



Secured autonomic traffic management for a Tera of SDN flows



D5.2: Implementation of pilots and first evaluation

Deliverable type	R (Report)
Dissemination level	PU (Public)
Due date	31.12.2022
Submission date	12.01.2023
Lead editor	Carlos Natalino (CHAL)
Authors	Lluís Gifre, Ricardo Martínez, Ricard Vilalta, Javier Vilchez, Raul Muñoz, Michela Svaluto, Laia Nadal (CTTC), Alberto Mozo, Amit Karamchandani Batra, Luis de la Cal (UPM), Antonio Pastor, Pablo Armingol, Juan Pedro Fernández Díaz, Óscar González de Dios (TID), Georgios P. Katsikas (UBI) Jose Juan Pedreño (ADVA), Achim Autenrieth (ADVA), Sergio González, Javier Moreno (ATOS), Carlos Natalino (CHAL), Sebastien Andreina, Konstantin Munichev, Giorgia Mason (NEC), Min Xie, Jane Frances Pajo, Abdelhakim Cherifi, Håkon Lønsethagen (Telenor), Mika Silvola (Infinera), Michele Milano, Nicola Carapellese (SIAE), Peer Stritzinger (Stritzinger)
Reviewers	Paolo Monti (CHAL), Thomas Zinner (NTNU)
Quality check team	Adrian Farrel (ODC), Daniel King (ODC)
Work package	WP5

Abstract

This deliverable reports the efforts in three different aspects: *i)* continuously improving the processes adopted for the integration of TeraFlowSDN, *ii)* designing the metrics collection framework for the performance analysis of TeraFlowSDN, and *iii)* refining the scenario description and defining their workflows and deployment. The integration efforts led to the creation of several processes to be adopted by the project and documentation to facilitate ease and speed of TeraFlowSDN deployment. In addition, the metrics collection framework leverages state-of-the-art open-source software to enable easy and insightful monitoring of TeraFlowSDN. Finally, each scenario is detailed, including which KPIs and KVI are relevant to the scenario and the specific workflows and deployments, followed by the preliminary performance evaluation.

[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 License¹.

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 – 2020 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	22.04.2022	Editor	Initial structure of the document
0.2	26.10.2022	Editor	Initial contributions to Scenario 3
0.3	03.11.2022	Amit Karamchandani Batra	Contributions to Scenario 3
0.4	29.11.2022	Ricard Vilalta	Revision of the outline
0.4.1	30.11.2022	Sergio González	Contributions to Section 3
0.4.2	01.12.2022	Alberto Mozo	Contributions to Scenario 3
0.4.3	02.12.2022	Georgios P. Katsikas	Contributions to Scenario 1
0.5	13.12.2022	Editor	Contributions to Scenario 3
0.5.1	19.12.2022	Paolo Monti	Review of Sec. 4
0.5.2	20.12.2022	Partners	Contributions to Scenarios 1 and 2
0.5.3	21.12.2022	Editor	Inclusion of abstract, executive summary, introduction and conclusions
0.5.4	22.12.2022	Partners	Inclusion of content
0.6	29.12.2022	Editor	Initial draft completed
0.6.1	30.12.2022	Daniel King	Document Review
0.6.2	03.01.2023	Paolo Monti	Document Review
0.6.3	05.01.2023	Thomas Zinner	Document Review
0.7	06.01.2023	Ricard Vilalta	Consolidation of Reviews
0.7.1	07.01.2023	Partners	Addressing comments
0.7.2	09.01.2023	Editor	Addressing comments
1.0	16.01.2023	Daniel King	Q/A Review

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

EXECUTIVE SUMMARY

This deliverable summarizes the activities of WP5 during the second year of the TeraFlow project. The objective of this document is to describe the ongoing efforts towards *i)* continuously improving the processes adopted by TeraFlowSDN for integration, *ii)* designing the metrics collection framework that the components for performance assessment can leverage, and *iii)* refining the scenario definition and further detailing how the components will interact with each other to realize the scenario objective. In addition, we also focus on the definition and initial measurement of KPIs and KVs that will quantify the benefits of TeraFlowSDN.

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 document's structure. The second section presents an overview of the TeraFlowSDN architecture. The third section offers an integration report. It describes the modifications adopted by the TeraFlow project over the past year of development, as well as the initiatives and documentation provided to facilitate the introduction of TeraFlowSDN to new users. Finally, in the fourth section, we introduce the metrics collection framework for TeraFlowSDN to consolidate the performance assessment of all the components in a single solution and enable in-depth scalability and performance analysis.

The second half of the document includes sections 5, 6 and 7; these are devoted to the three scenarios used to evaluate TeraFlowSDN. Each scenario introduces its motivation and challenges. The alignment with TeraFlow architecture specifies how the scenario will utilize TeraFlowSDN and which components and use cases are relevant to each scenario. The scenario setup highlights the infrastructure adopted for realizing the scenario and evaluating the performance of TeraFlowSDN. Each scenario also details relevant metrics - in the form of KPIs and KVs - and how these metrics are measured. The workflows and current deployment specify how the components of TeraFlowSDN communicate the functionalities needed by the scenario. A preliminary performance evaluation illustrates the progress towards achieving all the KPIs and KVs specified. Finally, each scenario highlights the pending work that will be the target of the remaining project efforts.

Finally, this deliverable concludes with a short description of the next steps to be adopted by each one of the scenarios.

Table of contents

Executive Summary	4
List of Figures	8
List of Tables	10
Abbreviations	11
1. Introduction	15
1.1. Purpose	15
1.2. Relationship with other Deliverables	15
1.3. Structure	15
2. Architecture Overview	16
3. Integration Report	18
3.1. European Telecommunications Standards Institute (ETSI)	18
3.2. GitLab	19
3.2.1. Feature Request Procedure	19
3.2.2. Feature Lifecycle	22
3.2.3. Bug Report Procedure	23
3.2.4. Wiki	23
3.3. Slack	23
3.4. CI/CD Environment	24
3.5. Release Documentation	27
3.5.1. Installation Instructions	27
3.5.2. Wiki	28
3.5.3. Tutorial and TeraFlowSDN Virtual Machine	29
4. Metrics Collection Framework	31
4.1. Micro-service gRPC Calls	32
4.2. Prometheus	33
4.3. Grafana	35
4.4. Metric Definitions	36
5. Scenario 1: Autonomous Network Beyond 5G	38
5.1. Scenario Introduction	38
5.2. Alignment with TeraFlow Architecture	39
5.3. Scenario Setup	40
5.4. Scenario Metrics	46
5.5. Workflows and Current Deployment	47
5.5.1. Zero-touch Device Automation	47

5.5.2.	L2/L3VPN Service Management and Integration with ETSI OpenSource MANO	49
5.5.3.	Slice Grouping and End to End Slice Provisioning with SLA	51
5.5.4.	Service Restoration with P4 Devices	53
5.5.5.	Energy-efficient Path Computation	56
5.6.	Preliminary Performance Evaluation	57
5.6.1.	Zero-touch Device Automation.....	57
5.6.2.	L3VPN Service Management and Integration with ETSI OpenSource MANO	58
5.6.3.	Slice Grouping and End to End Slice Provisioning with SLA	60
5.6.4.	Service Restoration with P4 devices	62
5.6.5.	Energy-Efficient Path Computation	62
5.7.	Pending Work and Summary	64
6.	Scenario 2: Inter-domain	65
6.1.	Scenario Introduction	65
6.2.	Alignment with TeraFlow Architecture.....	66
6.3.	Scenario Setup	68
6.4.	Scenario Metrics	69
6.5.	Workflows and Current Deployment.....	70
6.5.1.	Inter-domain Provisioning using Transport Network Slices with SLA.....	70
6.5.2.	Distributed Ledger Technologies	71
6.5.3.	Service/Slice Request Scalability.....	72
6.5.4.	Location-aware Service Updates.....	73
6.6.	Preliminary Performance Evaluation	74
6.6.1.	Inter-domain Provisioning using Transport Network Slices with SLA.....	74
6.6.2.	Distributed Ledger Technologies	75
6.6.3.	Service/Slice Request Scalability.....	78
6.6.4.	Location-aware Service Updates.....	78
6.7.	Pending Work and Summary	78
7.	Scenario 3: Cybersecurity.....	79
7.1.	Scenario Introduction	79
7.2.	Alignment with TeraFlow Architecture.....	80
7.3.	Scenario Setup	81
7.3.1.	MouseWorld Setup for Layer 3 Cybersecurity Experiments.....	81
7.3.2.	Emulated Optical Setup for Optical Cybersecurity Experiments	82
7.4.	Scenario Metrics	84
7.5.	Workflows and Current Deployment.....	86
7.5.1.	Layer 3 Cybersecurity.....	87

7.5.2. Optical Cybersecurity	90
7.6. Preliminary Performance Evaluation	94
7.6.1. Layer 3	94
7.6.2. Optical	104
7.7. Pending Work and Summary	109
8. Conclusions and Next Steps	111
References	112

List of Figures

Figure 1. TeraFlowSDN architecture for release 2.0	17
Figure 2. ETSI OSG TFS governance	18
Figure 3. Gitlab For a new feature request	20
Figure 4. Example of new feature request	21
Figure 5. An example of the TeraFlowSDN Slack	24
Figure 6. TeraFlowSDN updated GitLab repository structure	25
Figure 7. Global .gitlab-ci.yml configuration file	26
Figure 8. Code coverage of the Monitoring component	27
Figure 9. List of pages composing the TeraFlowSDN public wiki	28
Figure 10. TeraFlowSDN extended architecture encompassing the metrics collection framework	31
Figure 11. Architecture of the service mesh with sidecar proxy and service container	32
Figure 12. LINKERD dashboard during a Scenario 3 experiment	33
Figure 13. Example of Prometheus exported metrics	34
Figure 14. Screenshot of Prometheus WebUI with metrics collected from Python	35
Figure 15. The Grafana dashboard for the OFC'22 demonstration	36
Figure 16. Scenario 1 high-level architecture	39
Figure 17. Scenario 1 E2E TeraFlow instantiation	40
Figure 18. iFusion Testbed	41
Figure 19. Openstack and IP router scenario interconnected through iFusion Testbed	42
Figure 20. AS7315-30X chassis layout	42
Figure 21. Spirent N12U chassis layout	43
Figure 22. Edgecore DRX-30 chassis layout	43
Figure 23. Dell R730	44
Figure 24. Dell R720xd	44
Figure 25. ASNK ODU radio link	45
Figure 26. Scenario 1 workflow: Adding a device	48
Figure 27. Scenario 1 workflow: Device bootstrap	48
Figure 28. Scenario 1 workflow: Activate Device Monitoring	49
Figure 29. Scenario 1 workflow: Monitor Device Ports	49
Figure 30. Integration of NFV-O and Transport SDN Controller	50
Figure 31. Scenario 1 workflow: NS Provisioning	51
Figure 32. Example of ietf-l2vpn-svc:site-network-access	51
Figure 33. Transport Network Slice grouping on a hierarchical multi-layer SDN scenario	52
Figure 34. Slice grouping sequence diagram	53
Figure 35. Policy-driven service restoration on a P4-based topology	55
Figure 36. Basic network Service Creation relying on the PathComp component output: route and resource selection	57
Figure 37. Screen capture of specific device information obtained after Device Automation	58
Figure 38. OSM screen capture with provisioned NS instance	58
Figure 39. IETF L2VPN Extensions for end-to-end disjoint paths	59
Figure 40. OSM-TFS Wireshark capture to deploy end-to-end network service	59
Figure 41. Example of slice templates	60
Figure 42. Applying slice grouping on new slice request depending on previously deployed slices	61
Figure 43. Elbow method applied to slice grouping	61
Figure 44. Allocated network slices and their slice groups	62

Figure 45. PathComp: REST API requesting a network service with energy-based Context Information	63
Figure 46. Scenario 2 high-level architecture	66
Figure 47. Scenario 2 TeraFlow instantiation in a single domain	67
Figure 48. Interconnected CSWGs at CTTC Testbed	68
Figure 49. Telenor's testbed	68
Figure 50. Scenario 2 workflow: Inter-domain E2E slice provisioning.....	70
Figure 51. PDL proposed architectures, Full PDL (left); Complementary PDL (right)	71
Figure 52. Scenario 2 workflow: Sequence diagram for DLT use.....	72
Figure 53. Scenario 2 workflow: Service Request Scalability.....	73
Figure 54. Scenario 2 workflow: Location-aware Service updates	74
Figure 55. Wireshark capture of Authenticate sequence	75
Figure 56. Inter-domain End-to-End Transport Network Slice deployment.....	75
Figure 57. Transport Network topology for DLT evaluation	76
Figure 58. CDF for the DLT Delay	76
Figure 59. Inter-domain Transport Network Slice that includes sub-slices	77
Figure 60. Sub-slice information details	77
Figure 61. Cybersecurity scenario and threats	79
Figure 62. TeraFlow components used in the cybersecurity scenario	81
Figure 63. Deployment of the cybersecurity scenario focusing on L3.....	82
Figure 64. Simplified view of the emulated deployment.....	83
Figure 65. Scenario 3 workflow: General communication when creating a new service.....	87
Figure 66. Scenario 3 workflow: Traffic Capture and Feature Extraction Workflow	87
Figure 67. Scenario 3 workflow: Attack Detection Workflow (Layer 3).....	88
Figure 68. Scenario 3 workflow: Attack Mitigation Workflow (Layer 3).....	88
Figure 69. Scenario 3 workflow: Cybersecurity KPIs Monitoring Workflow (Layer 3).....	90
Figure 70. Scenario 3 workflow: Initialization of the optical cybersecurity components	91
Figure 71. Scenario 3 workflow: Receiving service events from the Context component	92
Figure 72. Scenario 3 workflow: Periodical optical cybersecurity monitoring using supervised learning	93
Figure 73. Scenario 3 workflow: Periodical optical cybersecurity monitoring using unsupervised learning	93
Figure 74. Energy consumption reduction obtained with each optimization strategy in the inference phase with respect to the non-optimized model using a batch size of 256.	102
Figure 75. Training performance of the ANN for attack detection and identification	105
Figure 76. Confusion matrices for the ANN	106
Figure 77. Number of active optical services (y axis) over time (x axis) in the network as collected by Prometheus.....	107
Figure 78. Time taken for the optical cybersecurity monitoring loop (y axis, in seconds) over time (x axis) as collected by Prometheus.....	108
Figure 79. Average response time over all replicas (y axis, in seconds) of the optical attack detector over time (x axis) as measured by Prometheus.....	109

List of Tables

Table 1. Summary of metrics relevant for the TeraFlow project.....	36
Table 2. KPIs and KVI for the Scenario 1.....	46
Table 3. Target and achieved KPIs and KVI for Scenario 1	64
Table 4. KPIs and KVI for the Scenario 2.....	69
Table 5. Target and achieved KPIs and KVI for Scenario 2	78
Table 6. KPIs and KVI for Scenario 3	84
Table 7. Selected features of the Crypto dataset to train the cryptomining detector.....	96
Table 8. Energy efficiency optimization strategies considered in the experimental framework.....	100
Table 9. Summary of the results of unsupervised learning detecting unknown optical physical layer attacks.....	106
Table 10. Target and achieved KPIs and KVI for Scenario 3	109

Abbreviations

ACL	Access Control List
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASBR	Autonomous System Boundary Router
B5G	Beyond 5G
BGP	Border Gateway Protocol
CAD	Centralized Attack Detector
CCAM	Cooperative, Connected and Automated Mobility
CD	Continuous Delivery
CDF	Cumulative distribution function
CI	Continuous Integration
CO	Central Office
CotS	Commercial off-the-Shelf
DAD	Distributed Attack Detector
DB	Database
DC	Data Centre
DLT	Distributed Ledger Technology
DNN	Deep Neural Network
EAR	Energy-Aware Routing
ECA	Event-Condition-Action
E2E	End-to-End
ETSI	European Telecommunications Standards Institute
FEC	Forward Error Correction
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
gNMI	gRPC Network Management Interface
gRPC	gRPC Remote Procedure Call

IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPM	Intelligent Pluggables Manager
IT	Information Technology
KPI	Key Performance Indicator
KVI	Key Value Item
L2	Layer 2
L2VPN	Layer 2 Virtual Private Network
L3	Layer 3
L3NM	Layer 3 Network YANG Model
L3VPN	Layer 3 Virtual Private Network
LG	Leadership Group
MANO	Management and Orchestration
MDG	Module Development Groups
MEC	Multi-access Edge Computing
ML	Machine Learning
MPLS	Multiprotocol Label Switching
MW	Microwave
NBI	North-Bound Interface
NFV	Network Function Virtualization
ODU	Optical Data Unit
OLS	Open Line System
ONF	Open Networking Foundation
OPM	Optical Performance Monitoring
OS	Operating System
OSG TFS	Open Source Group TeraFlowSDN
OSI	Open Systems Interconnection
OSM	Open-Source MANO
OSS/BSS	Operation Support System/Business Support System

OTA	Over-the-Air
OXC	Optical Crossconnect
PE	Provider Edge
PCEP	Path Computation Element Protocol
PDL	Permissioned Distributed Ledger
PMC	Power Management Controller
RAPL	Running Average Power Limit
ReLU	Rectified Linear Unit
REST	Representational State Transfer
ROADM	Reconfigurable Optical Add/Drop Multiplexer
RPC	Remote Procedure Call
SASE	Secure Access Service Edge
SBI	South-Bound Interface
SDN	Software-Defined Networking
SD-WAN	Software-defined Wide Area Network (SD-WAN)
SLA	Service Level Agreement
SLE	Service Level Expectation
SLO	Service Level Objective
TAPI	Transport API
TSC	Technical Steering Committee
TF	Task Forces
TLS	Transport Layer Security
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
TR	Technical Reference
VIM	Virtual Infrastructure Manager
VNF	Virtualized Network Functions
VPN	Virtual Private Network
WAN	Wide Area Network

YAML	YAML Yet Another Markup Language
-------------	----------------------------------

1. Introduction

The second version of TeraFlowSDN delivers state-of-the-art open-source cloud-native Software-Defined Networking (SDN). It provides efficient, reliable, scalable, and flexible control for B5G (Beyond 5G) networks. In this context, it is crucial to ensure that TeraFlowSDN correctly interacts with existing networking devices and can take advantage of existing protocols. WP5 performs the TeraFlowSDN integration, followed by experimentation, validation, and evaluation using a range of KPIs and KVIIs.

Given the highly distributed nature of the TeraFlowSDN development, it is vital to enumerate, evaluate, and select suitable techniques, processes, and tools that can be used to assist the partners during the development of TeraFlowSDN components and scenarios. Moreover, the adopted setup needs to support collaboration among all partners while ensuring the consistency and reliability of the resulting software.

The project leverages the infrastructure available at the partners' premises to build testbed setups, realizing three scenarios described in this deliverable. The scenarios are first described, highlighting their context and motivation. Then, details of the scenarios related to the setup, metrics, workflows, deployments, and preliminary performance evaluation are presented.

1.1. Purpose

The purpose of D5.2 is threefold. The first objective is to describe the development and progress made since D5.1 in terms of code integration, documentation, and development environment. The second objective is to report the design of the metrics collection framework developed for TeraFlowSDN, which enables partners and users to obtain a detailed performance analysis of the internal TeraFlowSDN components, potentially enabling further code optimizations. Finally, this deliverable reports on the efforts in refining and implementing the scenarios.

1.2. Relationship with other Deliverables

D5.2 takes input from MS2.2, where new details on use cases, and requirements via feedback, have been defined, including an updated architecture of TeraFlowSDN. Moreover, the components tested in the scenarios reported in this deliverable are thoroughly described in D3.2 and D4.2.

1.3. Structure

This deliverable is structured as follows. Section 2 presents an architectural overview of TeraFlowSDN. Section 3 offers an integration report summarising our most recent code integration efforts and documentation. Section 4 introduces the metrics collection framework developed for TeraFlowSDN, which uses state-of-the-art open-source software to provide a comprehensive monitoring framework for TeraFlowSDN and its components. Sections 5, 6, and 7 detail the three scenarios used to evaluate the performance of TeraFlowSDN. Each section introduces the context and motivation for the scenario, the partner setup to test TeraFlowSDN, the relevant metrics, workflows, and deployment, and a preliminary performance evaluation. Finally, section 8 concludes this deliverable by presenting final remarks and describing the next steps for each one of the initiatives related to WP5.

2. Architecture Overview

A detailed architecture description for TeraFlowSDN release 2.0 is provided in D2.2. In this section, we briefly describe the overall architecture to make the deliverable self-contained by introducing the main design of TeraFlowSDN.

The SDN controller cloud-native architecture consists of stateless micro-services interacting with each other to fulfill network management tasks, in addition to a few stateful micro-services responsible for keeping the state of the network. TeraFlowSDN relies on Kubernetes to handle the containers supporting the micro-services. Kubernetes is a state-of-the-art container orchestrator that provides a broad set of management capabilities and can operate geographically distributed infrastructures.

Figure 1 shows the proposed micro-service-based architecture. Following the design principles from cloud-native applications, each component is implemented as a micro-service that is able to export a set of Remote Procedure Call (RPC) services to other components. Each micro-service can be instantiated once or with multiple replicas, which allow the application of load balancing techniques. By adopting stateless micro-services, requests can be handled by any replica of the micro-service. Load balancing works by establishing an endpoint that will receive all the requests for a service. The endpoint acts as a load balancer by delegating each request to one of the replicas of the service. The load balancer is also responsible for keeping track of the replicas, i.e., tracking the addition and deletion of replicas and updating its internal list of replicas. Depending on the RPC implementation adopted, we may use the built-in Kubernetes load balancer, or adopt an external one. Each replica is composed of a Pod, i.e., a collection of containers that are managed by Kubernetes as a single entity. More information is provided in Sec. 4.1.

Context component stores the network configuration (e.g., topologies, devices, links, services) and its status as managed by the TeraFlowSDN components in a No-SQL database to optimize concurrent access. Internally, it implements a Database API enabling to switch between different backends. The TeraFlowSDN controller uses its North-Bound Interface (NBI) component (previously known as Compute) to receive Layer 2 Virtual Private Network (L2VPN) requests and convert them to necessary connectivity services or Transport Network Slices via the Slice and Service components. The Service component is responsible for selecting, configuring, and deploying the requested connectivity service through the South-Bound Interface (SBI). To this end, the SBI component interacts with the network equipment through pluggable drivers. In addition, a Driver Application Programming Interface (API) has been defined to facilitate the addition of new network protocols and data models to the SBI component. The Automation component implements several Event-Condition-Action (ECA) loops defining the automation procedures in the network. Monitoring manages the different metrics configured for the network equipment and services, stores monitoring data related to selected Key Performance Indicators (KPI), and provides means for other components to access the collected data. Internally, the Monitoring component relies on a database to store the monitoring data as time series, exploiting its powerful querying and aggregation mechanisms for retrieving the collected data.

North-Bound Interface (NBI) component serves as the interface from internal gRPC (gRPC Remote Procedure Call) and protocol buffers towards external Representational State Transfer (REST)-like requests. It provides a Representational State Transfer API (REST-API)-based to NBI external systems, such as Network Function Virtualization (NFV) and Multi-access Edge Computing (MEC) frameworks. We also include a Web-based User Interface (WebUI) that uses the gRPC-based interfaces made available by the TeraFlowSDN components to inspect the network state and issue operational requests to the TeraFlowSDN components.

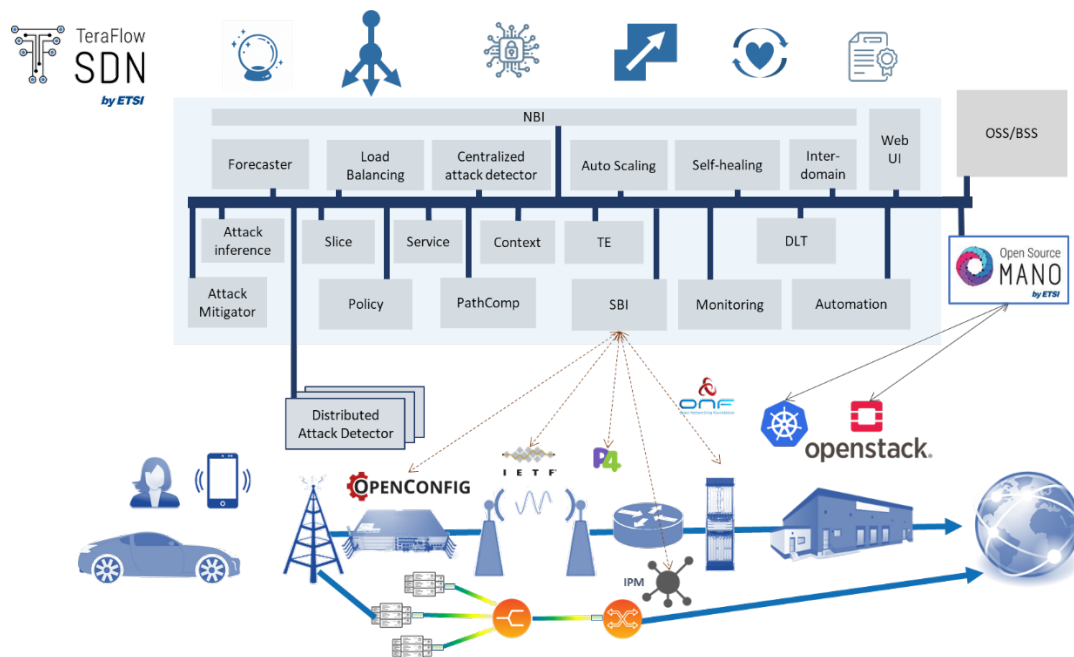


Figure 1. TeraFlowSDN architecture for release 2.0

TeraFlowSDN Release 2 provides extended and validated support for OpenConfig-based routers and interaction with optical SDN controllers through the Open Networking Foundation (ONF) Transport API (TAPI). Moreover, TeraFlowSDN release 2 includes complete integration for microwave network elements (through the Internet Engineering Task Force - IETF - network topology YANG model), and Point-to-Multipoint integration of XR optical transceivers and P4 routers. New features for P4 routers include loading a P4 pipeline on a given P4 switch; getting runtime information (i.e., flow tables) from the P4 switch; and pushing runtime entries into the P4 switch pipeline, thus allowing total usage of P4 switches.

