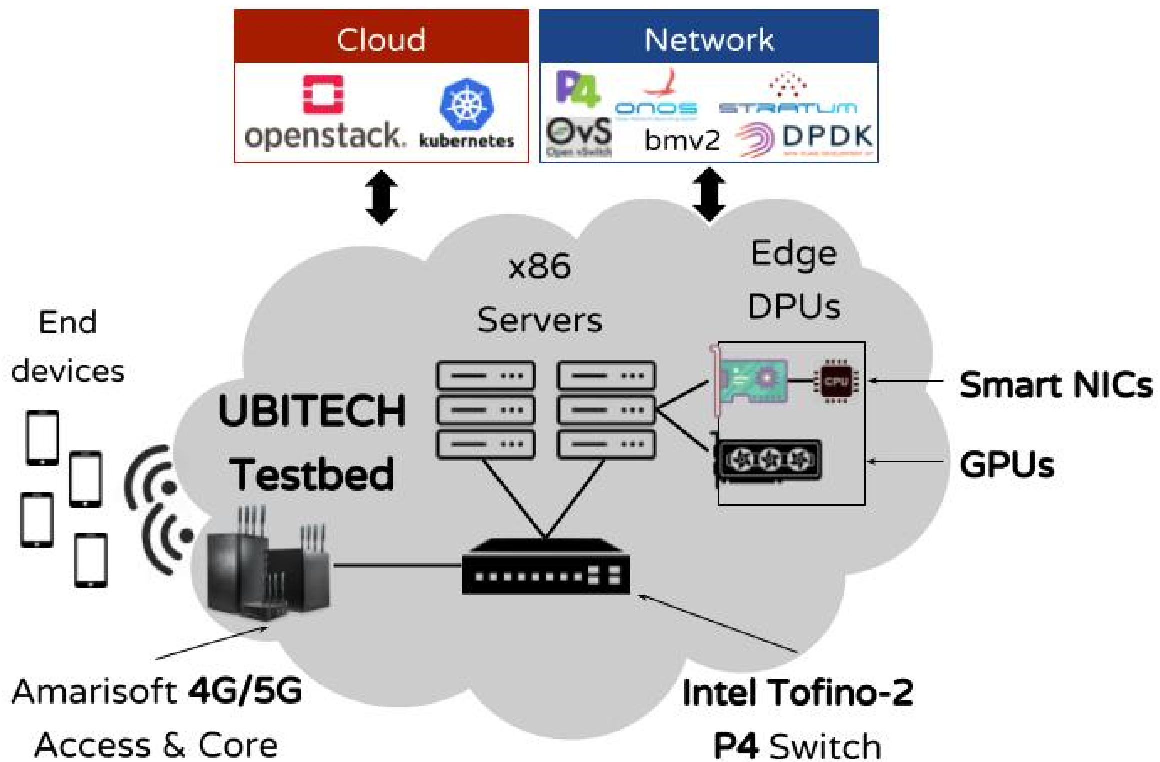



*Figure 28 UBITECH's testbed*

Software

- Compute, memory, and storage resources are provisioned through a cloud platform, such as OpenStack, CloudStack, or Kubernetes.
- A variety of state-of-the-art networking technologies are available for TeraFlow. Such technologies include - but are not limited to – OVS, DPDK, the bmv2 software switch, and the ONOS SDN controller supporting P4 through Stratum.

In the future, we foresee an additional DPU per server, equipped with local ARM and GPU processing as well as a high-performance NIC. Such DPUs are announced by, e.g., NVIDIA (namely the Bluefield-2X/3X boards) and are expected to be available in 2022 or later.

© 2021 - 2023 TeraFlow Consortium Parties 

# References

[1] Vilalta R, de la Cruz JL, López-de-Lerma AM, López V, Martínez R, Casellas R, Muñoz R. uABNO: A Cloud-Native Architecture for Optical SDN Controllers. In2020 Optical Fiber Communications Conference and Exhibition (OFC) 2020 Mar 8 (pp. 1-3). IEEE.