Service Level Agreement (SLA) validation has been re-engineered through all the workflows, from Device monitoring to Service and Slice life cycle management. Thus, the Slice, Service, Policy, Device and Monitoring Components have been updated to support the necessary network automation workflows. Moreover, Slice grouping has also been introduced, along with the Path Computation Component. This component allows new use cases, such as energy-aware service placement.

Cybersecurity mechanisms have been updated, including novel components for attack detection (either distributed or centralized), attack inference, and attack mitigation. In addition, several novel use cases are supported. Distributed Ledger Technology (DLT) has also been extended to interact with the Inter-domain Component and use the deployed Hyperledger Fabric.

3. Integration Report

The previous section outlined the TeraFlowSDN architecture which is also the reference point for the integration process. In the following, we will overview the tools used for the component integration and some helpful documentation and instructions for installing the TeraFlowSDN controller.

We will first present the new GitLab repository hosted by European Telecommunications Standards Institute (ETSI), including all the new features in addition to the ones inherited from the previous repository. Then, we will introduce our primary communication channel, i.e., Slack, followed by a summary of our Continuous Integration/Continuous Delivery (CI/CD) environment and its updates since D5.1. Finally, we will overview the different documentation available for the users.

3.1. European Telecommunications Standards Institute (ETSI)

The ETSI Open Source Group (OSG) TeraFlowSDN (TFS) defines a framework for developing a cloud-native SDN controller for high-capacity networks aiming to support future networks beyond 5G. Based on a cloud-native, micro-services architecture, the software will be able to integrate with existing frameworks (NFV, MEC) and provide a toolbox for different ETSI groups to experiment with new features for flow aggregation, management (service layer), network equipment integration (infrastructure layer), AI/ML (Artificial Intelligence/Machine Learning)-based security, and forensic evidence for multi-tenancy.

The OSG TFS is established on the initiative of a group of ETSI Full, Associate and Applicant members. The project is responsible for defining its own detailed [Working Procedures](#) within the limits of the [ETSI Terms of Reference](#). It will also be responsible for the validation of the source code it produces, together with any associated documentation.

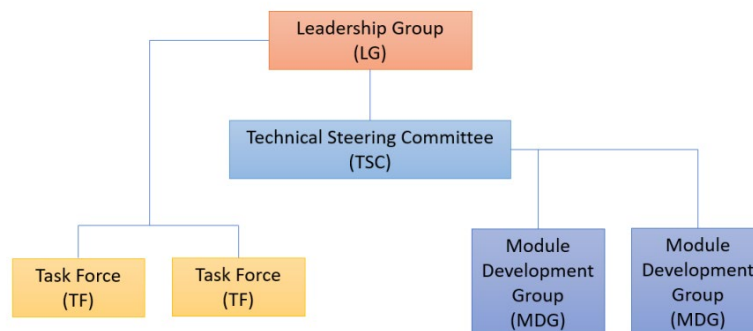


Figure 2. ETSI OSG TFS governance

The governance of OSG TFS is shown in Figure 2 and it is mainly structured into the following bodies:

- Leadership Group (LG)
- Technical Steering Committee (TSC)
- Module Development Groups (MDG)
- Dedicated Task Forces (TFs)

Organizations participating in these bodies and involved in code contributions shall sign the OSG TFS Member or Participant agreements to guarantee their adherence to the Terms of Reference and

license(s) used in the project.

For more detailed information, we suggest to access <https://tfs.etsi.org/legal/>

3.2. GitLab

After the creation of the ETSI TeraFlowSDN community, we have moved from the original Gitlab towards a GitLab hosted inside the ETSI Laboratories:

<https://labs.etsi.org/rep/tfs/controller>

All features from the previous GitLab repository are available in the new one.

This section presents the features used and the project-defined workflows for feature requests, feature lifecycle, bug reporting and technical wiki.

3.2.1. Feature Request Procedure

Two kinds of feature requests are considered in this procedure:

- New Feature: a big change that potentially affects several components and requires an appropriate design phase;
- Enhancement: a relatively small change enhancing TFS that does not require a design phase.

Project features go through a discussion and approval process. To propose both types, TFS uses the issues on its GitLab code repository.

- Important: a feature request is about functionality, not about implementation details;
- Please describe WHAT you are proposing and WHY it is important;
- DO NOT describe HOW to do it. This is done when the new feature is approved by TSC by populating the design details.

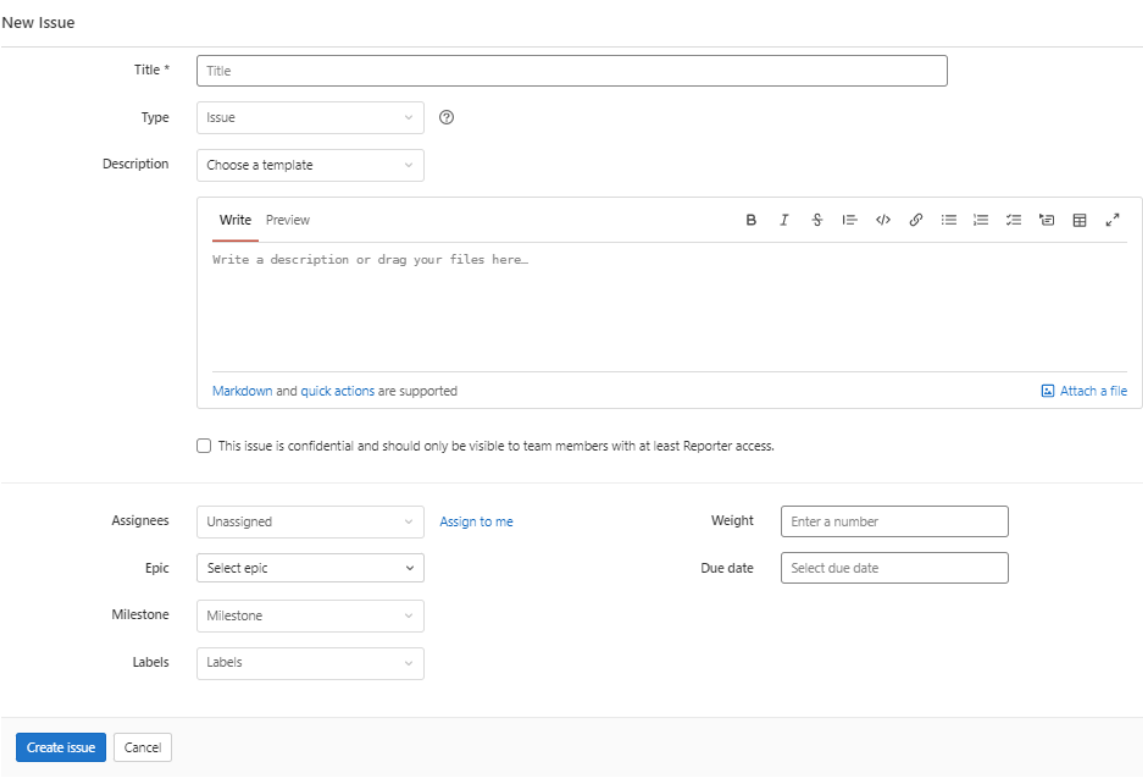
3.2.1.1. Procedures

1. Go to New Issue page

<https://labs.etsi.org/rep/tfs/controller/-/issues/new>

You need to be authenticated.

2. Create a New Issue for your feature. Figure 3 shows the Gitlab form for new feature request.



New Issue

Title *

Type Issue ?

Description Choose a template

Write Preview

Write a description or drag your files here...

Markdown and quick actions are supported

Attach a file

☐ This issue is confidential and should only be visible to team members with at least Reporter access.

Assignees Unassigned Assign to me

Epic Select epic

Milestone Milestone

Labels Labels

Weight Enter a number

Due date Select due date

Create issue Cancel

Figure 3. Gitlab For a new feature request

- Title: A concise high-level description of the feature (see some other examples in GitLab)
 - Type: Issue
 - Description: Choose the "new-feature" or "enhancement" project templates and fill-in the auto-generated template describing the feature/enhancement.
 - Labels:
 - o Select the type of request: type::new-feature / type::enhancement
 - o If you foresee the components affected by the request, please pick the appropriate labels.
 - Component labels have the form comp-<component-name>.
 - o PLEASE: Do not set other types of labels (to be set by TSC).
 - PLEASE: Do not set the following fields (to be set by TSC): EPIC, Assignee, Milestone, Weight, Due Date
 - Submit the Issue
3. Interact with the TSC and the Community throughout the issue.

TSC will review your request. It will be approved if it makes sense and its purpose is clear. Otherwise, TSC will provide questions for clarification.

3.2.1.2. Designing a Feature

Once a feature has been approved, the design phase starts. The design should be included within the feature description (GitLab issue description) by concatenating the Design Feature Template (see below) and correctly filling it in. If the feature description becomes too long, attached files could also be submitted.

The design is expected to be socialized with the relevant stakeholders (e.g. MDLs and TSC). Dedicated slots can be allocated in the TECH calls on a per-request basis to discuss and refine it.

For writing the design, you can check the design of existing features or use the design templates below.

3.2.1.3. New Feature / Enhancement Request Template

The following describes the template for new features or enhancements in TFS.

```
# Proposers
- name-of-proposer-1 (institution-of-proposer-1)
- name-of-proposer-2 (institution-of-proposer-2)
...
# Description
Describe your proposal in ~1000 characters.
You can reference external content listed in section "References" as [Ref-1].
# Demo or definition of done
Describe which high level conditions needs to be fulfilled to demonstrate this feature
implementation is completed.
You can reference external content (example, demo paper) listed in section "References" as [Ref-
2].
# References
1. [Reference name](https://reference-url)
2. Author1, Author2, Author3, et. al., "My demo using feature," in Conference-Name Demo Track,
20XX.
```

The Figure 4 below shows an example of a new feature request.

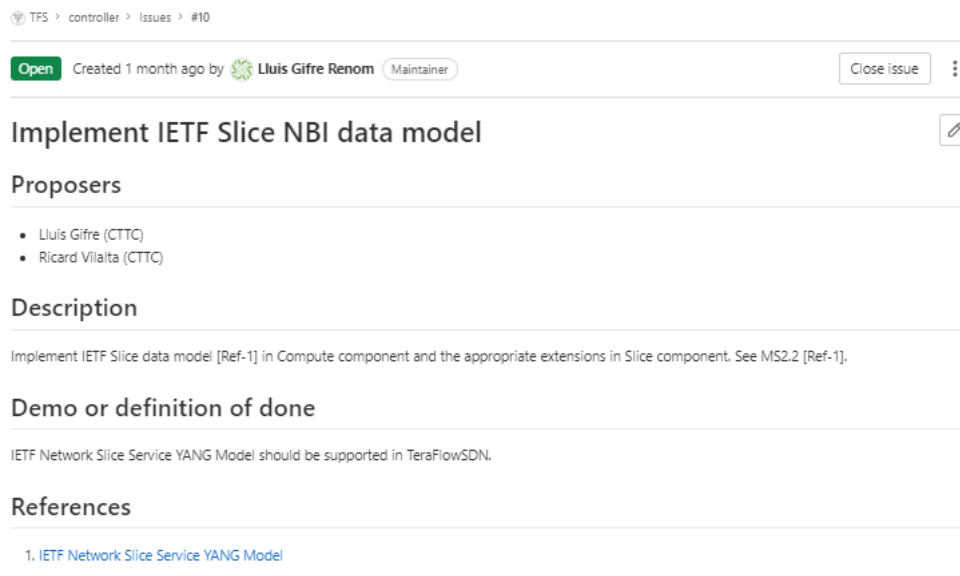


Figure 4. Example of new feature request

3.2.1.4. Feature Design Template

The following text describes a Feature design template to be completed when submitting a feature request.

```
# Feature Design
## Clarifications to Expected Behavior Changes
Existing component logic and workflows between components that need to be altered to realize
this feature.
```

Remember to justify these changes.

...

References

List of relevant references for this feature.

...

Assumptions

Enumerate the assumptions for this feature, e.g., fix XXX is implemented and merged, specific configurations, specific components deployed.

...

Impacted Components

List of impacted components: Context, Device, Service, PathComp, Slice, Monitoring, Automation, Policy, Compute, etc.

Just an enumeration, elaboration of impacts is done below.

Component1 Impact

Describe impact (changes) on component1.

...

Component2 Impact

Describe impact (changes) on component2.

...

Testing

Describe test sets (unitary and integration) to be carried out.

This section can include/reference external experiments, demo papers, etc.

...

3.2.2. Feature Lifecycle

Once approved, a feature request could transition through the following steps:

- Approved: Feature approved by TSC; design phase can start;
- Design: Feature under design; discussing on HOW to do it. Only for New Features;
- Development: Design approved; feature under development/implementation;
- Testing and Review: Feature implemented and under review and testing by the developers and the community;
- Completed: Testing and review completed, and feature merged;
- Abandoned: Feature abandoned.

Important: An approved feature is not a guarantee for implementation.

Implementing a feature requires resources, and resources come from the members, participants and individual contributors within the TFS Community, which might have prioritized the development of other features based on their interests and the interests expressed by the LG, the TSC, and the MDGs.

Once a Feature is mature, e.g., Testing, Review, Completed, it can be accepted for inclusion in a specific Release.

This is accomplished by including the issue ticket in the respective EPIC "ReleaseX.Y".

For instance, to see the features included in Release X.Y, check EPIC "ReleaseX.Y".

3.2.3. Bug Report Procedure

Project bugs go through a review, confirmation, and resolution process. TFS uses the issues on its GitLab code repository for bug reporting and tracking. Important: New bugs must be properly documented. Details are requested on the details on the deployment environment (Operating System, MicroK8s, etc.), the TFS version (or branch/commit), the TFS deployment settings (components, particular configurations, etc.), the particular sequence of actions that resulted in the bug, the TFS components affected by the bug (if you know them), the expected behaviour (if you know it).

Without this minimal information, it might be difficult to reproduce and resolve the bug and validate the solution's completeness.

The reporting procedure is described as follows.

1. Go to New Issue page <https://labs.etsi.org/rep/tfs/controller/-/issues/new>.

You will then need to be authenticated.

2. Create a New Issue for your bug
 - Title: A concise high level description of your bug (see some other examples in GitLab)
 - Type: Issue
 - Description: Choose the "bug" project template and fill-in the auto-generated template describing the bug.
 - Labels:
 - Select the type of request: type::bug
 - If you foresee the components affected by the bug, pick the appropriate labels for them.
 - Component labels have the form comp-<component-name>.
 - PLEASE: Do not set other types of labels (to be set by TSC).
 - PLEASE: Do not set the following fields (to be set by TSC): EPIC, Assignee, Milestone, Weight, Due Date
 - Submit the Issue
3. Interact with the TSC and the Community through the issue.

The TSC will review the reported bug and try to reproduce it. If we succeed in reproducing it, we will mark it as confirmed, and include its resolution in the development plans. Otherwise, TSC will provide questions for clarification.

3.2.4. Wiki

The documentation using the Git Wiki page is described later in Section 3.5.2.

3.3. Slack

Slack [SLK] is the main communication platform used by the TeraFlowSDN project consortium. Slack is an instant messaging platform with different add-ins and workplace tools, or as they say, *"a single place for messaging, tools, and files"*. Slack provides two communication methods: topic-based group chats and direct person-to-person chats. In the TeraFlowSDN Slack, we set multiple channels for different purposes. Some of them are public to the consortium, and some others are private for direct communication among a reduced number of partners.

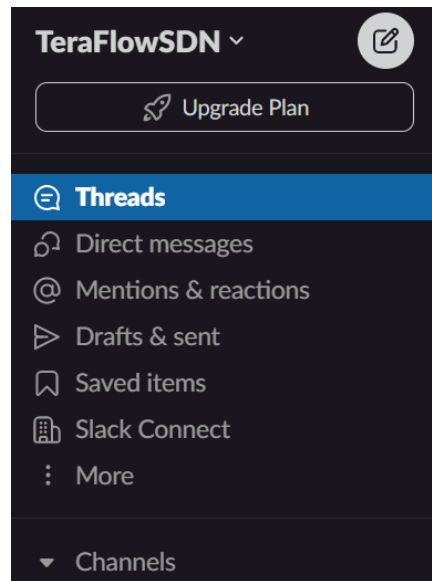


Figure 5. An example of the TeraFlowSDN Slack

Figure 5 depicts an example of the slack interface. Our main communication channel is the *#general* channel, where all the consortium members are participants. There are also dedicated channels for specific task purposes, such as the *#wp3*, *#cicd*, *#release-v1*, or *#integration-demo-validation-wp5*, where only the contributing partners are participants. We also have private channels like *#atos-cttc* and *#ubitech-atos* devoted to straightforward communication between partners, especially useful for component integration purposes. In addition to the channels where multiple users can participate, there are direct messages that can be used to contact any consortium person directly.

It is worth noting that Slacks provides a formatting toolbar with dedicated formatting options for code and code blocks, which is especially useful for software development and component integration. In addition to the chats, Slack also provides tools for audio/video calls and file sharing. The consortium uses these options less since we usually use Microsoft Teams for resource-sharing purposes.

3.4. CI/CD Environment

The CI/CD environment is still based on GitLab CI, fully integrated within GitLab, which is the source code management tool used in the project. The CI/CD methodology was already described in D5.1, presenting the infrastructure, the GitFlow, the branch naming schema, and the testing methodology. However, some minor changes have been applied to the project file structure since the release of D5.1. Figure 6 shows the updated repository structure. As can be seen, the root folder is named *controller* and contains the following files and folders:

- *proto*: this folder contains the data models of the TeraFlowSDN components in a *.proto* file;
- *manifests*: this folder contains the Kubernetes manifest files for the deployment of the TeraFlowSDN services in a Kubernetes environment;
- *src*: this folder contains the source code of the micro-services with the following folders inside:
 - *common*: this folder contains some common resources for the development of the micro-services (*tools* folder, multiple scripts, and *logger.py*) as well as symbolic links to the compiled protocol buffers that are generated during the installation phase inside the */proto/src* folder

- *uService_i*: this folder contains the source code of the *uService_i*. Within this folder, you can find the *client*, *service*, and *tests* folders, the *Dockerfile* and the *.gitlab-ci.yml* file for the Gitlab CI pipeline of that micro-service.
- *.gitlab-ci.yml*: this is the global Gitlab CI configuration file. It defines the stages of the CI/CD pipeline and includes each micro-service's individual *.gitlab-ci.yml* files;
- *Deployment scripts*: the root folder also contains a set of scripts for the deployment of the TeraFlowSDN controller into a Kubernetes infrastructure;
- *INSTALL.md* and *README.md*: installation and readme files for the users.

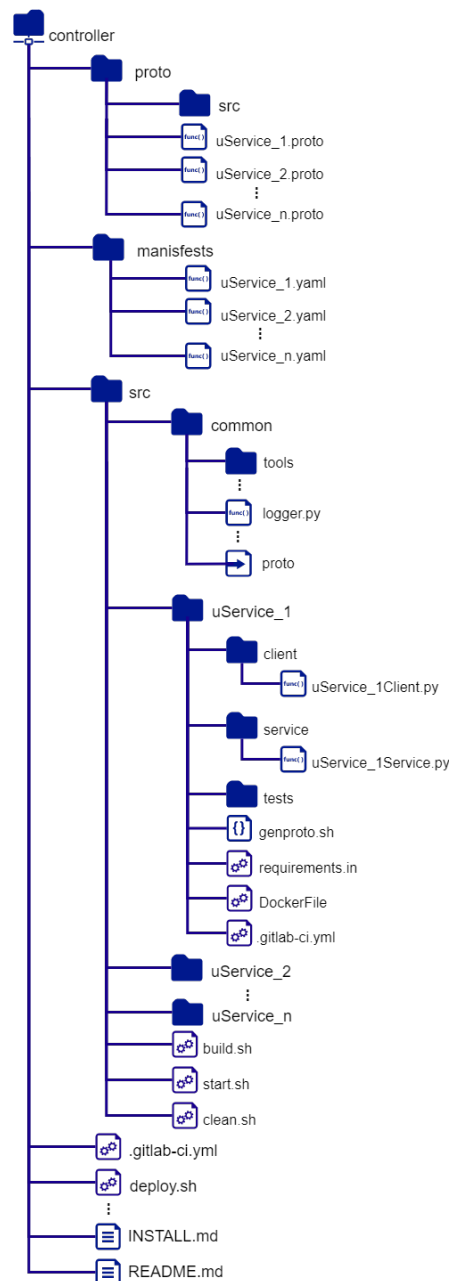


Figure 6. TeraFlowSDN updated GitLab repository structure

There are two main components in a Gitlab CI pipeline, *jobs*, and *stages*. Jobs define *what* to do and stages *when* to execute the jobs. Stages are sequential and determine the order in which jobs should be completed. As can be seen in Figure 6, the Gitlab CI configuration file (*.gitlab-ci.yml*) is hierarchically structured, defining the stages of the pipeline in the global Gitlab CI configuration file

located in the *root* folder (Figure 7) and defining the jobs for each micro-service in the individual files located under the */src/uService_i* folder that are included as local files in the global *.gitlab-ci.yml*.

```
# stages of the cicd pipeline
stages:
  - dependencies
  - build
  - unit_test
  - integ_test
  - deploy
  - funct_test

# include the individual .gitlab-ci.yml of each micro-service
include:
  - local: '/manifests/.gitlab-ci.yml'
  - local: '/src/monitoring/.gitlab-ci.yml'
  - local: '/src/compute/.gitlab-ci.yml'
  - local: '/src/context/.gitlab-ci.yml'
  - local: '/src/device/.gitlab-ci.yml'
  - local: '/src/service/.gitlab-ci.yml'
  - local: '/src/dbscanserving/.gitlab-ci.yml'
  - local: '/src/opticalattackmitigator/.gitlab-ci.yml'
  - local: '/src/opticalcentralizedattackdetector/.gitlab-ci.yml'
  - local: '/src/automation/.gitlab-ci.yml'
  - local: '/src/policy/.gitlab-ci.yml'
  - local: '/src/webui/.gitlab-ci.yml'
  - local: '/src/l3_distributedattackdetector/.gitlab-ci.yml'
  - local: '/src/l3_centralizedattackdetector/.gitlab-ci.yml'
  - local: '/src/l3_attackmitigator/.gitlab-ci.yml'
  - local: '/src/slice/.gitlab-ci.yml'
  - local: '/src/interdomain/.gitlab-ci.yml'
  - local: '/src/pathcomp/.gitlab-ci.yml'
  - local: '/src/dlt/.gitlab-ci.yml'
```

Figure 7. Global *.gitlab-ci.yml* configuration file

As illustrated in Figure 7, in the TeraFlowSDN CI/CD pipeline, we defined the following six stages:

- *dependencies*: this stage is devoted to deploying the dependency services of the TerFlowSDN Kubernetes cluster;
- *build*: this stage is in charge of building the micro-services and uploading the images to the Gitlab container registry;
- *unit_test*: this stage is dedicated to the unit testing of the micro-services, i.e., testing isolated, small portions of code of individual micro-services;
- *integ_test*: this stage is associated with integration testing, aiming to check whether multiple micro-services can properly work together;
- *deploy*: this stage has been created for deploying the micro-service in the development infrastructure to perform end-to-end testing;
- *funct_test*: this stage is dedicated to functional testing, aiming to find any problems in fulfilling an end-to-end function.

Therefore, each micro-service implements its jobs based on the six stages defined in the global configuration file. To properly assure the testing stages' effectiveness and facilitate the integration between components, we measure the *code coverage*, a software testing metric specifying the code percentage and the number of code lines successfully validated during the testing phase. An example of code coverage of the monitoring component can be found in Figure 8. The report shows the code coverage of each file and the lines that have not been tested as well as the total code coverage of the monitoring component (e.g., in the Monitoring component is 78%). At the moment of writing this report, the code coverage reported in the *master* branch is 87%.

Name	Stmts	Miss	Cover	Missing
monitoring/_init_.py	0	0	100%	
monitoring/client/MonitoringClient.py	139	7	95%	171-174, 178-183
monitoring/client/_init_.py	0	0	100%	
monitoring/service/AlarmManager.py	30	1	97%	37
monitoring/service/EventTools.py	59	9	85%	42-46, 59-60, 94-95
monitoring/service/ManagementDBTools.py	207	60	71%	28-30, 48-50, 66-68, 86-88, 105-106, 120-124, 141-142, 150-151, 157-158, 166-167, 173-174, 182-183, 189-190, 199-202, 211-214, 223-227, 234-235, 242-243, 250-251, 259-260, 269-270, 278-279, 287-288
monitoring/service/MetricsDBTools.py	241	124	49%	50-51, 54-55, 58-59, 62-63, 66-67, 70-72, 75-76, 79-80, 123-126, 150-175, 184, 189-190, 201, 206-207, 216-217, 226-227, 236-237, 246, 250-317, 320-321
monitoring/service/MonitoringService.py	12	0	100%	
monitoring/service/MonitoringServiceServiceImpl.py	420	106	75%	93, 101-104, 120, 122-124, 135, 145-147, 174-176, 189, 205-208, 238-240, 242-245, 271-272, 293-295, 334-336, 348, 359-361, 385-387, 401, 403-405, 428, 438-440, 467-469, 494-495, 497-499, 534-544, 548-554, 570, 572-574, 580-590, 595-619
monitoring/service/SubscriptionManager.py	28	3	89%	38, 52-53
monitoring/service/_init_.py	0	0	100%	
monitoring/tests/Messages.py	102	21	79%	21-23, 37-45, 73-75, 107-109, 126-130, 146-148
monitoring/tests/Objects.py	8	0	100%	
monitoring/tests/_init_.py	0	0	100%	
monitoring/tests/test_unitary.py	367	18	95%	174-176, 188-201, 313, 376-377, 380
TOTAL	1613	349	78%	

Figure 8. Code coverage of the Monitoring component

3.5. Release Documentation

In this subsection, we provide an overview of the documentation provided with the release of the TeraFlowSDN controller. Then, we provide the installation instruction to facilitate the execution of an instance of the software locally in any environment. We also present the TeraFlowSDN wiki and some tutorials to test the functionalities of the software. Finally, we introduce the TFS virtual machine we have created with an environment that contains all the requirements for adequately deploying the TeraFlowSDN controller.

3.5.1. Installation Instructions

TeraFlowSDN is based on micro-service architecture and is composed of multiple containers. For properly orchestrating these containers, we recommend installing TeraFlowSDN in a Kubernetes cluster. There are multiple Kubernetes distributions. For the sake of simplicity, we recommend MicroK8s, a powerful, lightweight, and reliable Kubernetes distribution that offers multiple add-ons out of the box with a minimal disk and memory footprint. A guided tutorial on installing MicroK8s can be found in the Wiki described in Sec. 3.5.2.

Once a Kubernetes distribution is installed, it is required to clone the repository from the ETSI-hosted GitLab (<https://labs.etsi.org/rep/tfs/controller.git>). Once the repository is cloned, the desired branch must be selected. By default, the repository points to the *master* branch.

The next step is to prepare the environment for the deployment of the TeraFlowSDN controller. For this purpose, we designed a script (i.e., *my_deploy.sh*) that sources the deployment settings. This script can be found in the repository root folder. Once the environment is ready, the controller can be deployed using the *deploy.sh* script in the root directory of the repository. This script builds, tags, and pushes the images to the repository of the local Kubernetes distribution. Additionally, the script creates a dedicated namespace for the deployment, deploys all the micro-services, creates an ingress controller for the WebUI, and initializes the Grafana dashboard. A more detailed tutorial with all the installation instructions can be found in the Wiki described in Sec. 3.5.2.

3.5.2. Wiki

The TeraFlowSDN offers a comprehensive set of guidelines in the form of a wiki page hosted in the GitLab of the project. The wiki provides new and experienced users with reference material that can be followed when new installations of TeraFlowSDN are being created or when required to reproduce experiments. Figure 9 shows the list of pages currently available at the TeraFlowSDN wiki. The wiki pages are constantly being developed and updated to reflect the latest developments of the components.

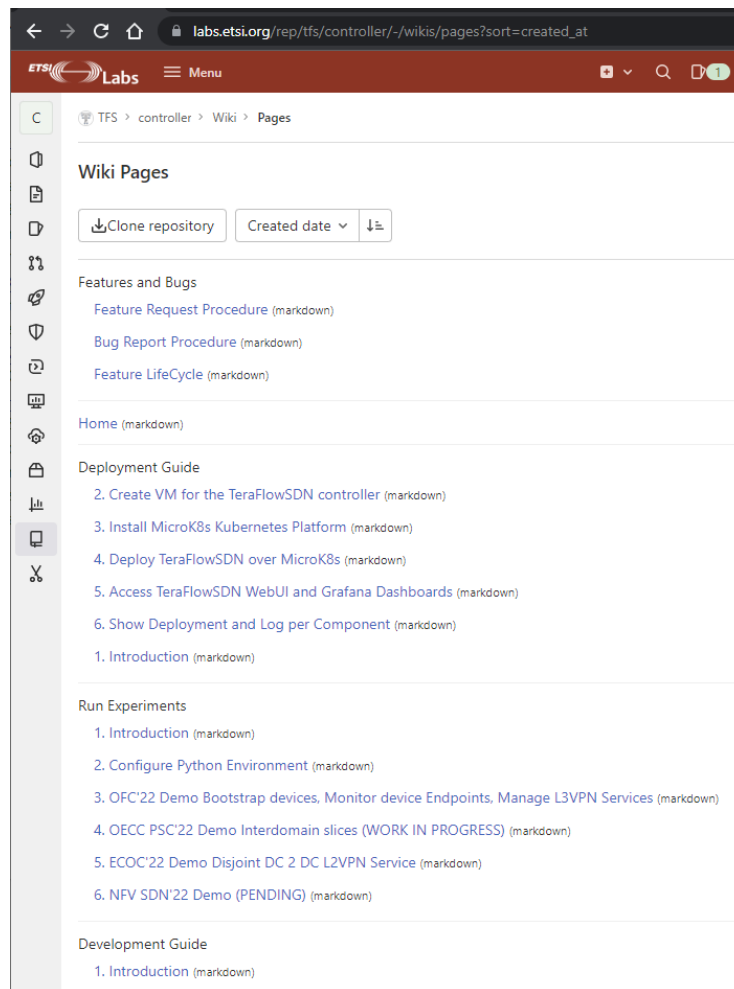


Figure 9. List of pages composing the TeraFlowSDN public wiki

There are currently four categories of pages:

1. **Deployment guide:** These pages detail how to set up an environment (i.e., install a Linux OS in a virtual machine or bare metal), install dependencies, and deploy TeraFlowSDN in its reference architecture;
2. **Run experiments:** These pages show how to reproduce experiments using TeraFlowSDN. Most of the currently available experiments are related to the demonstrations presented at conferences. However, for the final version, we will also include specific documentation on how to run the scenario experiments;
3. **Features and bugs:** These pages document how new feature requests should be placed and how bugs should be reported;

4. *Development guide*: These pages are currently under development. The main objective is to document how to create a new component from scratch using the main languages supported by TeraFlowSDN, e.g., Python and Java.

3.5.3. Tutorial and TeraFlowSDN Virtual Machine

To onboard users to ETSI TeraFlowSDN, we prepared an intuitive virtual tutorial, which includes all the basic knowledge to understand and use ETSI TeraFlowSDN.

This tutorial offers an overview and hands-on experience in programming the necessary tools to control and monitor the packet optical networks while introducing ETSI TeraFlowSDN as the cloud-native SDN controller that enables innovative connectivity services for future networks beyond 5G. Furthermore, this new class of cloud-native SDN controllers allows rapid prototyping and experimentation in R&D and standardization activities.

First, an overview of the YANG data modelling language and NETCONF protocol is presented. Later, TeraFlowSDN controller is introduced. Then, we detail the dynamic establishment of L3VPN using OpenConfig routers. Later, RESTconf interfaces are explained, and ONF Transport API is exploited to obtain network information.

The tutorial enables participants to:

- Learn and use open-source tools to control and monitor packet optical networks;
- Develop simple code for NETCONF agents and clients, including learning to create the necessary bindings;
- Understand OpenConfig data models and how to use them to control and monitor network equipment;
- Obtain practical hands-on experience on RESTconf-based interfaces for Control of Transport Networks;
- Develop a monitoring application using gRPC and gNMI (gRPC Network Management Interface) protocols;
- Understand and implement publish/subscribe mechanisms for data using Kafka broker.

This tutorial is prepared for the following audience:

- Network Operators and Service providers who want to get first-hand operational experience with TeraFlowSDN Controller;
- System Integrators who want to develop their expertise with TeraFlowSDN;
- Academia and Universities who are using or considering TeraFlowSDN as a platform for their research activities in networking;
- TeraFlowSDN developers and users that want to share and test with the community;
- Members of other research projects that may be interested in using TeraFlowSDN Controller in their research and proof-of-concept activities.

The tutorial recordings can be found at:

- 1) [Controlling and Monitoring Optical Networks, by Lluís Gifre and Ricard Vilalta \(CTTC\)](#)
- 2) [Introduction ETSI TeraFlowSDN, Deployment, Onboarding Network Devices, Programmable L3 Routers, by Ricard Vilalta and Lluís Gifre \(CTTC\)](#)
- 3) [Introduction to P4 and a mini P4 demo, by Georgios P. Katsikas and Panagiotis Famelis \(UBITECH\)](#)

The tutorial slides are available at:

- 1) [Controlling and Monitoring Optical Networks, by Lluís Gifre and Ricard Vilalta \(CTTC\)](#)
- 2) [Introduction ETSI TeraFlowSDN, Deployment, Onboarding Network Devices, Programmable L3 Routers, by Ricard Vilalta and Lluís Gifre \(CTTC\)](#)
- 3) [Introduction to P4 and a mini P4 demo, by Georgios P. Katsikas and Panagiotis Famelis \(UBITECH\)](#)

The necessary TFS Virtual Machine to follow tutorial can be downloaded at:
<https://www.dropbox.com/s/gbqyybdv6nndufn/TFS-HF-VM.rar?dl=0>

More information is available at:

<https://labs.etsi.org/rep/groups/tfs/-/wikis/TFS-HACKFEST-1>

4. Metrics Collection Framework

TeraFlowSDN offers a built-in metrics collection framework that can benchmark and monitor the performance of the internal components. Unlike the *Monitoring component*, which is responsible for collecting KPI data from the infrastructure, the metrics collection framework is concerned with KPI data coming from the TeraFlowSDN components. For instance, we may want to monitor how long a specific method takes to run (e.g., how long does the *Context* component take to reply with a list of current services?).

This section introduces the Metrics Collection Framework and suggests several metrics definitions that are relevant for the different scenarios. Later, these metrics are detailed in next sections on per-scenario basis.

The metrics collection framework is developed by integrating state-of-the-art open-source software into the TeraFlowSDN architecture. As illustrated in Figure 10, two leading open-source software platforms are used:

1. *Prometheus*: a solution for exposing and collecting metrics about the software performance at run time. Its adoption has two main steps: (i) instrumenting your component to capture the relevant metrics and (ii) configuring the main Prometheus server to extract the exposed metrics periodically;
2. *Grafana*: a solution for creating graphical dashboards combining multiple data sources. This last characteristic is essential for TeraFlowSDN because we need dashboards depicting data collected from the devices (therefore coming from the database used by the Monitoring component) and data related to the performance of TeraFlowSDN itself (i.e., using the information coming from Prometheus).

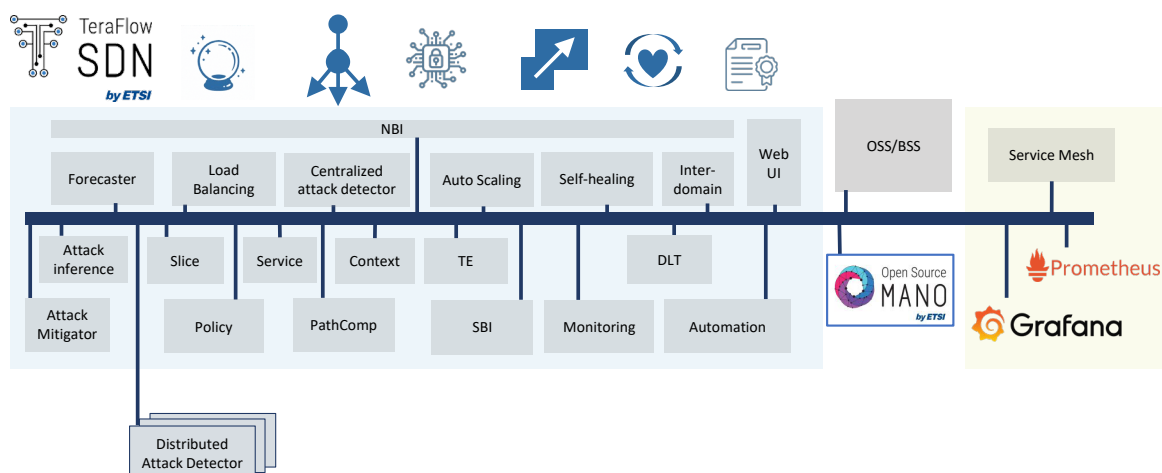


Figure 10. TeraFlowSDN extended architecture encompassing the metrics collection framework

In addition to these two main pieces of software, we rely on a service mesh software capable of performing load balancing for gRPC requests. Among the alternatives, Istio and Linkerd are regarded as the two most used service mesh implementations. Adopting Prometheus, Grafana, and a service mesh grants TeraFlowSDN a wide range of functionalities that can be used to understand the system's performance and identify potential bottlenecks or targets for optimization.

4.1. Micro-service gRPC Calls

TeraFlowSDN adopts gRPC as the standard protocol for internal communication among components. The adoption of gRPC is motivated by several factors: (i) the explicit definition of services and messages provided by the protobufs, (ii) the binary format that provides lower communication overhead, and (iii) the easy use and interoperability across programming languages. However, the gRPC protocol leverages HTTP/2 as the transfer protocol, i.e., gRPC is built on top of HTTP/2. Unlike HTTP/1.1, HTTP/2 allows for connection multiplexing, i.e., the same transport connection can be used for sending several application requests concurrently. While this feature is beneficial in terms of reducing signaling overhead (e.g., connection establishment time), it makes it harder to provide load balancing when gRPC is used in environments with multiple replicas of the same service, as it is the case of TeraFlowSDN. This happens because once a gRPC client establishes a connection with a gRPC server, the tendency is that the same connection will continue to be used as long as the client (i.e., the client object) still exists, or it times out due to inactivity. This prevents the client from taking advantage of any load balancing among existing replicas.

To solve this issue, TeraFlowSDN adopts a service mesh, a specific piece of software responsible for facilitating the load balancing among different gRPC server replicas. In addition to providing the basic functionality of gRPC load balancing, most service mesh implementations provide built-in monitoring for the health and detailed parameters of the connections among all components within a deployment. For instance, service mesh monitoring can measure how many requests per second a component/service or replica receives and the response time distribution for such calls.

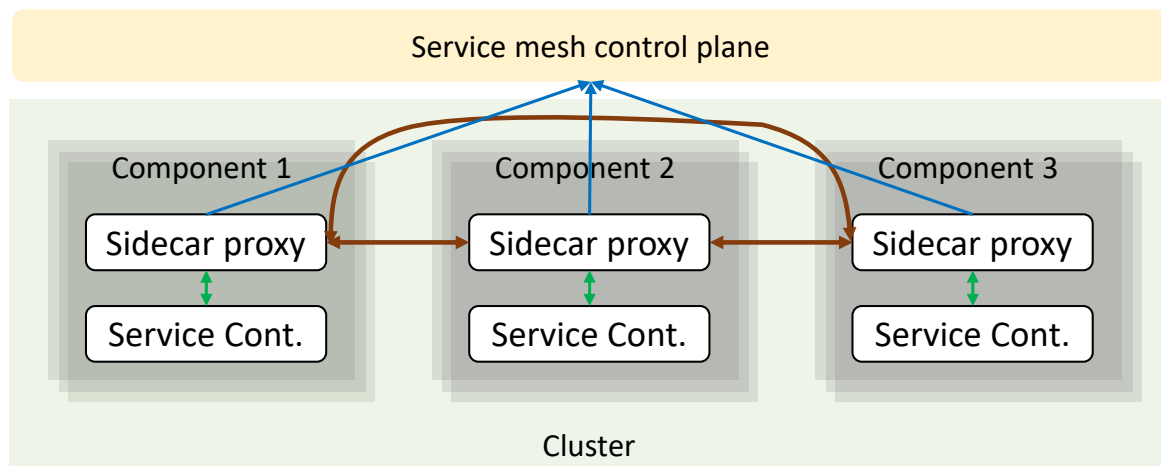


Figure 11. Architecture of the service mesh with sidecar proxy and service container

Figure 11 illustrates the architecture of a service mesh deployment, specifically, the deployment of Scenario 3 in the optical layer (described in Sec. 7.5.2). Each component is configured to let the service mesh control plane act over it. The main actions that the service mesh control plane performs are to include a new container in the Pod called *sidecar proxy*. The sidecar proxy is responsible for intercepting any outgoing communication to other components and routing it through their respective sidecar proxies. The control plane is responsible for disseminating information about replicas to all the sidecar proxies. This way, the load balancing is not done in the transport layer (the default in Kubernetes) but rather in the application layer. When new replicas are added, sidecar proxies are included in the new Pod and start being part of the pool of replicas soon after. Potential candidates for deployment are the *Istio service mesh*, and the *Linkerd service mesh*. Both tools are free to use and open source. In addition, their core functionalities are pretty similar.

Figure 12 shows the Linkerd dashboard during the execution of a scenario 3 experiment with optical physical layer attacks. We can see that the service mesh measures the number of requests per second and statistics about the response time and success rate of requests. This dashboard can analyse component performance, and help identify bottlenecks and communication issues.

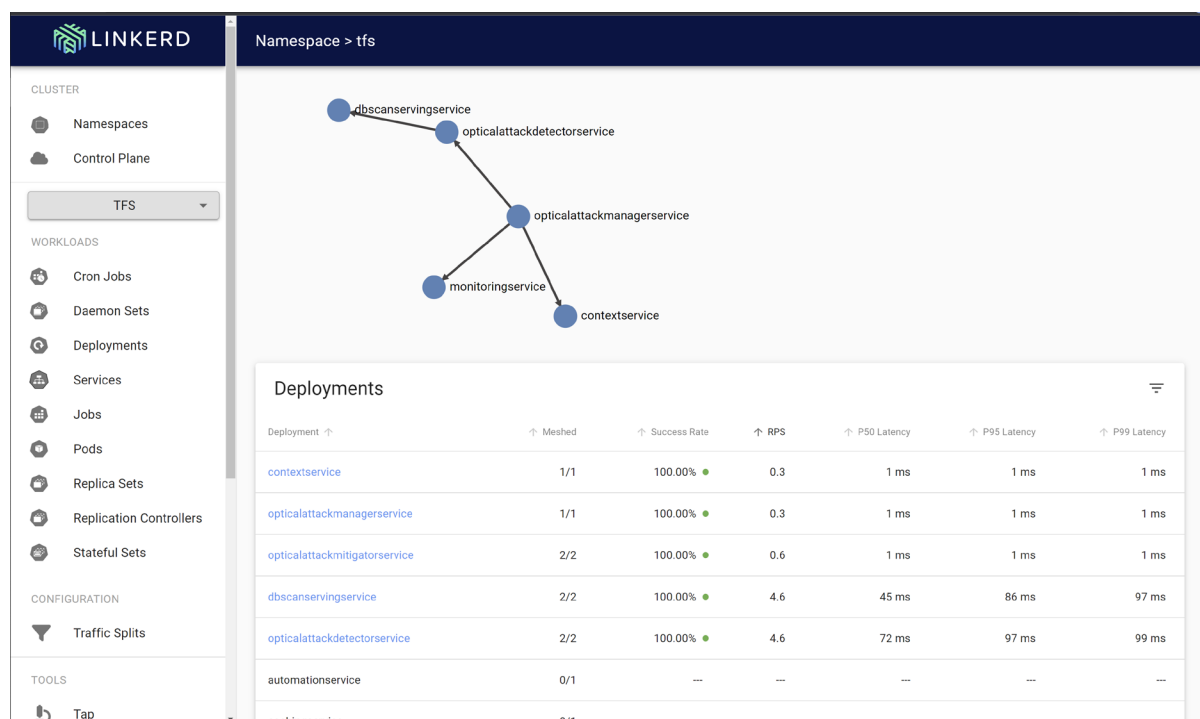


Figure 12. LINKERD dashboard during a Scenario 3 experiment

4.2. Prometheus

Prometheus is an open-source software widely used to implement monitoring of internal software performance. Prometheus is composed of two parts, the *metrics exporter* and the *server*. The metrics exporter is embedded into the code being monitored. This encompasses launching a web server that exposes the current state of the metrics upon request to a specific URL. The code in Figure 13 illustrates the response of a Prometheus exporter upon a request, specifically after instrumenting a TeraFlowSDN component (i.e., the optical attack detector) written in Python. There are two comments

before every value or set of values (i.e., lines starting with #). The two comments are called HELP and TYPE. The HELP line is used to export the metric description.

The TYPE describes which type is associated with the metric, out of the ones available in Prometheus. The types of metrics that can be stored in Prometheus are:

- *Counter*: numerical metrics that can only be incremented and are reset when the process restarts;
- *Gauge*: numerical metrics that can have their value set, incremented, or decremented;
- *Summary*: a vector that can be used to store observations and further processed to obtain averages and other statistics;
- *Histogram*: it 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.