[2] TeraFlow OS controller GitLab: https://gitlab.com/teraflow-h2020/controller

[3] R. Rokui et. al, "Definition of IETF Network Slices," draft-nsdt-teas-ietf-network-slice-definition-02. Work in progress. URL: https://datatracker.ietf.org/doc/html/draft-nsdt-teas-ietf-network-slice-definition-02. Accessed: 2021-12-20.

[4] S. Barguil et. al, "Instantiation of IETF Network Slices in Service Providers Networks," draft-barguil-teas-network-slices-instantation-02. Work in progress. URL: https://datatracker.ietf.org/doc/html/draft-barguil-teas-network-slices-instantation-02. Accessed: 2021-12-20.

[5] S. Barguil et. al, "A Layer 2 VPN Network YANG Model," draft-ietf-opsawg-l2nm-12. Work in progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-l2nm. Accessed: 2021-12-20.

[6] S. Barguil et. al, "A Layer 3 VPN Network YANG Model," draft-ietf-opsawg-l3sm-l3nm-18. Work in progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-l3sm-l3nm. Accessed: 2021-12-20.

[7] A. Farrel et. al, "Framework for IETF Network Slices," draft-ietf-teas-ietf-network-slices-05. Work in progress. URL: https://datatracker.ietf.org/doc/draft-ietf-teas-ietf-network-slices/. Accessed: 2021-12-20.

[8] X. Liu et. al, "IETF Network Slice YANG Data Model," draft-liu-teas-transport-network-slice-yang-04. Work in progress. URL: https://datatracker.ietf.org/doc/draft-liu-teas-transport-network-slice-yang/. Accessed: 2021-12-20.

[9] gRPC Remote Procedure Call. URL: https://grpc.io/. Accessed: 2021-12-20.

[10] YAML: YAML Ain't Markup Language. URL: https://yaml.org/. Accessed: 2021-12-20.

[11] Prometheus client libraries. URL: https://prometheus.io/docs/instrumenting/clientlibs/. Accessed: 2021-12-20.

[12] Git source code management. URL: https://git-scm.com/. Accessed: 2021-12-20.

[13] Jenkins. URL: https://www.jenkins.io/. Accessed: 2021-12-20.

[14] Jenkins remote access API. URL: https://www.jenkins.io/doc/book/using/remote-access-api/. Accessed: 2021-12-20.

[15] Jenkins Pipeline. URL: https://www.jenkins.io/doc/book/pipeline/. Accessed: 2021-12-20.

[16] JenkinsX. URL: https://jenkins-x.io/. Accessed: 2021-12-20.

[17] Reference for the .gitlab-ci.yml file. URL: https://docs.gitlab.com/ee/ci/yaml/. Accessed: 2021-12-20.

[18] GitLab Pricing. URL: https://about.gitlab.com/pricing/. Accessed: 2021-12-20.

[19] GitLab Kubernetes clusters. URL: https://docs.gitlab.com/ee/user/project/clusters/. Accessed: 2021-12-20.

[20] A new era of Kubernetes integrations on GitLab.com. URL: https://about.gitlab.com/blog/2021/02/22/gitlab-kubernetes-agent-on-gitlab-com/. Accessed:

© 2021 - 2023 TeraFlow Consortium Parties 

2021-12-20.

[21]Introduction to GitLab Flow. URL: https://docs.gitlab.com/ee/topics/gitlab_flow.html. Accessed: 2021-12-20.

[22]TeraFlow OS SDN Controller Wiki. URL: https://gitlab.com/teraflow-h2020/controller/-/wikis/home. Accessed: 2021-12-20.

[23]ADRENALINE testbed. URL: http://networks.cttc.cat/ons/adrenaline. Accessed: 2021-12-20.

© 2021 - 2023 TeraFlow Consortium Parties 