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 580.0
python_gc_objects_collected_total{generation="1"} 315.0
python_gc_objects_collected_total{generation="2"} 8.0
# HELP process_start_time_seconds Start time of the process since unix
epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.67031966977e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent
in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 40.8
# HELP optical_security_loop_seconds Time taken by each security loop
# TYPE optical_security_loop_seconds histogram
optical_security_loop_seconds_bucket{le="1.0"} 22779.0
optical_security_loop_seconds_bucket{le="2.5"} 22779.0
optical_security_loop_seconds_bucket{le="5.0"} 22779.0
optical_security_loop_seconds_bucket{le="7.5"} 22779.0
optical_security_loop_seconds_bucket{le="10.0"} 22779.0
optical_security_loop_seconds_bucket{le="12.5"} 22779.0
optical_security_loop_seconds_bucket{le="15.0"} 22779.0
optical_security_loop_seconds_bucket{le="17.5"} 22779.0
optical_security_loop_seconds_bucket{le="20.0"} 22779.0
optical_security_loop_seconds_bucket{le="22.5"} 22779.0
optical_security_loop_seconds_bucket{le="25.0"} 22779.0
optical_security_loop_seconds_bucket{le="27.5"} 22779.0
optical_security_loop_seconds_bucket{le="30.0"} 22779.0
# HELP optical_security_loop_seconds_created Time taken by each
security loop
# TYPE optical_security_loop_seconds_created gauge
optical_security_loop_seconds_created 1.6703196713060427e+09
# HELP optical_security_active_services Active optical services
currently in the network
# TYPE optical_security_active_services gauge
optical_security_active_services 0.0
```

Figure 13. Example of Prometheus exported metrics

The second part of Prometheus is the server. The server has three primary responsibilities:

- *Metrics collector*: responsible for collecting all the metrics based on a list of places/components to be monitored;
- *Database*: The time-series database responsible for persisting/storing the collected metrics;
- *Web-based UI and API*: The user interface where users can query and visualize the data stored in the database. The API allows access to the same information without the GUI, which becomes ideal for extracting only the data.

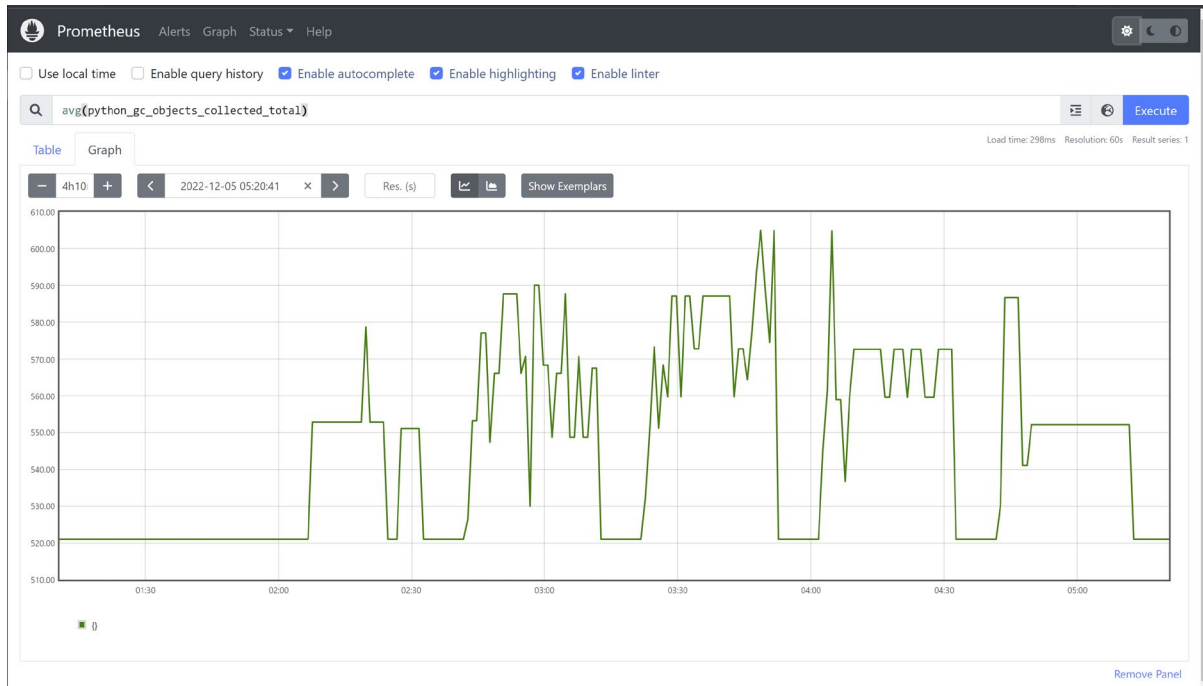


Figure 14. Screenshot of Prometheus WebUI with metrics collected from Python

Figure 14 shows a screenshot of the Prometheus WebUI. The plot shows the collected metrics from the Python processes during experiments of scenario 3 for optical physical layer attack detection. In particular, the plot shows the average number of objects collected by the garbage collector of Python. This detailed monitoring of the internal performance of the components enables TeraFlowSDN users (e.g., network operators) to obtain deep insight into potential bottlenecks that may arise, facilitating the analysis of such bottlenecks.

4.3. Grafana

Grafana is the final open-source software used in the metrics collection framework. Grafana is focused on visualizing different KPIs, allowing for their concurrent analysis. This is done through the creation of dashboards. A significant feature of Grafana is that it enables the creation of dashboards that combine data from different data sources. For instance, in the case of TeraFlowSDN, we have the Monitoring component responsible for monitoring services and devices currently active in the network. In addition, we also have Prometheus, where the internal monitoring data is stored. Therefore, Grafana makes it easy to generate dashboards combining data from the monitoring database and Prometheus.



Figure 15. The Grafana dashboard for the OFC'22 demonstration

Figure 15 shows a screenshot of the dashboard used for monitoring L3 services. This dashboard was created for the demonstration in [OFC22]. Filters in the top left corner also allow the user to select which devices, endpoints, and KPI types to show in the dashboard. For the final version of TeraFlowSDN, each scenario will provide a custom dashboard in Grafana where all the relevant KPIs can be analyzed near real-time.

4.4. Metric Definitions

This section provides a complete overview of the relevant metrics for the TeraFlow project. They have been expanded from D5.1. Table 1 contains the overview of the metrics. In each specific scenario section, the metrics are further detailed in the context of the scenario.

Table 1. Summary of metrics relevant for the TeraFlow project

Metric	Definition	Relevant scenarios
Device on-boarding time	Control plane latency to on-board a new device and upload its configuration. Does not consider the necessary time to communicate with the device.	1
Service setup delay	Required time to setup a new service, from control plane perspective only. Does not consider the necessary time to communicate with device.	1,2
Slice setup delay	Required time to setup a new slice, from control plane perspective only. Does not consider the necessary time to communicate with device.	1,2
Data rate	Amount of data transmitted during a specific time period over a network	1

Latency	Latency is the time it takes for a device to send one small 'echo' packet to the serving content server and the corresponding 'echo-reply' packet to return to the device. This time is also called the round-trip time. It has become common practice to use the terms synonymously.	1
Energy	Measuring the reduction in total average energy consumption and average resource utilization metrics	1,3
Economic	Cost reduction both in CAPEX (disaggregated networks) and OPEX (automation).	1, 2
Resource efficiency	Measurement of the resources needed to serve a given traffic request with and without using integrated resource orchestration.	1
Multi-tenancy	Stress the slice/service management system and measure allocated slices	2
Trust	Secured deployment of services through DLT	2
Privacy	Percentage of exposure of physical topological details	
DLT transaction delay	Measurement of the delay introduced by the usage of DLT instead of other inter-domain communication mechanisms	2
Positioning	Deployment of a position-based technique for all vehicles	2
Security	Attacks need to be detected with high accuracy to make sure they do not remain undetected or unaddressed in the network	3
Reliability	Measuring the performance of the model on detecting unseen adversarial attacks	3

5. Scenario 1: Autonomous Network Beyond 5G

This section presents the first scenario in TeraFlow. It focuses on the evolution of autonomous networks beyond 5G. It can be considered an operator-led scenario, as it focuses on the evolution of transport networks through the hierarchical integration of SDN and NFV technologies. Firstly, we introduce the scenario. Secondly, we present its alignment with TeraFlow architecture. Thirdly, we present the scenario setup in the laboratories. Fourthly, we present the relevant metrics and KPIs for the scenario. Fifthly, we introduce the designed workflow and the current deployments. Sixthly, a preliminary performance evaluation is presented when available. Finally, we provide a summary of pending work and the next steps.

5.1. Scenario Introduction

Scenario 1 has the motivation that with 5G networks comes the opportunity to deploy new services in an automated manner. In this sense, 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 network slice selects its deployment type from a list of predefined specifications, defining specific network resources and having requirements or constraints.

It has become clearer during the duration of TeraFlow that 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.

TeraFlowSDN controller supports a set of operator-driven use cases and workflows that include the objectives of this scenario dealing with the programmability of network elements and technology-based SDN controllers with the north bound and south bound interface requirements.

Figure 16 provides the high-level architecture of the envisioned scenario. A set of multiple integrated network elements are considered in network technological domains and used to support the autonomous provisioning and subsequent configuration and management of transport network slices, consisting of multiple Virtual Private Network (VPN) services such as Layer 2 (L2VPN) and Layer 3 (L3VPN) services with dedicated Service Level Agreements (SLA) (more details in D3.2). Another possibility is the interaction of an NFV Orchestrator (e.g., ETSI OpenSource MANO) with TeraFlowSDN North-Bound Interfaces (NBI), which includes provisioning L2/L3VPN connectivity. The TeraFlowSDN controller can trigger the necessary handlers to interact with the underlying technological domains in all these service requests.

The optical network domain can be managed using the Open Networking Foundation (ONF) Transport API (TAPI). The TAPI is used as an SBI towards an Optical Line System (OLS) or optical SDN controller,

which is responsible for optical network elements, such as optical transceivers, Reconfigurable Optical Add/Drop Multiplexers (ROADMs) or Optical Crossconnects (OXC).

The microwave transport network follows the ONF Technical Reference (TR) 352 and Internet Engineering Task Force (IETF) Network Topology data models. To this end, a dedicated microwave SDN controller is used, which can interact with TeraFlowSDN SBI based on ETSI mWT 024.

The Layer 3 (L3) routers can be controlled using OpenConfig data models. In this setting, the TeraFlowSDN Controller can be instantiated as a dedicated Internet Protocol (IP) SDN controller and can interact with a parent TeraFlowSDN controller instantiated as an End-to-End Orchestrator.

The L3 routers can also be controlled using Path Computation Element Protocol (PCEP). In this setting, the TeraFlowSDN Controller can be instantiated as a dedicated PCEP SDN controller and interact with a parent TeraFlowSDN controller instantiated as an End-to-End Orchestrator.

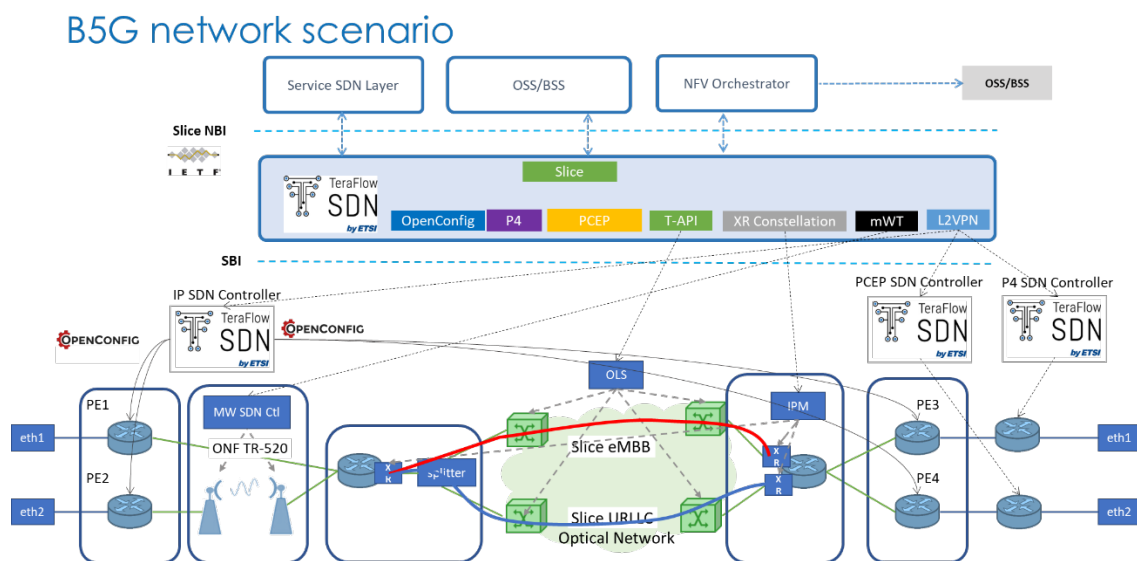


Figure 16. Scenario 1 high-level architecture

P4 switches also can be controlled using the TeraFlowSDN controller with a dedicated instance. In addition, this controller can interact with a parent TeraFlowSDN controller instantiated as an End-to-End Orchestrator.

5.2. Alignment with TeraFlow Architecture

Figure 17 shows the instantiation (configuration and TFS templates) for the End-to-End (E2E) TeraFlowSDN controller running as an SDN orchestrator. It may be observed that The ETSI OpenSourceMANO (OSM) NFV orchestrator is used to provision the network services and delegates to the TeraFlowSDN (TFS) controller, which is used as a Wide Area Network (WAN) Infrastructure Manager (WIM), the establishment of the inter-Data Centre (DC) connectivity through the WAN infrastructure. The OSM orchestrator uses the IETF L2VPN WIM connector to interact with the TFS controller.

This scenario involves the following components:

- NBI
- Forecaster

- Slice
- Service
- Context
- TE
- Path Computation
- Monitoring
- Automation
- SBI

The following use cases from D2.2 are of interest for testing the validity of these components and the overall scenario:

- Zero-touch device automation
- L3VPN Service Management
- Integration with ETSI OpenSource MANO
- Slice grouping
- Service restoration with P4 devices
- End-to-End Slice Provisioning with SLA

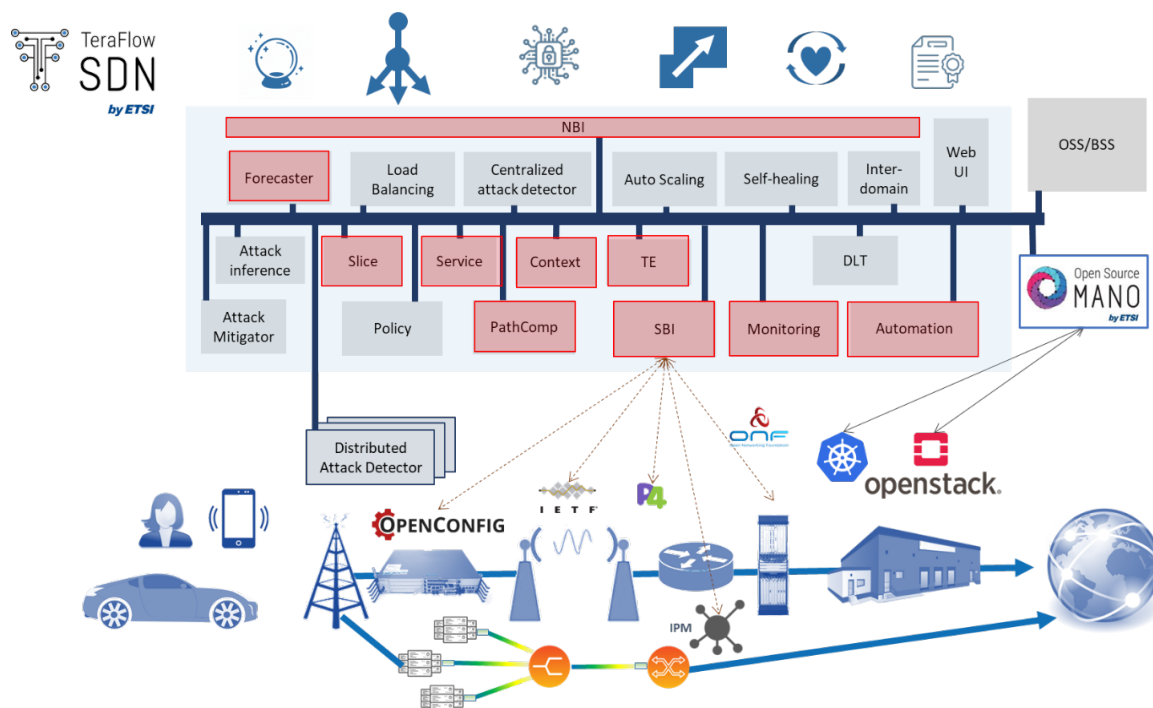


Figure 17. Scenario 1 E2E TeraFlow instantiation

5.3. Scenario Setup

This section briefly describes the scenario setup, including references to the lab equipment that supports the performance evaluation work. The multiple partners, facilities and network elements required in this scenario and use cases are described below. The different partner premises will be connected utilising secure VPN tunnels forming a distributed testbed where the use case on

Autonomous Network Beyond 5G will be assessed from the control plane perspective. Network elements residing in the same laboratory will also be interconnected through data plane connectivity, but verifying and assessing the TFS control plane performance is not required.

Telefónica contributes with their Future Network Lab, providing access to different IP routers and optical devices. IP routers include Infinera DXR-30, ADVA and IP-Infusion whiteboxes based on EdgeCore CSR310. For the optical network, equipment includes 3 FSP 3000 optical nodes and an SDN controller from ADVA.

The test environment is based on the iFusion Testbed deployed in the Telefonica CTIO lab in Sur 3 Building in Madrid. The iFusion Testbed replicates the IP/Multiprotocol Label Switching (MPLS) Network of a Telefonica Business Unit. The access and cell site gateways usually form regions concentrated in an aggregator router in flexible hierarchy levels depending on the topology, as shown in Figure 18. The naming convention starts in HL5 for a Cell Site, continuing to HL4 for Access Routers working as intermediate hubs towards the Aggregator Provider Edge (PE) referred to as HL3. The HL2 hierarchical level comprises only PE routers transmitting data between regions. The testbed has two zones, and a backbone interconnecting them. The routers belonging to a region and the backbone act as Autonomous System Boundary Router (ASBR) routers. Thus, to forward the traffic from the L3VPN/L2VPN services, the ASBR routers from each region establish an external Border Gateway Protocol (eBGP) session against the core routers. Each region runs IS-IS as an IGP protocol and has a full mesh of BGP sessions between the nodes.

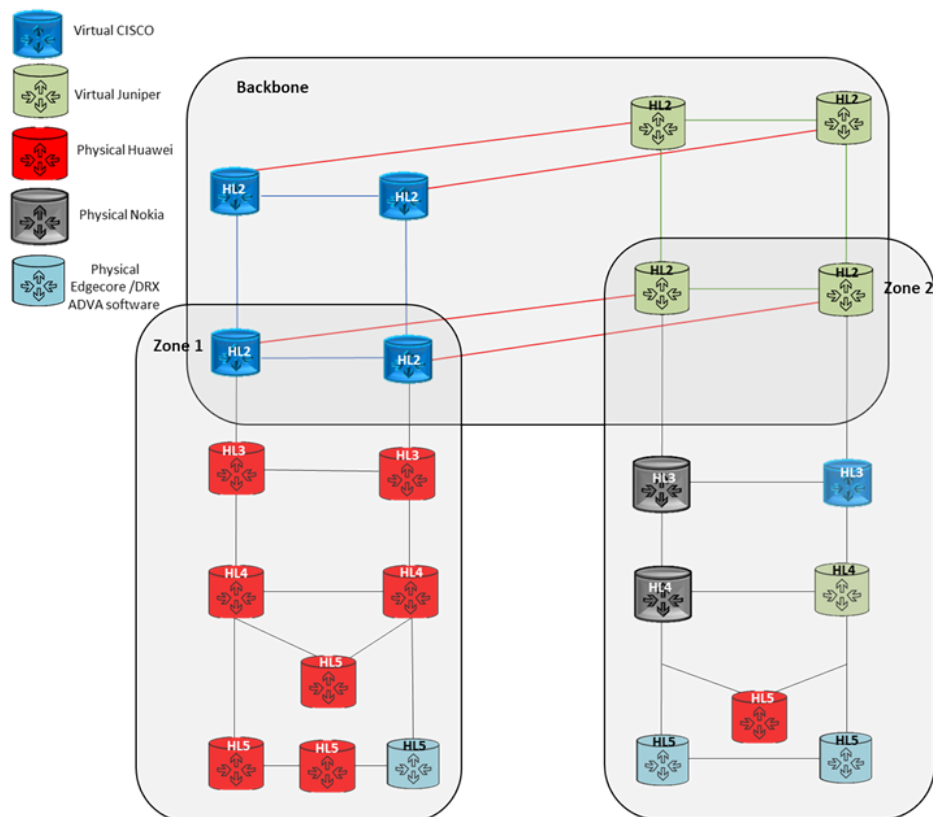


Figure 18. iFusion Testbed

The IP/MPLS testbed is connected to microstack servers where the services run directly or via a microwave link installed at Telefonica Lab. The microwave link is connected to an HL5 site, emulating the real scenarios in Telefonica Network, as illustrated in Figure 19.

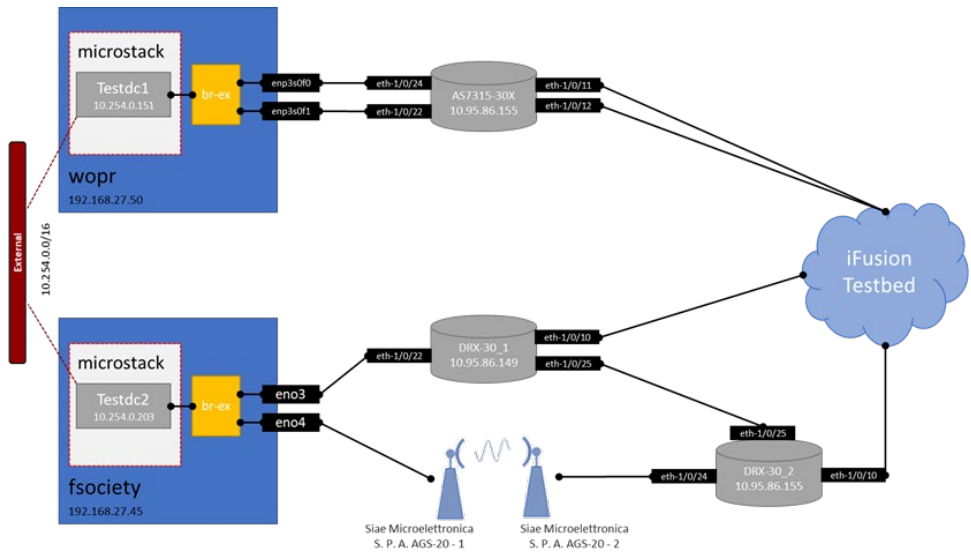


Figure 19. Openstack and IP router scenario interconnected through iFusion Testbed

The following paragraphs provide more insight into the tested platform.

- Hardware and Software used:

Role	Device	Version	Qty
HL5	Edgecore AS7315-30X	NOS-OPX-B-21.1.1 (8769)	1
HL5	Edgecore DRX-30	NOS-OPX-B-21.1.1 (8769)	2
Generator	Ubuntu virtual machine	20.04.TLS	2
Generator	SpirentTest Center	STC 5.20	1
MW	Siae AGS-2	003	2

- Cell Site Gateway Bare-Metal Hardware AS7315-30X.



Figure 20. AS7315-30X chassis layout

The Edgecore AS7315-30X in Figure 20 is an open cell site gateway platform that provides a combination of 1GE, 10 GE, 25 GE and 100GE interfaces utilizing merchant silicon and an x86 processor to optimize performance for mobile networks.

- Spirent SPT-N12U

The Spirent N12U Mainframe Chassis in Figure 21 provides test solutions for 800/400/200/100/50G, FlexE (Flex Ethernet) testing to address 5G transport, unified Layer 2 to Layer 7 traffic generation, investment protection with QSFP-DD, CFP8, and OSFP interfaces and wide-scale adoption by world's largest NEMs, Service Providers, and Enterprises.

This testing scenario has been used to verify the correct connectivity between the devices.



Figure 21. Spirent N12U chassis layout

A datasheet and additional information can be found at [SPI22].

- Edgecore DRX-30.

The DRX-30 devices (highlighted in Figure 22) are unbundled routers that combine a carrier-class white box portfolio with Infinera's scalable and proven CNOS software. For the equipment used in testing the scenario, the software has been modified by implementing a version of ADVA.

These devices are an open cell site gateway platform that combines 1GE, 10 GE, 25 GE and 100GE interfaces using commercial silicon to provide a cost-effective, software-centric, and flexible solution for network routing.

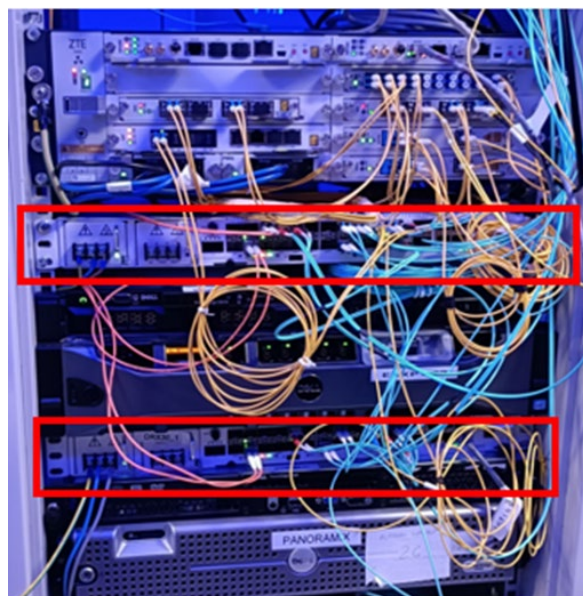


Figure 22. Edgecore DRX-30 chassis layout

- Virtualization servers:
 - Dell PowerEdge R730 (fsociety): It has 56 CPU intel Xeon E5-2690 v4 @ 2.60GHz cores capable to boost up to 3.50GHz, 125GB of DDR4 RAM and 2.7TB of hybrid storage (SSD and HDD). It runs Ubuntu 20.04.4 LTS and 10Gbps network interfaces. One illustration is provided in Figure 23.



Figure 23. Dell R730

- Dell PowerEdge R720xd (wopr): It has 32 CPU intel Xeon E5-2680 @ 2.70GHz cores capable to boost up to 3.50GHz, 125GB of DDR4 RAM and 100GB of SSD storage. It runs Ubuntu 20.04.5 LTS and 10Gbps network interfaces. One illustration is provided in Figure 24.



Figure 24. Dell R720xd

- Microwave radio equipment:
 - AGS-20: it is L2/L3 capable, compact, an indoor unit with up to 10GbE ports. It also integrates basic L3 networking, compatible with SNMP and NETCONF with YANG models (ONF and IETF). Both ends are linked using outdoor units with a 40dB attenuator to simulate the channel.
 - ASNK ODU (Figure 25): this takes the intermediate frequency of the AGS-20 and converts it to 23GHz. It is capable up to 4096 QAM modulation. These ODUs can transmit radio at 23dBm but we have configured them at 4dBm so as not to melt the receiver at the other end.

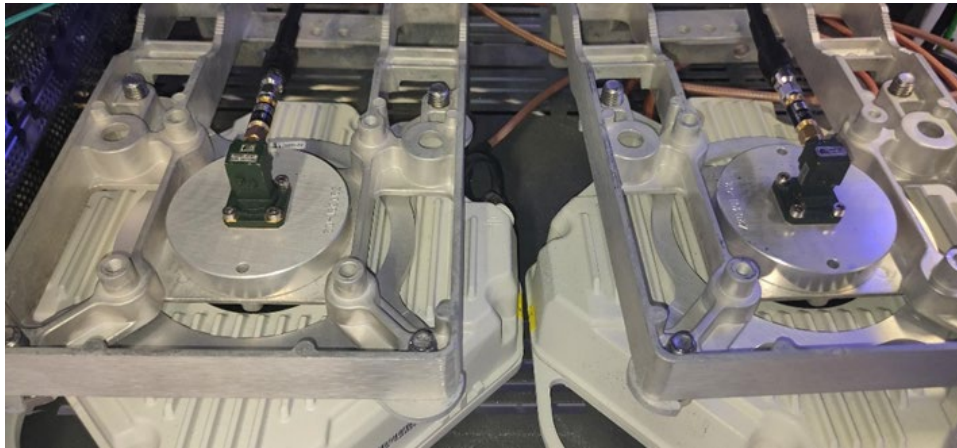


Figure 25. ASNK ODU radio link

SIAE contributes with their SDN controller and Microwave (MW) link equipment. The SDN controller will oversee the MW equipment through the ONF TR-352 Driver. It interacts with the E2E TeraFlowSDN controller using ETSI mWT 024 interface.

Infinera contributes with XR VTI mode Nx25G bandwidth allocation and verifies its interoperability with Intelligent Pluggables Manager (IPM). XR VTI mode optical transport port is presented as {device}/{port}.{vlan}. Other components on environment includes 400G SONiC device (Edgecore DSC240) hosting XR pluggables and XR-CA, 400G XR transceivers, optical splitter/decoupler and external traffic generator/analyzator.

Infinera integrates and ports XR-CA software components to standard whiteboxes DCS240 (and NOS SONiC) with management interface support. It has also enhanced SONiC CMIS support, and CLI commands for XR pluggable and has verified/hardened the requisites for SONiC features with XR-CA and non XR-CA use cases with DHCPv4/ND/NTP. The environment is used to verify XR pluggables on different operation modes and optical parameters with SONiC. The environment is multipurpose, allowing for example, XR module 4x100G breakout mode verification. Furthermore, different XR pluggable firmware versions are verified, providing feedback on XR transceivers interoperability in the open SONiC Environment.

IPM provides the necessary REST API to the TeraFlowSDN SBI driver to provide the necessary configuration parameters for XR constellation.

Ubitech contributes with a 32-port 400 GbE Intel Tofino-2 P4 switch acting as a high-performance network fabric. The device will be controlled through the P4 SDN Controller, also instantiated by Ubitech.

Stritzinger contributes with virtual routers based on Free Range Routing (FRR). They also provide an instantiation of the PCE-based SDN controller, which interacts with E2E TeraFlowSDN.

ADVA contributes the Ensemble Activator for whitebox devices in the Telefonica Future Lab and for Telenor, offering IP routing capabilities with OpenConfig APIs.

CTTC contributes with the ADRENALINE testbed®, providing an SDN/NFV packet/optical transport network and edge/core cloud infrastructure for 5G and Internet of Things (IoT) services. For this scenario, it includes an SDN-enabled disaggregated Optical Transport Network (OTN), consisting of a 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, controlled using a proprietary Open Line System (OLS) that acts as an optical SDN controller, offering an ONF Transport API.

CTTC also provides its Kubernetes-based infrastructure to execute the TeraFlowSDN Continuous Development/Continuous Integration environment.

5.4. Scenario Metrics

This section describes the scenario Key Performance Indicators (KPIs) to be reported as final achievements of the project. We have identified the necessary metrics and provided preliminary results in this document (D5.2). In deliverable D5.3, we will provide the complete measurements indicated in Table 2.

Table 2. KPIs and KVs for the Scenario 1

Name	Description	Relevance	Definition of measurement	Component
Device on-boarding time	< 50ms	Initial device Bootstrap in day 0 scenario.	Control plane latency to on-board a new device and upload its configuration. Does not consider the necessary time to communicate with the device.	Automation
Service setup delay	< 50ms	Very high. Necessary base time to deploy new services.	Required time to setup a new service, from control plane perspective only. Does not consider the necessary time to communicate with device.	Service
Slice setup delay	< 50ms	Very high. Necessary base time to deploy new slices.	Required time to setup a new slice, from control plane perspective only. Does not consider the necessary time to communicate with device.	Slice
Data rate	> 50%	Multi-layer optimization, introduction of optical layer closer to the edge and inclusion of novel optical technologies (SDM, FlexE, disaggregated flexigrid).	Percentage of increase of overall network resources when using multi-layer optimization in comparison with available network resources without network optimization.	Offline measurement based on the multi-layer topology

Latency	< 30%	Delay budget computation including traffic offloading mechanisms to optical layer.	Latency reduction percentatge when comparing multi-layer off-loading with and without.	Offline measurement based on the multi-layer topology
Energy	< 30%	Significant KPI	Reduction of energy consumption due to multi-layer optimization.	Path Computation
Economic	<20% cost	Significant relevance as is the main trigger for network upgrades.	Cost reduction both in CAPEX (disaggregated networks) and OPEX (automation).	Offline measurement based on the multi-layer topology
Resource efficiency	> 50%	Traffic optimization.	Measurement of the resources needed to serve a given traffic request with and without using integrated resource orchestration.	Offline measurement based on the multi-layer topology

5.5. Workflows and Current Deployment

Several workflows, including specific aspects of the proposed scenario, are presented in this section. These workflows and current deployments include: zero-touch device automation, L3VPN service management and integration with ETSI OpenSourceMANO, Slice grouping and End-to-End slice provisioning with SLA, Service restoration with P4 devices, and Energy-efficient Path Computation.

5.5.1. Zero-touch Device Automation

The automation component implements several Event-Condition-Action (ECA) loops defining the automation procedures in the network. These control loops deal with automation tasks such as bootstrapping new devices, configuring interfaces and forwarding tables, etc. They are triggered by relevant events (e.g., the addition of a device), when specific conditions are met (e.g., not configured), and they apply a set of actions (e.g., bootstrap the device) in response to these events.

The zero-touch device bootstrapping and monitoring workflow (Figure 26) is triggered by adding a new device in the TeraFlowSDN, for instance, through the WebUI.

The WebUI adds the new device through the SBI component, which triggers a connection to the physical device and the retrieval and storage of its current inventory and configuration in the Context database.

Such action triggers the distribution of a “Device Created”.

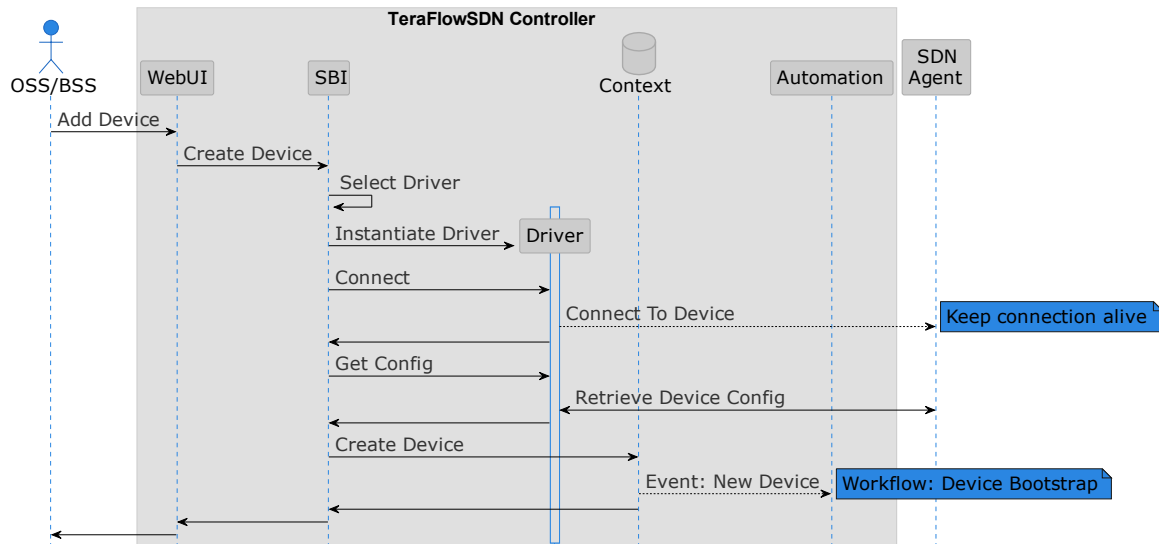


Figure 26. Scenario 1 workflow: Adding a device

When Automation receives the event, it retrieves the information and configuration of the new device from Context (see for reference Figure 27). If the device is not configured, it is bootstrapped by performing the following actions:

1. Retrieving from SBI the initial configuration template the driver defines for this device;
2. Populates the template with the appropriate values;
3. Configures the device through SBI, and;
4. Updates in Context database the configuration and new state of the device.

The update on the device triggers the distribution of a “Device Updated” event.

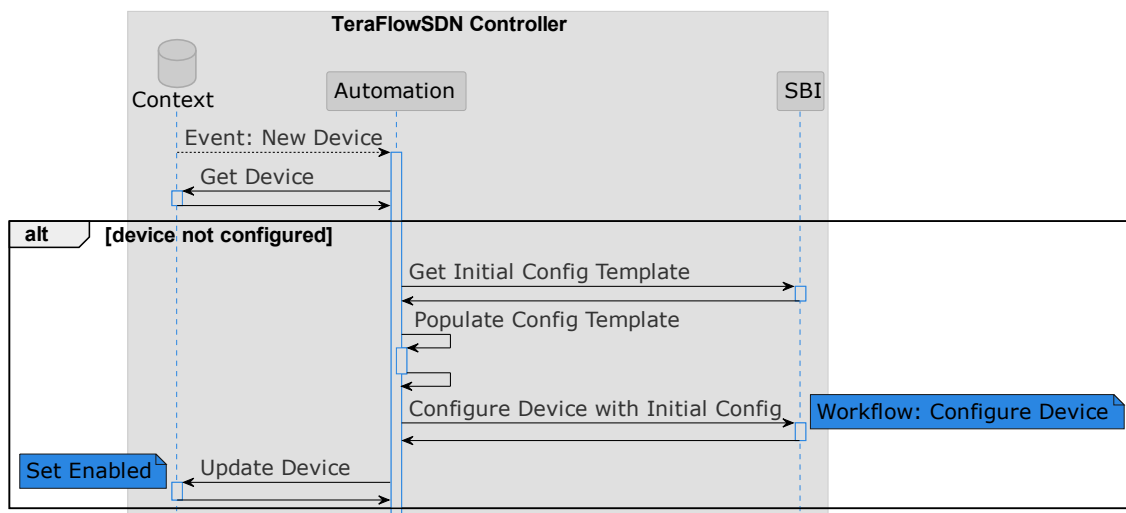


Figure 27. Scenario 1 workflow: Device bootstrap

When Monitoring receives this event, if the device is enabled but not being monitored, the former creates a set of KPIs for this device and starts monitoring them through the SBI component (Figure 28).

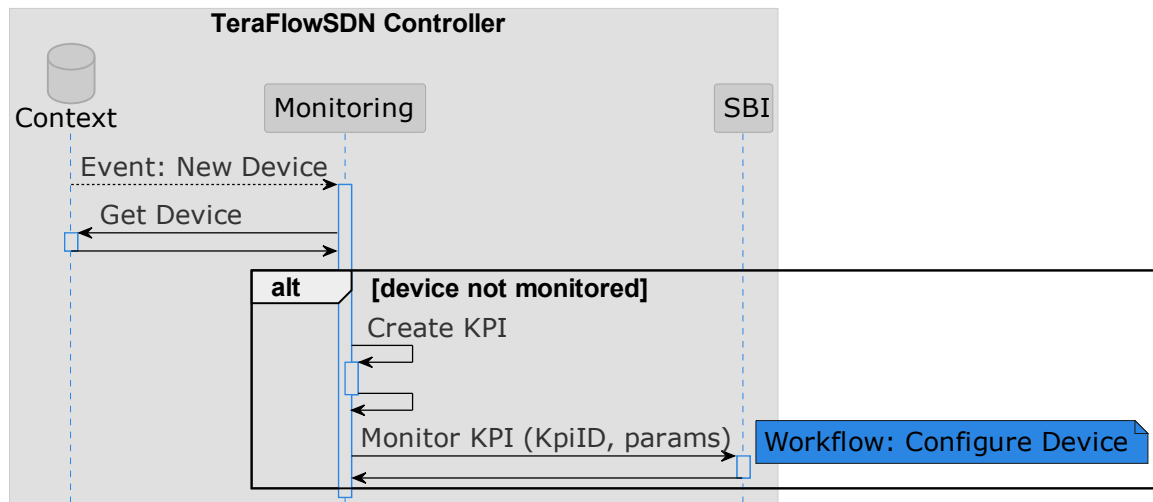


Figure 28. Scenario 1 workflow: Activate Device Monitoring

At this point, the samples coming periodically from the device are issued to the Monitoring component, which stores and makes them available for the other components (e.g., Grafana, Figure 29).

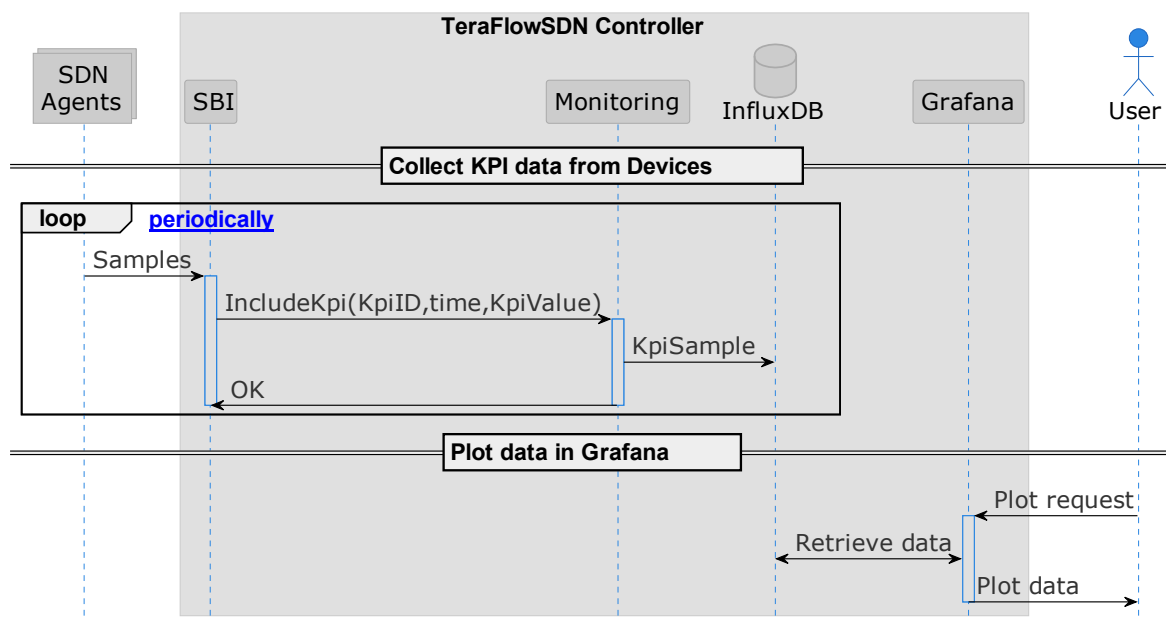


Figure 29. Scenario 1 workflow: Monitor Device Ports

5.5.2. L2/L3VPN Service Management and Integration with ETSI OpenSource MANO

The architecture used for this workflow is depicted in Figure 30. It shows two geographically-distant Data Centers (acting as Virtual Infrastructure Manager - VIM) that must be interconnected through a transport network slice. Each DC has network connectivity access through Customer Edge (CE) equipment connected to Provider Edge (PE) equipment, each located at a network operator's Point of Presence (PoP).

For instance, DC1 has its CE connected to data net, as well as DC2.

In this scenario, a transport network slice is deployed over a network connectivity service.

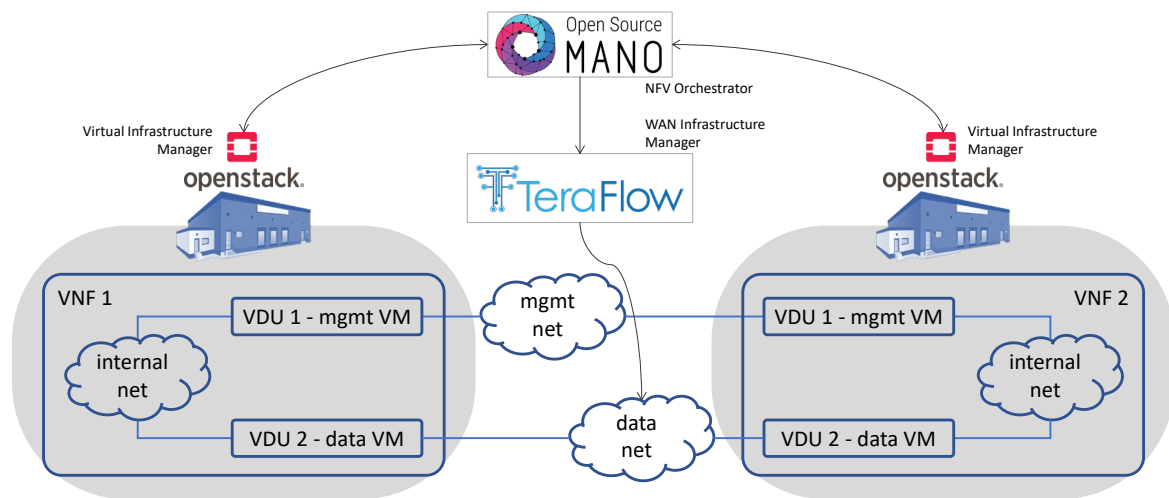


Figure 30. Integration of NFV-O and Transport SDN Controller

Figure 31 shows the complete provisioning of a Network Service (NS) through multiple VIMs. To this end, multiple VIMs are requested to deploy the allocated Virtual Network Functions (VNFs). Later, the point-to-point Service management workflow is triggered when OSM requests creating a new VPN service. Such a request has two phases. First, a new empty service is created to obtain a service identifier. Second, the endpoints are added to the service. When NBI receives the service creation request, it forwards the request to Service, which completes the missing required fields with default values, creates the service in the Context database, and returns the service identifier to OSM.

When NBI receives the request to add the endpoints to the service, it issues a service update request towards Service that identifies the devices owning the endpoints to be connected, identifies the device drivers they support, and chooses the appropriate service handler for the service.

This workflow first chooses and instantiates the Layer 3 Network Model (L3NM) service handler to configure an L3 VPN using Netconf/OpenConfig. Then it forwards the service request to that service handler. Next, the service handler creates the configuration rules for each involved device and configures them through SBI. Finally, it returns a confirmation to OSM.

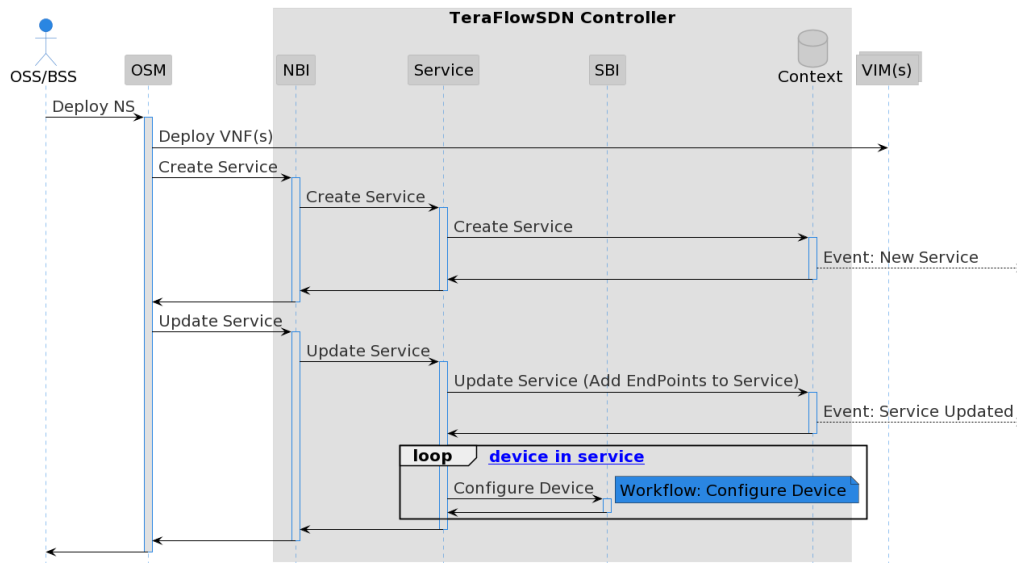


Figure 31. Scenario 1 workflow: NS Provisioning

The IETF L2VPN YANG data model for Service Delivery [RFC8466] enables to describe the transport network slices required by an OSS/BSS or an NFV orchestrator. An SDN controller can then consume the requests to provision the transport network connectivity services, as shown in Figure 32.

```

Hypertext Transfer Protocol
JavaScript Object Notation: application/json
{
  "ietf-l2vpn-svc:site-network-access": {
    0: {
      "connection": {
        "encapsulation-type": "dot1q-vlan-tagged"
      },
      "tagged-interface": {
        "dot1q-vlan-tagged": {
          "cvlan-id": 300
        }
      },
      "vpn-attachment": {
        "vpn-id": "b93d90c0-6b35-4bf5-8593-ac78f09f16a4"
        "site-role": "any-to-any-role"
        "network-access-id": "0a00b6a0-8911-4abd-9b89-f1a318d95456"
      },
      "bearer": {
        "bearer-reference": "CE1-PE1"
      },
      "availability": {
        "access-priority": 10
        "single-active": {
          0: null
        }
      },
      "access-diversity": {
        "constraints": {
          "constraint": {
            0: {
              "constraint-type": "end-to-end-diverse"
              "target": {
                "all-other-accesses": {
                  0: null
                }
              }
            }
          }
        }
      }
    }
  }
}

```

Figure 32. Example of ietf-l2vpn-svc:site-network-access

5.5.3. Slice Grouping and End to End Slice Provisioning with SLA

This workflow focuses on validating the proposed Network Slice grouping described in D3.2. As a reminder, we define a slice group as an entity consisting of one or multiple slices with a unique group identifier. One slice belongs to one and only one slice group. Slice grouping requires a mechanism to map a slice into its slice group, also known as a slice template or slice blueprint. From our transport network perspective, slice grouping can be based on mapping slice SLA requirements to the existing

set of slice groups. Thus, slice grouping introduces the need for a clustering algorithm to find service optimization while preserving the slice SLA.

Figure 33 shows the proposed architecture to evaluate Slice Grouping and End to End Slice Provisioning with SLA using hierarchical orchestration/control where the TeraFlowSDN orchestrator interacts and coordinates the underlying dedicated domain SDN controllers, namely: IP TeraFlowSDN controller, Microwave (MW) SDN controller, IPM, and Optical Lines System (OLS). Each domain SDN controller configures a particular network technology.

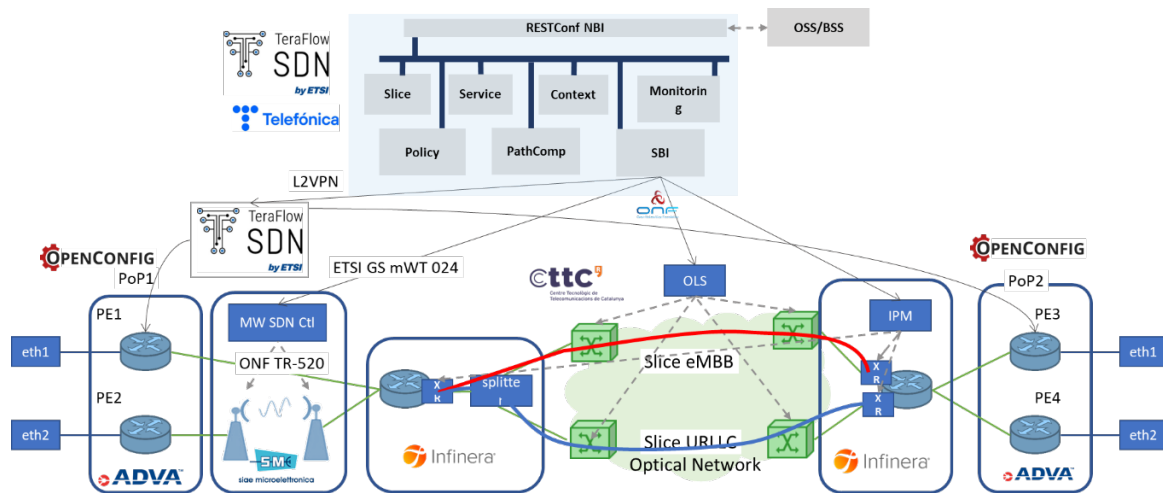


Figure 33. Transport Network Slice grouping on a hierarchical multi-layer SDN scenario

The workflow depicted in Figure 34 provides slice clustering based on slice requests that demand randomly distributed service availability and allocated bandwidth. Step 1 shows the request for the transport network slice, received from the NorthBound Interface (NBI) via a RESTconf interface. The request is then translated/mapped into the TeraFlowSDN protocol buffer and sent to the Slice component for processing (Step 2). Finally, in Step 3, the slice grouping algorithm is triggered, detailed below in Section 5.6.3. The outcome of the slice grouping algorithm can result in two options:

- i) the slice request is mapped to an existing slice group, or;
- ii) a new slice group might be required.

In the first case, the slice resources are related to a current/existing slice group. Then, using steps 8 and 9, Operation Support System (OSS) and Business Support System (BSS) are notified with the allocated resources.

In case new resources need to be allocated, the Slice component requests the necessary connectivity services to the Service component. The resources are then allocated following the necessary SDN orchestration mechanisms (steps 4-7). The underlying resource orchestration applied by TeraFlowSDN has been previously demonstrated, for IP over DWDM networks, as in the previous section.

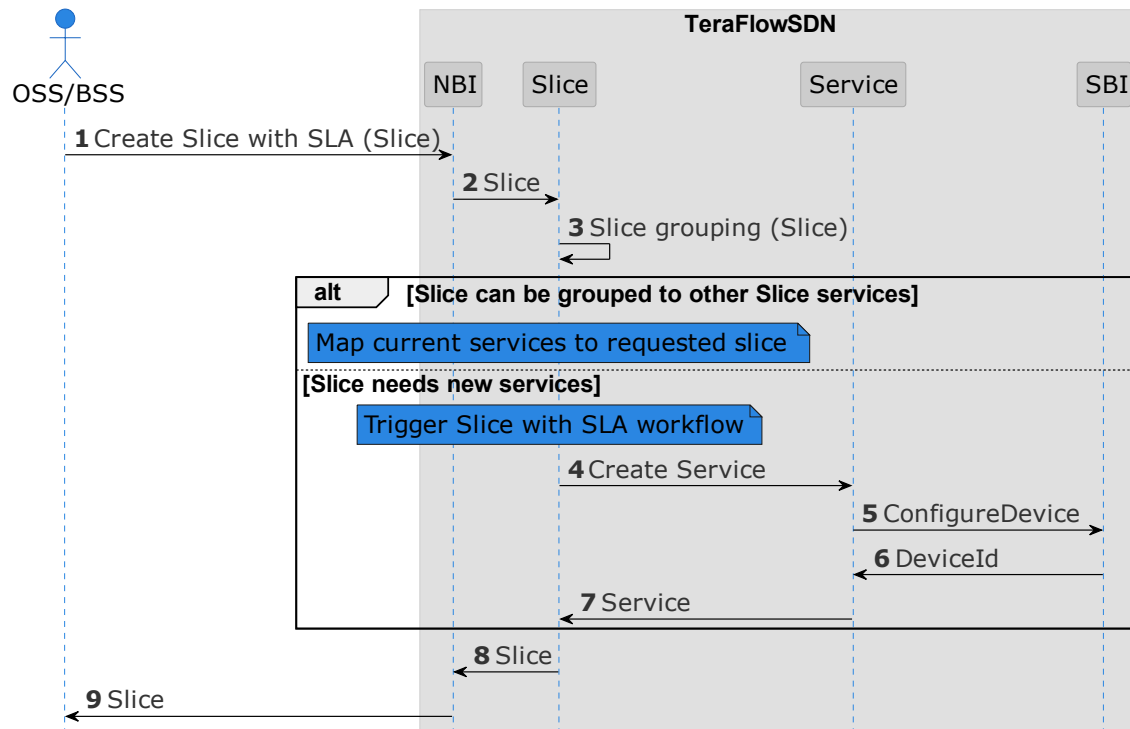


Figure 34. Slice grouping sequence diagram

5.5.4. Service Restoration with P4 Devices

This section presents the service restoration workflow assuming a given topology with several P4 devices. The key component to realize service restoration is the Policy component of the TeraFlowSDN controller, which offers an intuitive way to create event-driven SLAs by exploiting the alarm subsystem of the Monitoring component.

Device and link provisioning stage: The workflow shown in Figure 35 begins from an external OSS/BSS system, which uses the WebUI to add devices and links, thus establishing a topology of connected (P4) devices. These two initial parts of the workflow are highlighted as “Device provisioning” and “Link provisioning” in Figure 35.

Service creation stage: Then, the OSS/BSS requests the TeraFlowSDN NBI to create a new service on top of the established network, following the “Service creation” part of the workflow. The NBI passes the requests to the Service component, which creates a new service and stores it to the Context component. A relevant event is generated after the successful service creation, which notifies the WebUI and the OSS/BSS accordingly.

Policy creation stage: After the service is established, a new service-level policy can be applied to the network through the Policy component. This allows the OSS/BSS to map a service with an SLA. Specifically, the “policy creation” stage begins with a “policy add” call from the OSS/BSS to the Policy component, as shown in Figure 35. This call provides the Policy component with a Policy rule object which contains several internal objects denoting the service associated with the policy rule, a set of conditions for this rule to apply, and a set of actions to be enforced once the condition(s) is(are) met. First, the Policy component parses the received policy and validates that the policy rule object refers to a valid service ID and a valid set of KPIs and actions. Next, the input policy rule conditions are parsed

to identify which KPIs need to be requested from the Monitoring component. This entails the *(i)* creation of a KPI in case it does not already exist, *(ii)* monitoring of the KPI at the data plane, *(iii)* setting a KPI alarm which registers the Policy component to events when the KPI exceeds some range of values or specific threshold, and *(iv)* getting a stream of alarm responses when the KPI condition will be met. Once all these RPCs succeed, the policy rule transitions to the PROVISIONED state and the WebUI and OSS/BSS are notified via an event.

Potential SLA violation stage: At a later stage, an asynchronous event will be generated by the Monitoring component, when a KPI meets the associated policy condition(s). Upon receiving a KPI alarm, the policy rule transitions to the ACTIVE state, as the Policy component is ready to apply the corresponding policy action(s). Therefore, to avoid violating the SLA, the OSS/BSS should specify appropriate policy conditions which will trigger service restoration before the KPI reaches the critical threshold/range. This way, service restoration could be triggered just in time.

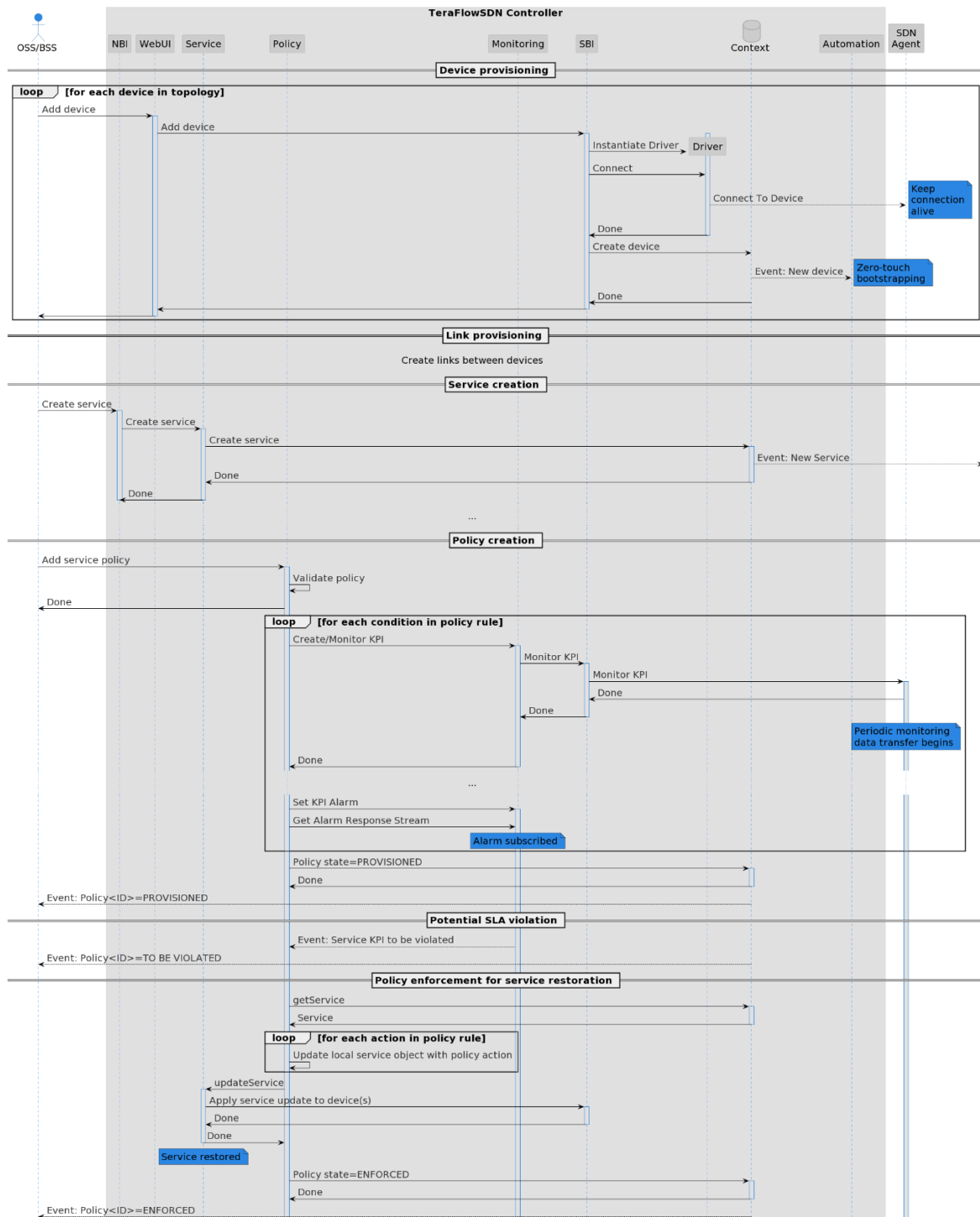


Figure 35. Policy-driven service restoration on a P4-based topology

Policy enforcement for service restoration stage: The last stage of the service restoration workflow is the one that applies the necessary policy action(s), when an alarm is received. First, the affected service is retrieved from the Context component, and its local configuration is updated by applying the list of policy actions. Then, to enforce the updates, the UpdateService RPC is called, which in turn results in Service component interactions, e.g., with (i) the Path Computation component (to compute a new path for the service) and (ii) the various underlying devices through the Device component. At

that point, the service is restored and the policy rule transitions to the ENFORCED state, which finally results in a relevant event being sent to the WebUI and the OSS/BSS.

More details about the Policy component are provided in D3.2, where the design, interfaces, operational workflows, and evaluation benchmarks of this component are introduced.

5.5.5. Energy-efficient Path Computation

One of the objectives of the TeraFlow project is to attain a reduction of the consumed energy when deploying network services involving both cloud and networking resources. In this context, as tackled in WP4 T4.3 activities, it is planned to exploit the PathComp component of the TeraFlow SDN controller to execute an energy-efficient routing algorithm. In summary, upon receiving a new network connectivity service request, the PathComp component is queried to devise a route and selected network resources, reducing the overall consumed network power. An energy-consumption model needs to be defined to accomplish this macroscopic objective, which is detailed in D4.2. The model considers that devices and links forming the underlying network can be in different operational states: active, asleep, or de-activated. The operational status affects the power consumed by both the device and the linecards (hosting the links/ports). An analytical model is presented in D4.2, describing the consumption of both devices and links according to their operational state. Then, for a given network connectivity requests, the algorithm at PathComp can compute the instantaneous network consumed energy depending on the traffic being transported over all active links and devices. The target is that the devised energy-aware routing policy reduces the overall network energy consumption while minimizing network degradation performance (e.g., network resource utilization, network service blocking, etc.). These metrics are intended to be assessed in a dynamic network service scenario and reported in D5.3.

Figure 36 depicts the basic workflow for processing any incoming network service request regardless of the adopted algorithm in the PathComp (whether energy-aware or not). Before triggering the path and resource selection algorithm, the PathComp needs to retrieve an updated view of the underlying transport network infrastructure, i.e., Context. To tackle the energy-efficient routing, specific device and link Context extensions need to be provided (please refer to D4.2 for the energy model description to complement the following information):

- Extended device attributes to i) determine the operational status; ii) specify the power *idle* in Watts. For the sake of clarification, the device power idle is the energy consumed regardless of data traffic being switched/transported over the device. This power/energy is mainly caused by interactions with the TeraFlow SDN controller, fans, etc.;
- Extended Link attributes to i) determine the operational status; ii) specify the energy/bit (i.e., J/bit). The links/ports are hosted in linecards equipped with memories for storing the packets, and a programmable look-up table (i.e., TCAM), etc. The consumption on the links/ports are proportional to the transported data rate.

Once the path (e.g., devices and links) is selected by the PathComp, this is passed to the Service Component to coordinate the programmability of the chosen network resources carried out by the Device/s components.

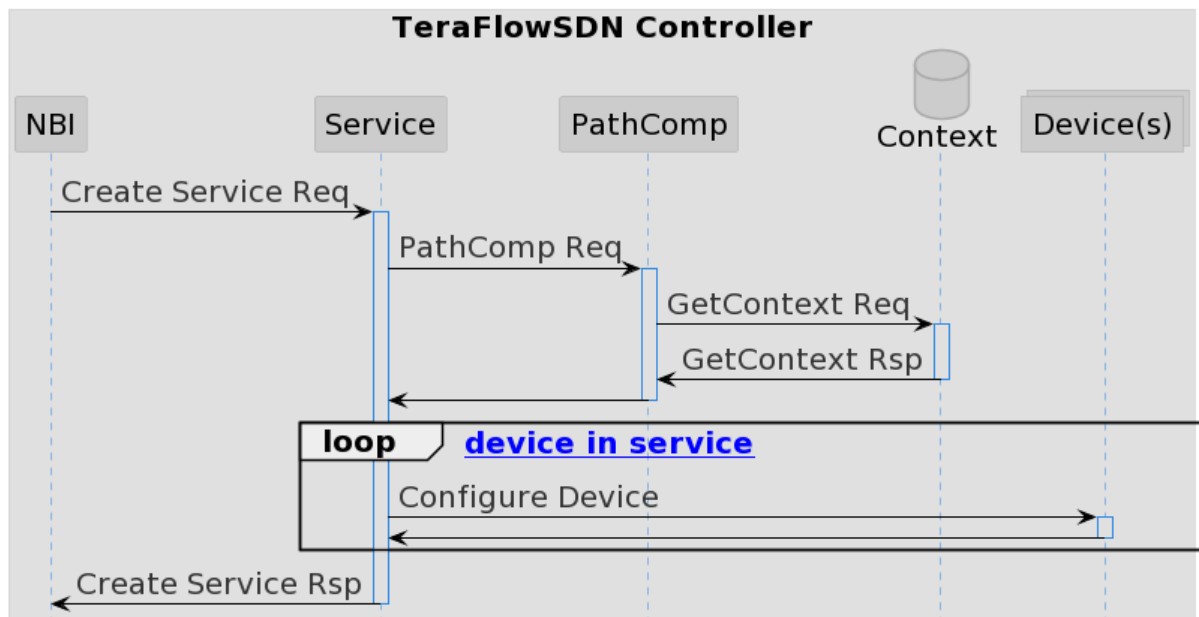


Figure 36. Basic network Service Creation relying on the PathComp component output: route and resource selection

Bearing the above in mind, the idea is to evaluate, over a single transport network domain, the performance of the proposed energy-aware routing with respect to a benchmark routing algorithm without energy-awareness. The network services will arrive and depart dynamically with heterogeneous network requirements in terms of bandwidth and maximum permitted latency.

Different traffic loads will be used to obtain a more complete performance comparison. This will be done considering the average network power consumption and average bandwidth blocked ratio metrics. Other metric could be potentially considered such as average PathComp execution time, or average amount of used devices and links.

5.6. Preliminary Performance Evaluation

In this section, we present some preliminary results for the proposed scenario. Results are provided in tables, plots, diagrams, and screenshots. The final evaluation of the scenario will be performed during 2023 and documented in D5.3.

5.6.1. Zero-touch Device Automation

This workflow has been demonstrated in [OFC22]. To this end, the device automation workflow is triggered when a new device is introduced to TeraFlowSDN to configure it. Figure 37 shows a screen capture of the received information after a device automated configuration.

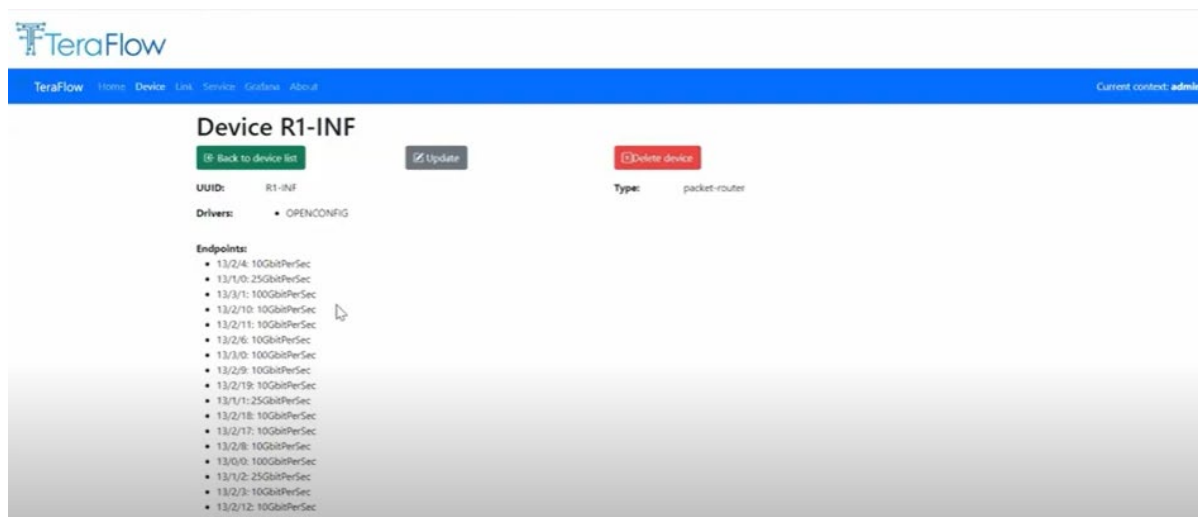


Figure 37 Screen capture of specific device information obtained after Device Automation

Name	Value	Comment
Device on-boarding time	-	This KPI will be analysed in D5.3

5.6.2. L3VPN Service Management and Integration with ETSI OpenSource MANO

This workflow has been demonstrated in [OFC22] and [ECOC22]. To this end, the experimental setup, illustrated in Figure 30, has been built on top of the CTTC's ADRENALINE Cloud Platform Testbed.

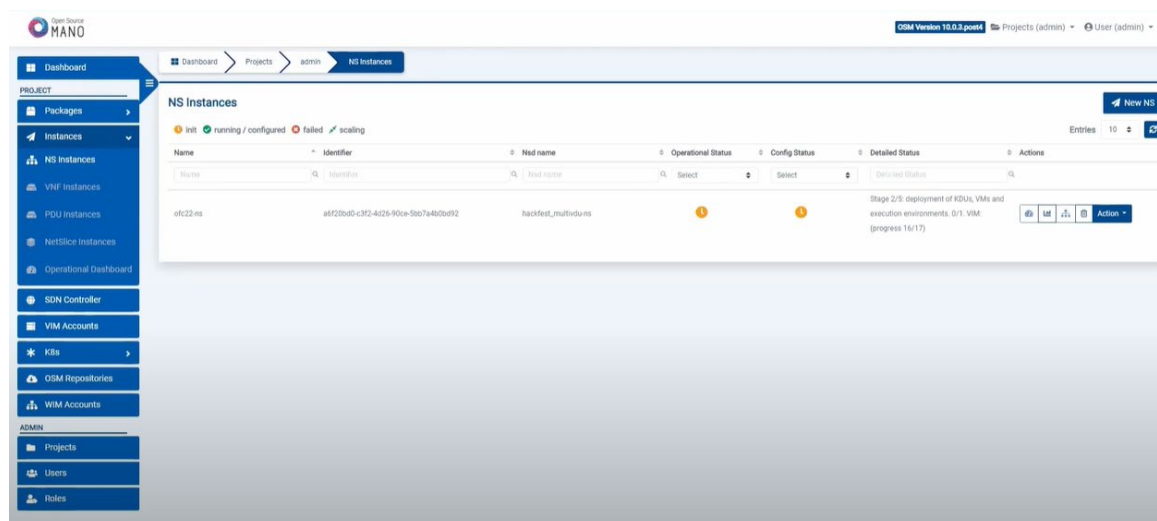


Figure 38. OSM screen capture with provisioned NS instance

The ETSI OpenSourceMANO (OSM) v10.0 (Figure 38) has been used as an NFV orchestrator to provision network services. At the same time, establishing the inter-DC connectivity through the WAN infrastructure is delegated to the TFS controller, which is used as a WAN Infrastructure Manager (WIM).

The OSM orchestrator uses the IETF L2VPN WIM connector to interact with the TFS controller.

This OSM WIM connector has been extended to support the request for disjoint paths. The proposed extensions for the IETF L2VPN connector have been contributed to the OSM source code and will be detailed in [D6.4]. Details of the extensions are provided in Figure 39.

On the TFS controller side, it uses its OpenConfig Device Driver to control the different IP routers and the TAPI Device Driver to control the optical core network, as previously demonstrated in [OFC22]. The VIMs are managed through OpenStack, controlled through the OpenStack REST-API by the OSM orchestrator.

1. VPN Service is created
2. Access sites are added, specifying:
 1. Associated VPN service identifier
 2. Encapsulation of L2 packets
 3. Bearer: identification of connection-point/link between DC and transport network.
 4. Availability: access priority and availability (single-active // all-active)
 5. Diversity: constraints between site accesses:
 - Constraints defined in the standard:
 - pe- / pop- / linecard- / bearer-diverse
 - New constraint defined:
 - end-to-end-diverse

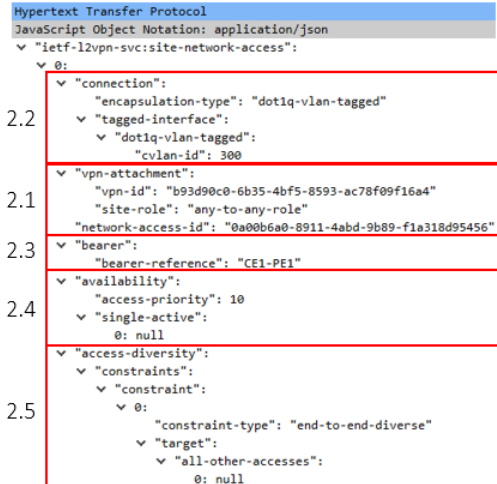


Figure 39. IETF L2VPN Extensions for end-to-end disjoint paths

The preliminary results of this scenario include a WireShark capture detailing the interactions between the OSM orchestrator and the TFS controller in Figure 40.

The interaction follows OpenConfig L2VPN provisioning as described in D3.2. It starts with a query of services available, followed by creating the VPN service handler. Then, each site network access is added to the VPN service handler, and finally, the status of the VPN service is verified.

Source	Destin	Protocol	Length	Info
OSM	TFS	HTTP	305	GET /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
TFS	OSM	HTTP/JSON	71	HTTP/1.0 200 OK
OSM	TFS	HTTP/JSON	226	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
TFS	OSM	HTTP/JSON	71	HTTP/1.0 201 CREATED
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC1/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC1/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC2/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP/JSON	643	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=DC2/site-network-accesses/
TFS	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
OSM	TFS	HTTP	355	GET /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services/vpn-service=b93...9f16a4/
TFS	OSM	HTTP/JSON	71	HTTP/1.0 200 OK

Figure 40. OSM-TFS Wireshark capture to deploy end-to-end network service

The live validation has showcased the entire provisioning and configuration procedure from OSM, the changes and operations performed by the TFS SDN controller, and the programming of the underlying network equipment in Telefónica premises. D5.3 will follow with the performance evaluation of the significant KPIs.

Name	Value	Comment
------	-------	---------

Service setup delay	-	This KPI will be analysed in D5.3
Data rate	-	This KPI will be analysed in D5.3
Latency	-	This KPI will be analysed in D5.3

5.6.3. Slice Grouping and End to End Slice Provisioning with SLA

The results of this proposed use case on slice grouping have been submitted and will be demonstrated at [OFC23], including hierarchical control of the underlying network technologies.

Figure 41 shows two network slice templates considered to allocate the requested transport network slices. The first one, referred to as gold, offers a service availability of 90% and a guaranteed bandwidth of 10 Gb/s. The second one, named platinum, provides a service availability of 99% with an allocated bandwidth of 100Gb/s.

```

{
  "id": "slice-template-gold",
  "service-slo-sle-policy":
  {
    "metric-bounds":
    {
      "metric-bound":
      [
        {
          "metric-type": "service-slo-one-way-bandwidth",
          "metric-unit": "mbps"
          "bound": "100"
        },
        {
          "metric-type": "service-slo-availability",
          "bound": "99.9%"
        }
      ]
    }
  }
}

{
  "id": "slice-template-platinum",
  "service-slo-sle-policy":
  {
    "metric-bounds":
    {
      "metric-bound":
      [
        {
          "metric-type": "service-slo-one-way-
bandwidth",
          "metric-unit": "mbps"
          "bound": "1000"
        },
        {
          "metric-type": "service-slo-availability",
          "bound": "99.999%"
        }
      ]
    }
  }
}

```

Figure 41. Example of slice templates

Figure 42 provides an example of a slice request. The requested slice includes a service-id along with a requested Service Level Objective (SLO) and Service Level Expectation (SLE) policy. By doing so, several metrics can be included, for example, SLO “one-way minimum guaranteed bandwidth” and SLO “guaranteed availability”. These are the two metrics considered in this work, but the network slice definition is flexible enough to support multiple SLO/SLE requirements.

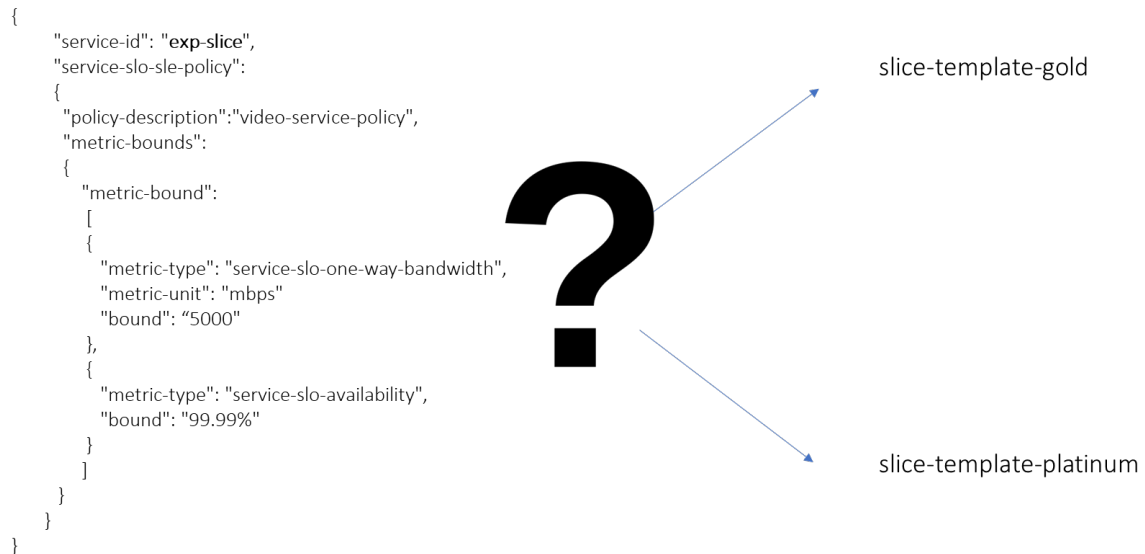


Figure 42. Applying slice grouping on new slice request depending on previously deployed slices

We use the K-Means clustering algorithm to support the slice grouping based on the requested SLO/SLE. This is an unsupervised machine learning algorithm, that groups data into a pre-determined (i.e., K) number of clusters. This number is defined by the user, and the K-Means algorithm groups the data into that specific number of clusters. This is the reason why a technique is needed to determine the optimal number of clusters for every specific case.

Figure 43 shows the application of the Elbow method to select the number of clusters on the received requests, on the x axis we have the selected number of cluster and on y axis we have the distance cost of the requests to the allocated clusters. We have run K-means algorithm for clustering the requests based on requested availability and bandwidth for a number of clusters (K value) ranging from 1 to 10. We have computed the sum of the squared distances from each point to its assigned center for each result. These plotted values allow us to determine the best value of K (i.e., 2 clusters in the proposed demonstration). The elbow method shows us that 2 is a possible good candidate for the number of clusters.

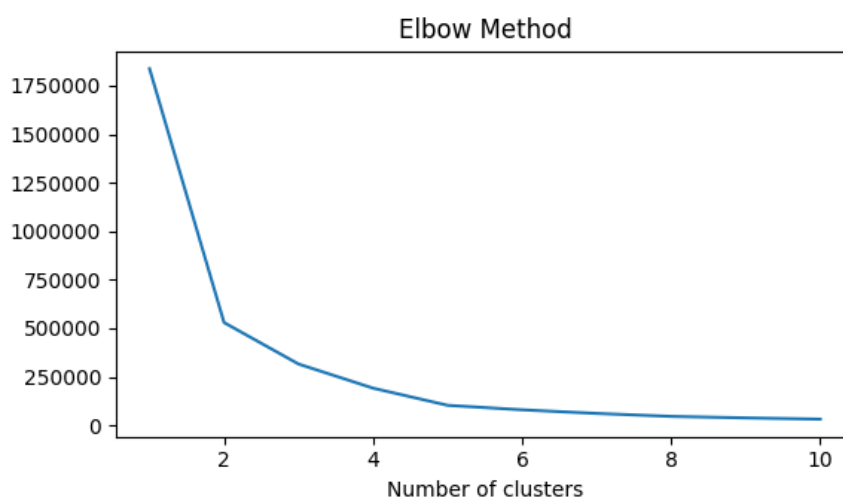


Figure 43. Elbow method applied to slice grouping

Finally, Figure 44 plots the received transport slice requests (each blue dot refers to a single request in terms of availability and bandwidth) and the clusters to which they are related (in red).

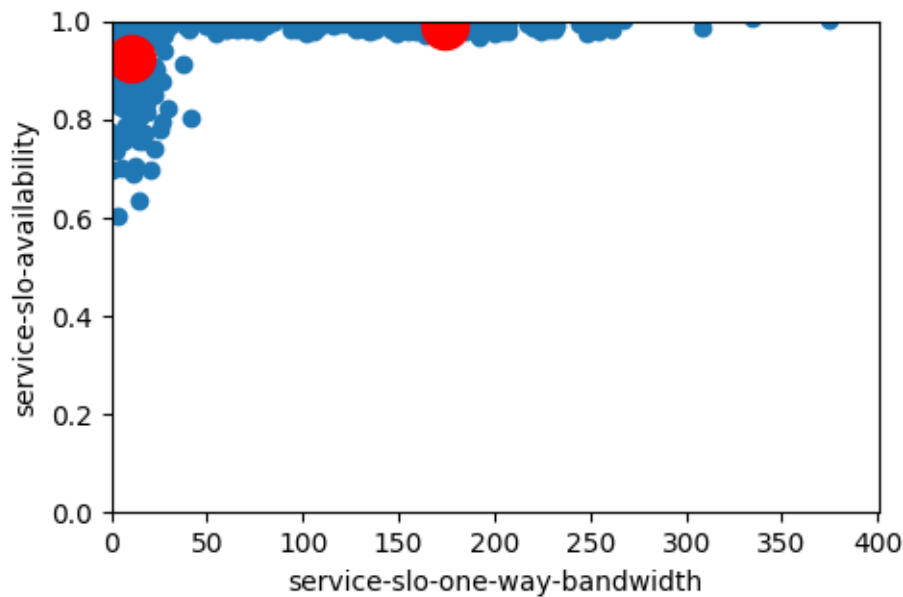


Figure 44. Allocated network slices and their slice groups

Now that the slice grouping approach has been validated, in D5.3 the necessary KPIs will be evaluated.

Name	Value	Comment
Slice setup delay	-	This KPI will be analysed in D5.3
Economic	-	This KPI will be analysed in D5.3
Resource efficiency	-	This KPI will be analysed in D5.3

5.6.4. Service Restoration with P4 devices

The results are not available yet. However, the plan is to present these measurements in D5.3.

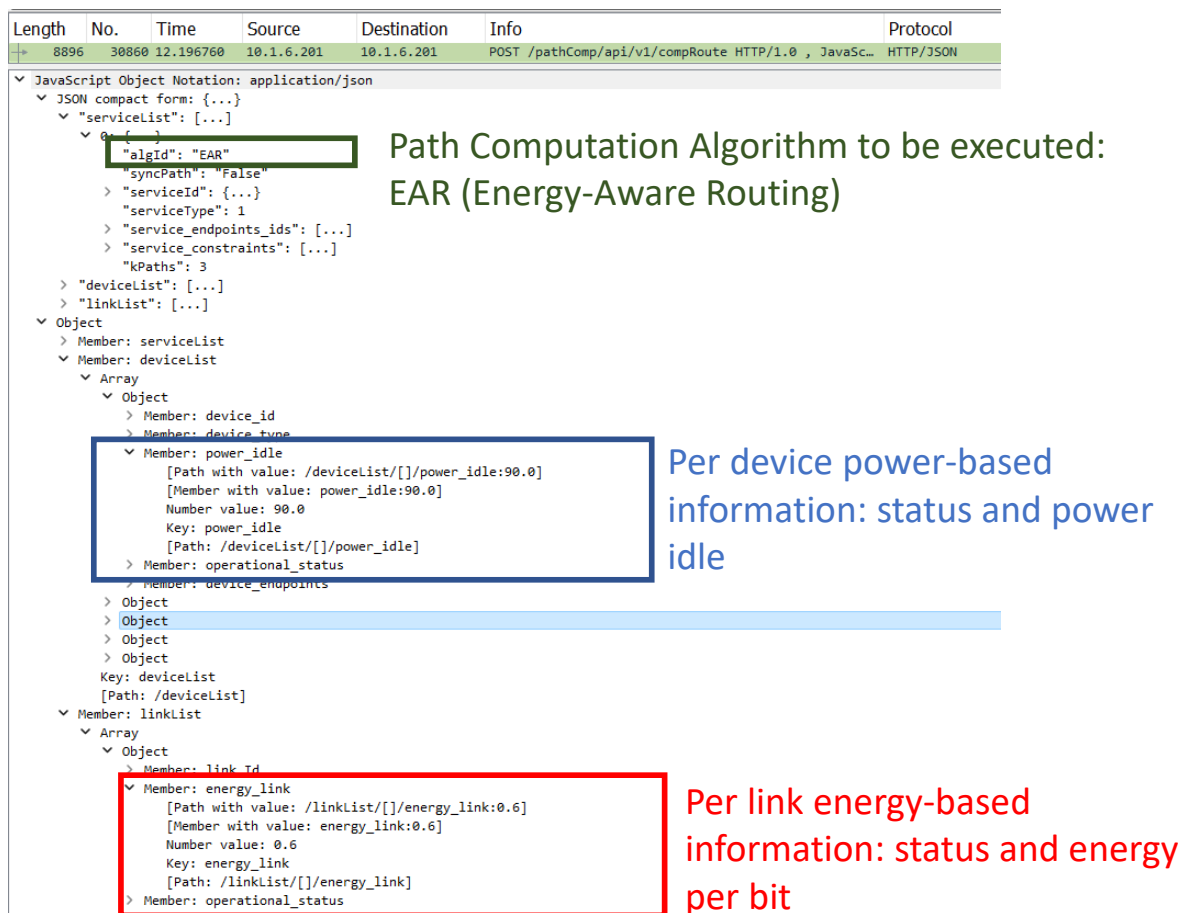
Name	Value	Comment
End-to-end service latency	5ms (indicative)	<p>The actual value depends on the topology setup. For example, hardware switches are faster than software switches, while software switches perform better on better Commercial off-the-Shelf (COTS) hardware. Therefore, this value may vary.</p> <p>The plan for this scenario is to use a software-based P4 topology atop Mininet, measure end-to-end service latency, and trigger service restoration using an appropriate threshold.</p>

5.6.5. Energy-Efficient Path Computation

The results are not yet available. The plan is to present these measurements in D5.3.

Name	Value	Comment
Energy	< 30%	The target is to attain a network power reduction around 30% upon adopting a energy-aware routing algorithm for serving the dynamic connectivity service requests. Different traffic loads and emulated network scenario will be considered for the final performance evaluation aiming at approaching a power reduction up to 30%.

As mentioned above in Section 5.5.5, the execution of the energy-aware routing at the PathComp requires that the Context device and link attributes are extended to provide specific power-based information (i.e., operational status, device power idle, link consumed energy/bit). Figure 45 shows the REST API POST message sent by the PathComp Front-End to the Back-End to request the execution of the Energy-Aware Routing (EAR) algorithm. The servicelist contents also carry the network endpoints, the constraints to be met, etc. The Context information is divided into the DeviceList and the linkList. In the former, each device is specified its operational status and the nominal value of the idle. For the links, it is described the operational status and the J/bit value for every individual link. Consequently, with this information, the PathComp component can trigger the EAR algorithm to accommodate a service while reducing network power consumption.



Path Computation Algorithm to be executed: EAR (Energy-Aware Routing)

Per device power-based information: status and power idle

Per link energy-based information: status and energy per bit

Figure 45. PathComp: REST API requesting a network service with energy-based Context Information

5.7. Pending Work and Summary

The Metrics Collection Framework has allowed us to start the recollection of KPIs, but until now, we are fixing several integration issues that preclude us from providing accurate results. To this end, we are fixing the reported issues, and the measurements will be provided in D5.3. Table 3 summarizes this status.

Table 3. Target and achieved KPIs and KVis for Scenario 1

KPI	Target	Validation results
Device on-boarding time	< 50ms	This KPI will be completed in D5.3.
Service setup delay	< 50ms	This KPI will be completed in D5.3.
Slice setup delay	< 50ms	This KPI will be completed in D5.3.
Data rate	> 50%	This KPI will be completed in D5.3.
Latency	< 30%	This KPI will be completed in D5.3.
Energy	< 30%	This KPI will be completed in D5.3.
Economic	< 20% cost	This KPI will be completed in D5.3.
Resource efficiency	> 50%	This KPI will be completed in D5.3.

6. Scenario 2: Inter-domain

This section presents the second scenario studied in TeraFlow. It focuses on the inter-domain deployment of transport network slices. It can be considered an operator-led scenario, as it focuses on the evolution of transport networks through peer orchestration of multiple domains. It also considers scalability and traceability. Firstly, we introduce the scenario. Secondly, we present its alignment with TeraFlow architecture. Thirdly, we present the scenario setup in the laboratories. Fourthly, we present the relevant metrics and KPIs for the scenario. Fifthly, we introduce the designed workflow and the current deployments. Finally, a preliminary performance evaluation is presented when available. Finally, pending work and summary is discussed.

6.1. Scenario Introduction

Several challenges need to be overcome when looking at the deployment of Cooperative, Connected and Automated Mobility (CCAM) services over a distributed edge and cloud infrastructure.

First, we need unified computing, storage, and networking resources management. In this respect, the TeraFlowSDN Controller, together with an NFV orchestrator (e.g., OSM), 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) concurrently.

Second, we must address multi-domain networking, where resources must be assigned in each domain and combined for an end-to-end service. In this respect, the TeraFlowSDN Controller will deploy several per-domain slice instances and compose them to create end-to-end transport network slices.

Finally, the different domains involved might belong to different network operators. This calls for methods enabling interdomain slicing between different operators while keeping internal network details private. In this respect, the TeraFlowSDN 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 TeraFlowSDN instances, if needed.

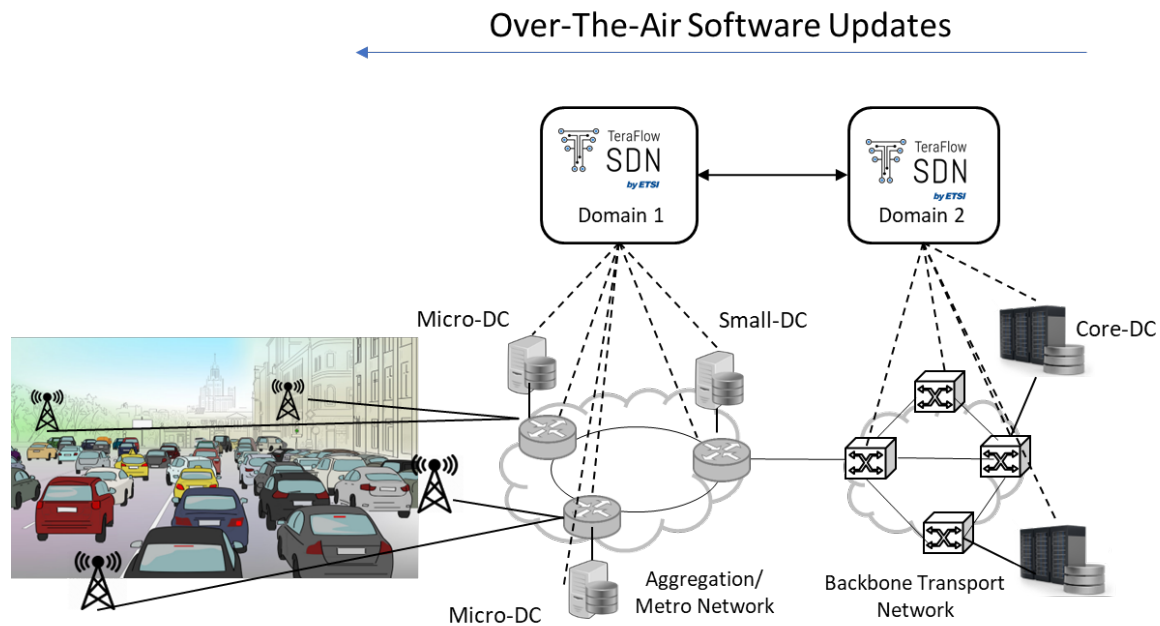


Figure 46. Scenario 2 high-level architecture

Figure 46 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 TeraFlowSDN Controller instance. In addition to selected uplink-heavy and latency-sensitive scenarios, the intention is to focus on Over-the-Air (OTA) software updates, which are software improvements that a car company sends wirelessly to vehicles. These OTA updates need to reach a moving target; thus, we provide an inter-domain scenario for moving connectivity services based on the position of the network elements. Testing and experimentation will be necessary to address the role of the Transport Network Slice and its endpoints regarding the interaction with adjacent access and service edge (SDN) control domains in this inter-domain scenario.

6.2. Alignment with TeraFlow Architecture

Figure 47 shows the single domain instantiation (configuration and TFS templates) of the TeraFlowSDN controller. It can be observed that interdomain connectivity will be provided either with DLT or inter-domain components, between multiple instances of the TFS controller.

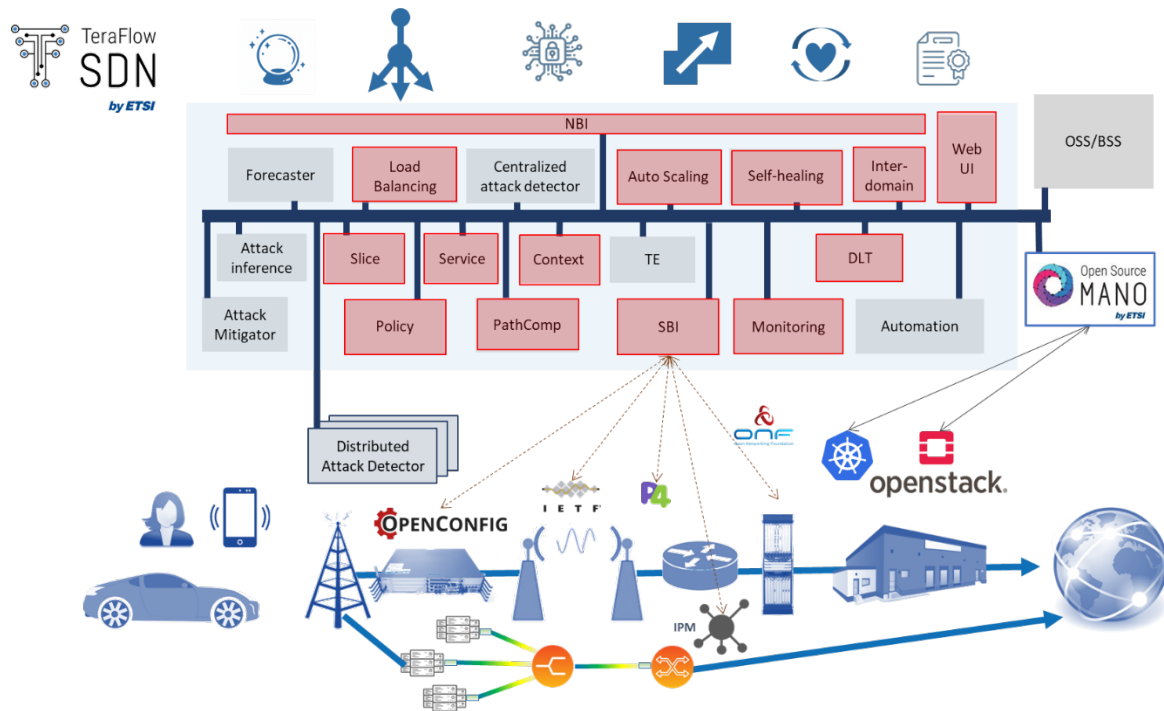


Figure 47. Scenario 2 TeraFlow instantiation in a single domain

This scenario involves the following components:

- NBI
- Load Balancing
- AutoScaling
- Self-healing
- Inter-domain
- Web UI
- Slice
- DLT
- Policy
- Monitoring
- Service
- Context
- Path Computation
- SBI

Use cases described in D2.2 of interest for testing the validity of these components and apps are:

- Operate TeraFlow at Scale
- Host tracking
- Flow Descriptors for IoT Services
- Using DLT for Inter-Domain Service Provisioning and SLA Violation Detection
- E2E Routing and SLA Violation Detection

6.3. Scenario Setup

The testbed envisioned to test the use cases belonging to this scenario involves the following partners and facilities:

- **CTTC** contributes with the ADRENALINE testbed® providing an SDN/NFV packet/optical transport network and edge/core cloud infrastructure for 5G and IoT services.

To validate this scenario, we will take advantage of the TAPI-enabled OLS controller and the underlying optical transport network infrastructure. Moreover, we have two whiteboxes cell-site gateways (CSGW) [EDG22] with IP Infusion OcNOS available and controlled using TeraFlowSDN (Figure 48).

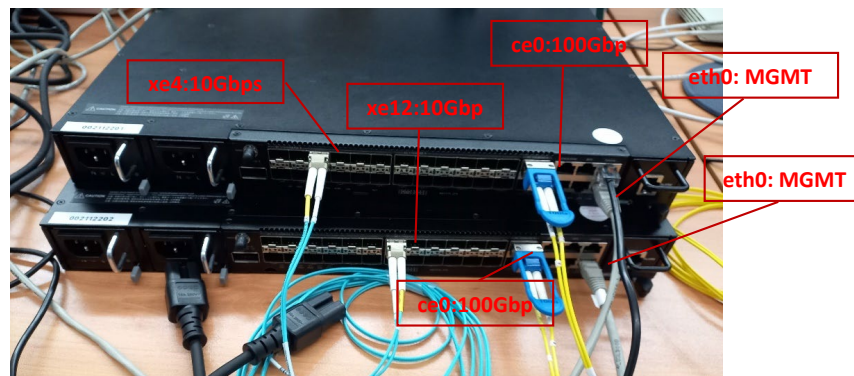


Figure 48 Interconnected CSGWs at CTTC Testbed

- **NEC** contributes with the blockchain infrastructure and runtime providing the means to interconnect different instances of the TeraFlowSDN for the different domains. More details are provided in D4.2 [D42].
- **Telenor** Telenor's testbed includes one server (HPE Proliant DL360 Gen10) and two Whitebox switches (Edge-Core AS7316-26XB), which are interconnected by the FS S5860-20SQ switch, as shown in Figure 49.

The physical server will deploy the TeraFlow SDN (TFS) and emulated domains/devices over Microk8s. In addition, TFS will be responsible for configuring the Whitebox switches, which runs the ADVA-NOS (Ensemble Activator).

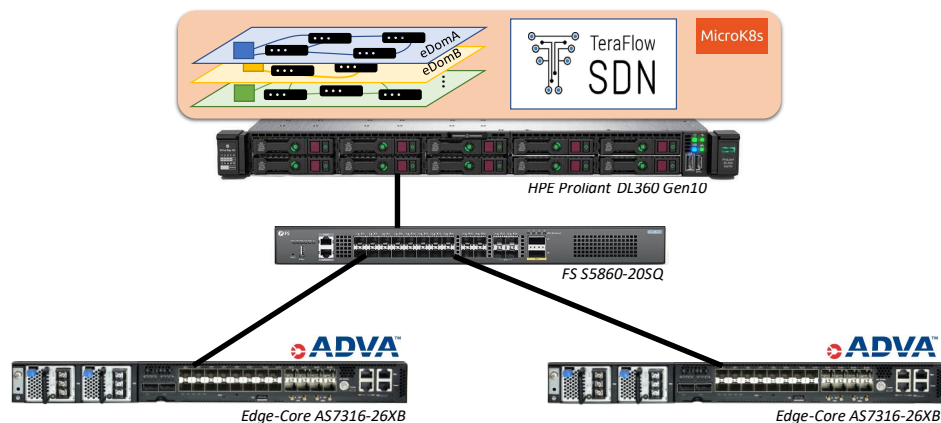


Figure 49. Telenor's testbed

The plan is to run experiments with one/multiple domains using emulated devices. Later, we hope to be able to configure the Whitebox switches and run experiments using physical devices.

- **ADVA** contributes the Ensemble Activator for whitebox devices in the Telefonica Future Lab and for Telenor, offering IP routing capabilities with OpenConfig APIs.

The different partner premises will be connected utilizing secure VPN tunnels forming a distributed testbed where the inter-domain scenario will be assessed. The setup will comprise two domains controlled by two different instances of the TeraFlowSDN Controller.

6.4. Scenario Metrics

This section describes significant metrics to be considered as scenario Key Performance Indicators (KPIs) to be reported as final achievements of the project. To this end, in D5.2, we have identified the necessary metrics and provided preliminary results when available. D5.3 will provide the complete measurements indicated in Table 4.

Table 4. KPIs and KVis for the Scenario 2

Name	Description	Relevance	Definition of measurement	Component
Service setup delay	< 50ms	Very high. Necessary base time to deploy new services.	Required time to setup a new service, from control plane perspective only. Does not consider the necessary time to communicate with device.	Service
Multi-tenancy	> 100 tenants	Scalability	Stress the slice/service management system and measure allocated slices	Slice/Service
Trust	100% Secured Conn.	Provide DLT for traceability	Secured deployment of services through DLT	DLT
Privacy	0%	Related to provide security and trust in multi-stakeholder scenario.	Percentage of exposure of physical topological details	Topology, DLT
DLT transaction delay	10s	Have a vision of which transactions can be recorded in DLT	Measurement of the delay introduced by the usage of DLT instead of other inter-domain communication mechanisms	DLT
Positioning	100% vehicles	Consider location in Service as a constraint	Deployment of a position-based technique for all vehicles	Service
Economic	< 20% cost	Reduction of OPEX costs	Reduction of Opex through automatic interconnection in	Off-line

			comparison with manual intervention.	
--	--	--	--------------------------------------	--

6.5. Workflows and Current Deployment

Several workflows, including specific aspects of the proposed scenario, are presented in this section. These workflows and current deployments include inter-domain provisioning using transport network slices, distributed ledger technologies, service/slice request scalability, and location-aware service updates.

6.5.1. Inter-domain Provisioning using Transport Network Slices with SLA

Figure 50 displays the sequence diagram regarding service preparation and activation. The workflow is initiated by a customer which can be any entity consuming TeraFlow services such as the OSS or other management domains including end-to-end service management. The slice component handles the customer's request, which forwards the end-to-end transport slice request to the inter-domain component. The inter-domain component, in turn, decomposes the end-to-end transport slice into per-domain sub-slices and requests their creation in the respective TeraFlow domains through the corresponding inter-domain components. This inter-domain communication is performed securely by mutual authentication before exchanging sub-slice requests.

If an appropriate sub-slice can be provided, the remote inter-domain component informs the requesting inter-domain component, and the latter can order the slice. Otherwise, a corresponding slice is created alongside its insertion into the catalogue and establishment of connectivity, triggering interaction with slice and service components, respectively.

Finally, the same sub-slice creation and connection establishment procedure is performed at the local TeraFlow domain (domain #1 in the figure).

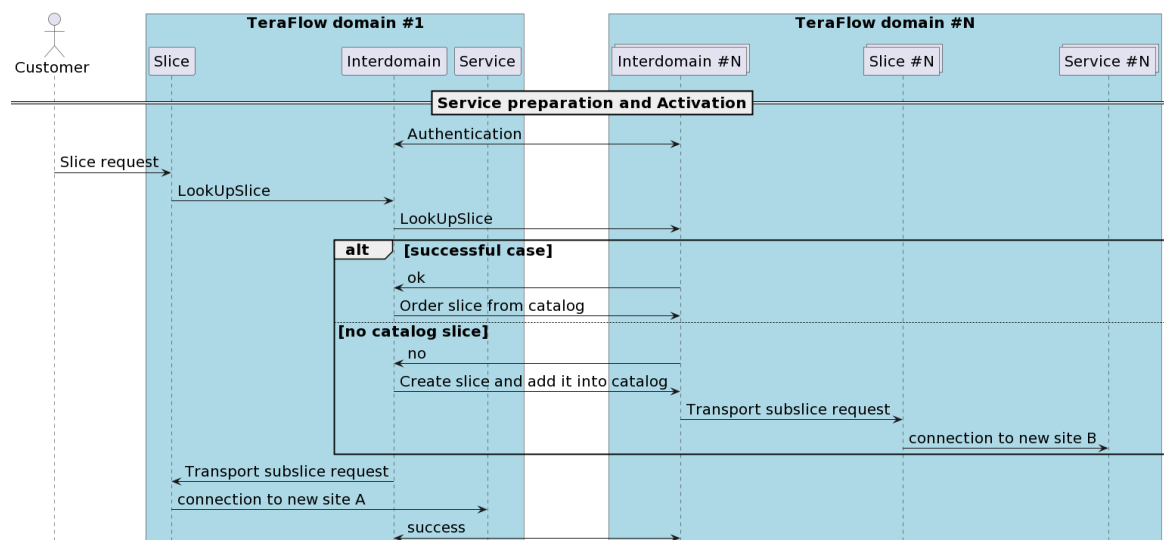


Figure 50. Scenario 2 workflow: Inter-domain E2E slice provisioning

6.5.2. Distributed Ledger Technologies

Figure 51 illustrates two architectures based on the options that Blockchain may offer using each transport domain SDN controller as the peer participating in the Blockchain network.

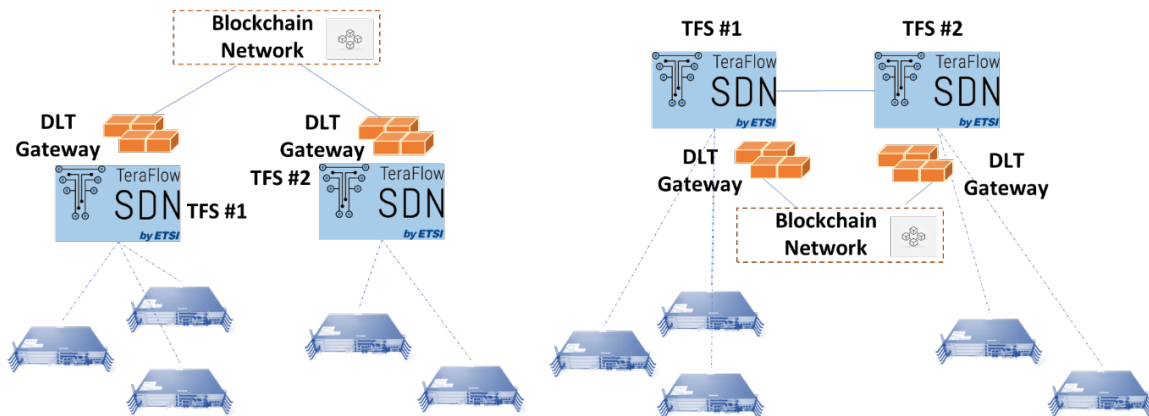


Figure 51. PDL proposed architectures, Full PDL (left); Complementary PDL (right)

The first model called "Full PDL", uses Blockchain as the key element in any interaction among the transport SDN controllers. In this model, Blockchain takes care of storing and distributing the information among the peers, moreover, the use of SCs may remove easy and repetitive tasks from the transport SDN controller solutions, making the Blockchain technology an even more integrated element within the inter-domain actions. In both proposed approaches, all the transport controllers are part of a Permissioned Distributed Ledger (PDL), as this avoids any non-desired entity may join the whole network and becoming a threat without the peer's knowledge. More information is provided in D4.2.

While having all the information within the Blockchain brings positive advantages in terms of security and immutability, it is not the best solution regarding latency. As presented in D4.1, this model needs to be carefully implemented, taking into account possible issues in terms of latency. This is because the validation and acceptance of new/updated information within the Blockchain may take a minimum of some seconds, which is a high delay compared to certain SDN actions that can be done in less than a second.

A second model called "Complementary PDL" was designed to solve the previous issue. In this case, Blockchain technology is used to store and distribute specific information samples, leaving the communication among peers to other existing communication protocols. In this model, Blockchain acts as a database for specific sets of information. Two possible use cases for this second model are:

- a) the topology export and;
- b) elements traceability.

In the first use case and as similarly done in D4.1, Blockchain is used only to store and distribute the static information related to the SDN topology using abstraction models. In the second use case, as there is no central authority on top of all domains, the use of Blockchain focuses on the immutability and transparency offered in order to check, if necessary, the owner of used resources or the responsibility of a committed element (i.e., Service Level Agreement).

Both models should not be considered as opposites but as two possibilities that may be implemented depending on the specific needs of a scenario.

For this reason, the transport SDN controllers should be adaptive enough to work with both models. In addition, TeraFlowSDN architecture based on micro-services allows the quick prototyping of the proposed use cases.

The main module that gives the adaptation capability to this SDN controller is the “DLT” component. Due to the cloud-native nature of the architecture, any module can interact with the others. So the two previous use cases (i.e., topology export and traceability) can be easily configured as the workflow to interact in the Blockchain is always the same (Figure 52). First, each DLT domain has to subscribe to the Peer (i.e., Blockchain system) to accomplish the "initialization" phase. Then, for each new data to record (i.e., Record{X}), the module owning the outcome (e.g., Inter-domain or Context) sends it to the DLT, which triggers the transaction with the "RecordtoDlt" and "DltRecordStatus" requests using the Peer. Then, the Peers (i.e., Blockchain) synchronize the data, and after it, an event is generated to distribute the record identifier (record_id) among all domains. Finally, the DLT of each domain obtains the Record{X} information and passes it to the corresponding module (Context or Inter-domain).

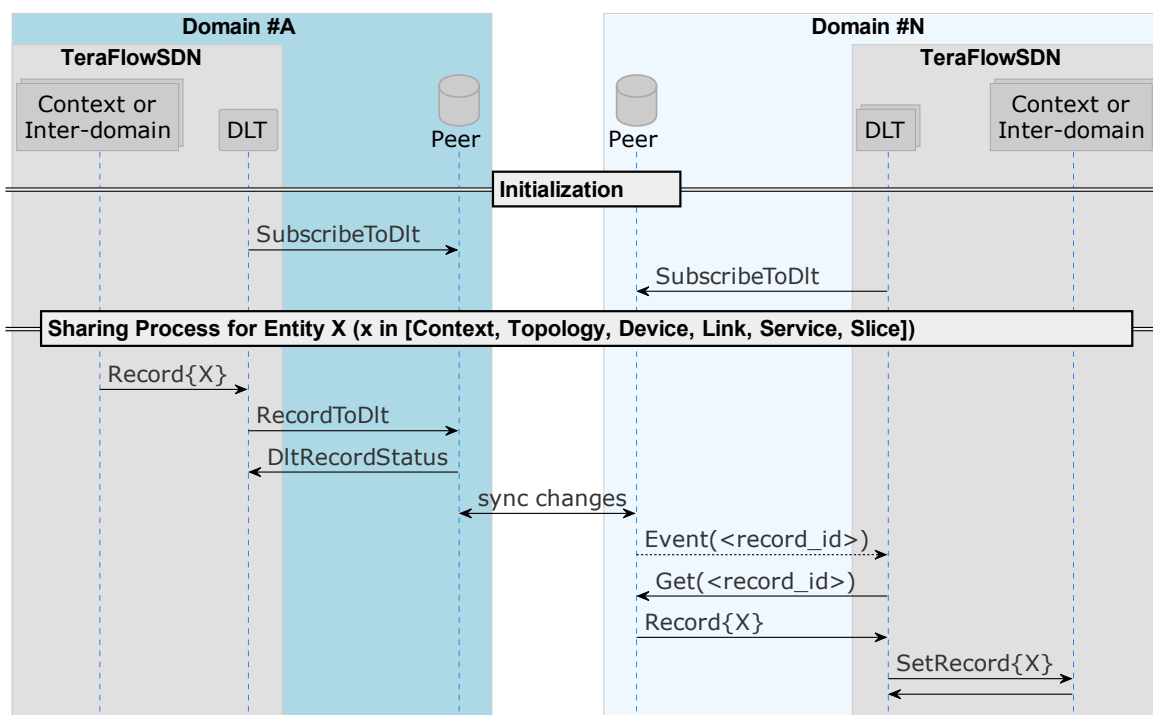


Figure 52. Scenario 2 workflow: Sequence diagram for DLT use

6.5.3. Service/Slice Request Scalability

This workflow shows the capability of TeraFlowSDN to handle a large amount of requests, verifying its scalability. To this end, we will perform a large number of requests, also considering a high load of requests per second in order to evaluate how well TeraFlowSDN performs. TeraFlowSDN uses load balancing and Horizontal Pod Autoscaler (HPA), as used in D3.2 to evaluate context, but to deploy multiple replicas of several components, such as Service, Slice and Context, in order to serve the requests. Scalability will be measured in terms of total number of requests handled, as well as demonstrated service and slice creation per second. Figure 53 provides the necessary workflow to evaluate Service scalability.

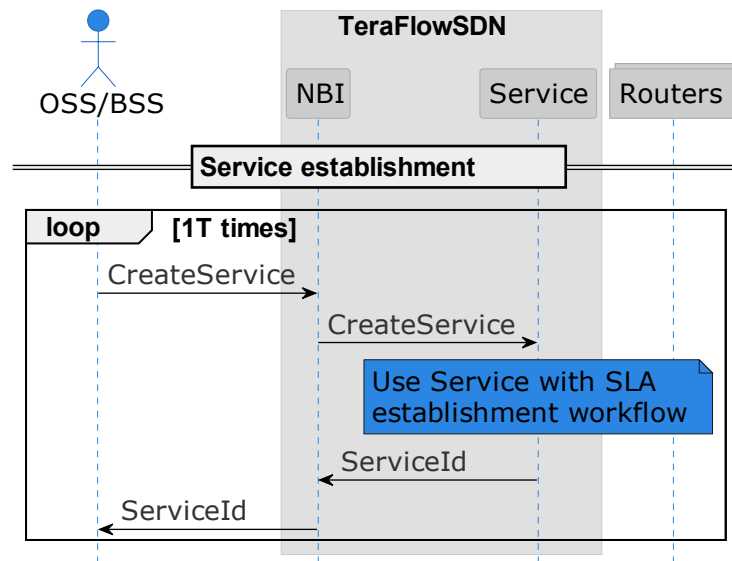


Figure 53. Scenario 2 workflow: Service Request Scalability

6.5.4. Location-aware Service Updates

The workflow in Figure 54 proposes the establishment of a connectivity service that includes information about its location, instead of the endpoints. Location and endpoints shall be matched at TeraFlow Service Handler in order to best provision the necessary endpoints depending on location. Once the service is provisioned, an update of the service is provided including new location. New endpoints shall be computed, and service updated following a break-before-make strategy.

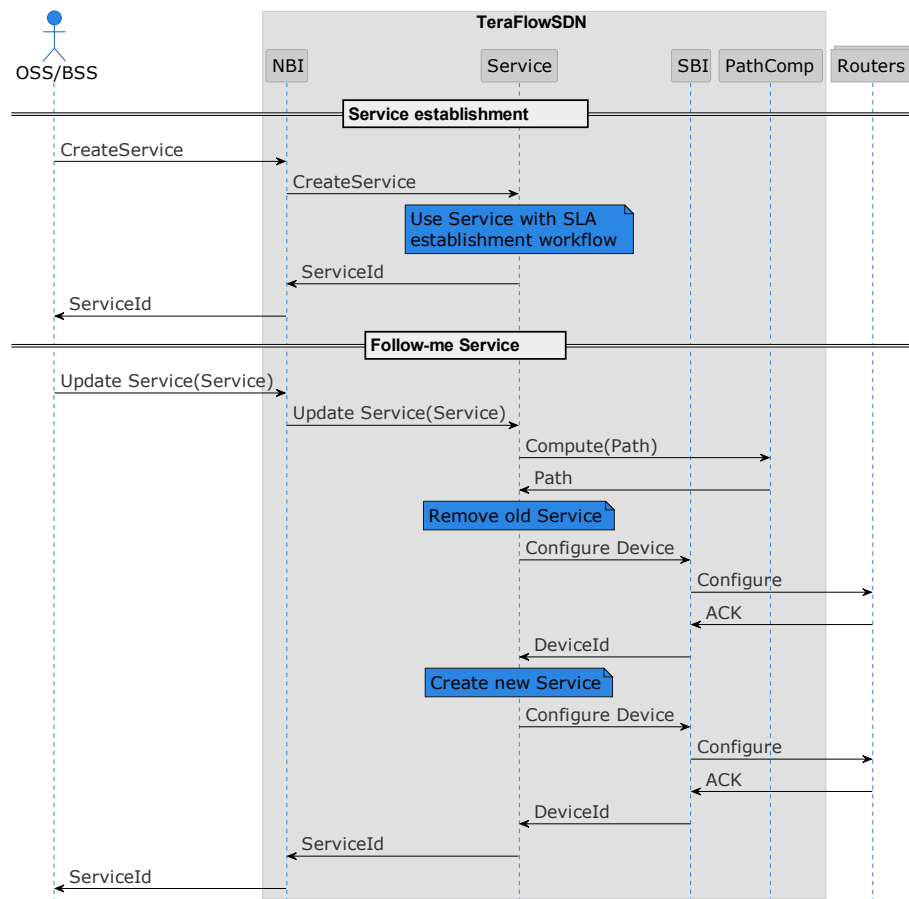


Figure 54. Scenario 2 workflow: Location-aware Service updates

6.6. Preliminary Performance Evaluation

In this section, we describe the preliminary results for the proposed scenario. Results are provided in form of tables, plots, diagrams, and screenshots. Final evaluation of the scenario will be performed during 2023 and documented in D5.3.

6.6.1. Inter-domain Provisioning using Transport Network Slices with SLA

This workflow has been validated and demonstrated in [OECC22]. The design of the inter-domain component is based on three use cases:

- service preparation and activation,
- service modification, and
- synchronization of service monitoring data between domains.

In order to validate the proposed workflow in previous Section 6.5.1, Figure 55 shows the authentication sequence between two IDC from different TeraFlowSDN controllers. The permitted TeraFlowSDN peer information is stored in the configuration file.

Time	Source	Destination	Protocol	Length	Info
10.009	d1-interdomain	d2-interdomain	GRPC	492	Magic, SETTINGS[0], SETTINGS[0], HEADERS[1]: POST /interdomain.InterdomainService/Authenticate, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.AuthenticationResult, HEADERS[1], WINDOW_UPDATE[0]
10.010	d2-interdomain	d1-interdomain	GRPC	261	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
10.127	d2-interdomain	d1-interdomain	GRPC	423	SETTINGS[0], HEADERS[1]: POST /interdomain.InterdomainService/Authenticate, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.AuthenticationResult, HEADERS[1], WINDOW_UPDATE[0]
10.128	d1-interdomain	d2-interdomain	GRPC	261	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]

Figure 55. Wireshark capture of Authenticate sequence

Once TeraFlowSDN controllers are authenticated, an E2E Transport Network Slice request can be triggered (for example from OpenSourceMANO). It can be observed as they are requested from inter-domain TeraFlowSDN controller 1 to controller 2. Figure 56 provides the wireshark traces between multiple instances of TeraFlowSDN controllers, and highlights the inter-domain provisioning.

REF	OSM	d1-compute	HTTP/JSON	226	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services HTTP/1.1, JavaScript Object Notation (application/json)
0.119	d1-compute	d1-slice	GRPC	435	SETTINGS[0], HEADERS[1]: POST /slice.SliceService/CreateSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.125	d1-slice	d1-compute	GRPC	299	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.126	d1-compute	OSM	HTTP/JSON	71	HTTP/1.0 201 CREATED, JavaScript Object Notation (application/json)
0.132	OSM	d1-compute	HTTP/JSON	439	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=1/site-network-accesses/ HTTP/1.1, JavaScript Object Notation (application/json)
0.276	d1-compute	d1-slice	GRPC	455	SETTINGS[0], HEADERS[1]: POST /slice.SliceService/UpdateSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.281	d1-slice	d1-compute	GRPC	299	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.282	d1-compute	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT
0.289	OSM	d1-compute	HTTP/JSON	439	POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=2/site-network-accesses/ HTTP/1.1, JavaScript Object Notation (application/json)
0.427	d1-compute	d1-slice	GRPC	475	SETTINGS[0], HEADERS[1]: POST /slice.SliceService/UpdateSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.440	d1-slice	d1-interdomain	GRPC	559	Magic, SETTINGS[0], SETTINGS[0], HEADERS[1]: POST /interdomain.InterdomainService/RequestSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.442	d1-interdomain	d2-interdomain	GRPC	280	HEADERS[3], WINDOW_UPDATE[3], DATA[3] (GRPC), WINDOW_UPDATE[0]
0.445	d2-interdomain	d1-interdomain	GRPC	131	HEADERS[3]: 200 OK, WINDOW_UPDATE[3], DATA[3], HEADERS[3] (GRPC), WINDOW_UPDATE[0]
0.446	d1-interdomain	d2-interdomain	GRPC	295	HEADERS[5], WINDOW_UPDATE[5], DATA[5] (GRPC), WINDOW_UPDATE[0]
0.448	d2-interdomain	d2-slice	GRPC	487	SETTINGS[0], HEADERS[1]: POST /slice.SliceService/CreateSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.459	d2-slice	d2-service	GRPC	456	SETTINGS[0], HEADERS[1]: POST /service.ServiceService/CreateService, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
0.469	d2-service	d2-slice	GRPC	311	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.ServiceId, HEADERS[1], WINDOW_UPDATE[0]
1.262	d2-slice	d2-interdomain	GRPC	311	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
1.282	d1-interdomain	d1-slice	GRPC	305	HEADERS[5]: 200 OK, WINDOW_UPDATE[5], DATA[5], HEADERS[5] (GRPC), WINDOW_UPDATE[0]
1.283	d1-interdomain	d1-slice	GRPC	484	SETTINGS[0], HEADERS[1]: POST /slice.SliceService/CreateSlice, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
1.292	d1-slice	d1-service	GRPC	452	SETTINGS[0], HEADERS[1]: POST /service.ServiceService/CreateService, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
1.303	d1-service	d1-slice	GRPC	308	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.ServiceId, HEADERS[1], WINDOW_UPDATE[0]
2.119	d1-slice	d1-interdomain	GRPC	308	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
2.133	d1-interdomain	d1-slice	GRPC	299	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
2.181	d1-slice	d1-compute	GRPC	299	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF) context.SliceId, HEADERS[1], WINDOW_UPDATE[0]
2.183	d1-compute	OSM	HTTP	176	HTTP/1.0 204 NO CONTENT

Figure 56. Inter-domain End-to-End Transport Network Slice deployment

After scenario validation, in D5.3 we will analyze the following Metrics for this workflow.

Name	Value	Comment
Service setup delay	-	This KPI will be analysed in D5.3
Economic	-	This KPI will be analysed in D5.3

6.6.2. Distributed Ledger Technologies

This workflow has been validated and demonstrated in [NFV22]. From the outside, it might seem similar to the previously presented workflow, but in this case, we are using DLT component to interact between multiple TFS instances.

In Section 6.5.2, we have detailed the workflow for establishing an inter-domain transport network slice. In our validation, we have provisioned an inter-domain Transport Network Slice. Figure 57 details a Wireshark capture with the externally-visible messages involved in this test, and taken from D4.2. It is worth noting that the DLT Connector and DLT Gateway run within the same pod and Kubernetes is not exposing these packets, so Wireshark cannot capture them. In that figure, an arbitrary TeraFlowSDN component issues a request to add a device into the Context component (messages 2009 and 2015). Then, that component triggers the recording of that device into the Blockchain (message 2030). To do that, the arbitrary component issues a “RecordDevice” request to the DLT component, that is received by the DLT Connector. The DLT connector then retrieves the device details from the Context component (not shown since it is an internal Kubernetes communication) and forwards the request to the DLT Gateway that triggers the upload into the Blockchain hosted by NEC in Germany (messages 2056-2885) Upon the operation is done, the DLT Gateway replies to the DLT Connector and the later replies to the requesting component (message 2888).

No.	Time	Source	Destination	Protocol	Length	Info
2009	4.749	10.0.2.10	10.1.105.117	GRPC	388	SETTINGS[0], HEADERS[1]: POST /context.ContextService/SetDevice, WINDOW_UPDATE[1], DATA[1] (GRPC)
2015	4.750	10.1.105.117	10.0.2.10	GRPC	234	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), HEADERS[1], WINDOW_UPDATE[0]
2030	4.751	10.0.2.10	10.1.105.77	GRPC	392	SETTINGS[0], HEADERS[1]: POST /dlt.DltConnectorService/RecordDevice, WINDOW_UPDATE[1], DATA[1]
2056	4.763	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1337	Application Data
2063	4.815	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2069	4.820	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47786 [ACK] Seq=53 Ack=1321 Win=32729 Len=1460 [TCP segment of a reassembled PDU]
2073	4.826	teraflow.nlehd.de	10.1.105.77	TLSv1.2	637	Application Data
2078	4.841	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1720	Application Data
2084	4.842	10.1.105.77	teraflow.nlehd.de	TLSv1.2	1720	Application Data
2104	4.916	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2105	4.916	teraflow.nlehd.de	10.1.105.77	TLSv1.2	108	Application Data
2112	4.916	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 39986 [PSH, ACK] Seq=53 Ack=1696 Win=32737 Len=1460 [TCP segment of a reassembled PDU]
2113	4.916	teraflow.nlehd.de	10.1.105.77	TCP	1516	9051 → 50938 [ACK] Seq=53 Ack=1696 Win=32737 Len=1460 [TCP segment of a reassembled PDU]
2120	4.917	teraflow.nlehd.de	10.1.105.77	TLSv1.2	737	Application Data
2121	4.917	teraflow.nlehd.de	10.1.105.77	TLSv1.2	730	Application Data
2128	4.918	10.1.105.77	teraflow.nlehd.de	TLSv1.2	112	Application Data
2848	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [ACK] Seq=1 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2849	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [PSH, ACK] Seq=1461 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2850	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47792 [ACK] Seq=1 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2851	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47792 [PSH, ACK] Seq=1461 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2852	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [ACK] Seq=2921 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2853	7.071	teraflow.nlehd.de	10.1.105.77	TCP	1516	7051 → 47798 [PSH, ACK] Seq=4381 Ack=1 Win=31879 Len=1460 [TCP segment of a reassembled PDU]
2854	7.071	teraflow.nlehd.de	10.1.105.77	TCP	2976	7051 → 47792 [ACK] Seq=2921 Ack=1 Win=31879 Len=2920 [TCP segment of a reassembled PDU]
2871	7.072	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2872	7.072	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2881	7.082	teraflow.nlehd.de	10.1.105.77	TCP	5896	9051 → 40978 [ACK] Seq=1 Ack=1 Win=31879 Len=5840 [TCP segment of a reassembled PDU]
2885	7.082	teraflow.nlehd.de	10.1.105.77	TLSv1.2	1282	Application Data
2888	7.088	10.1.105.77	10.0.2.10	GRPC	219	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC), HEADERS[1], WINDOW_UPDATE[0]

Figure 57. Transport Network topology for DLT evaluation

Figure 58 shows the Cumulative distribution function (CDF) of the DLT latency for the generated 100 requests. We observe that the delay takes around 10 seconds. The main contribution of this delay is due to the cost of uploading the record into the blockchain due to the consensus and ordering constraints that need to be fulfilled.

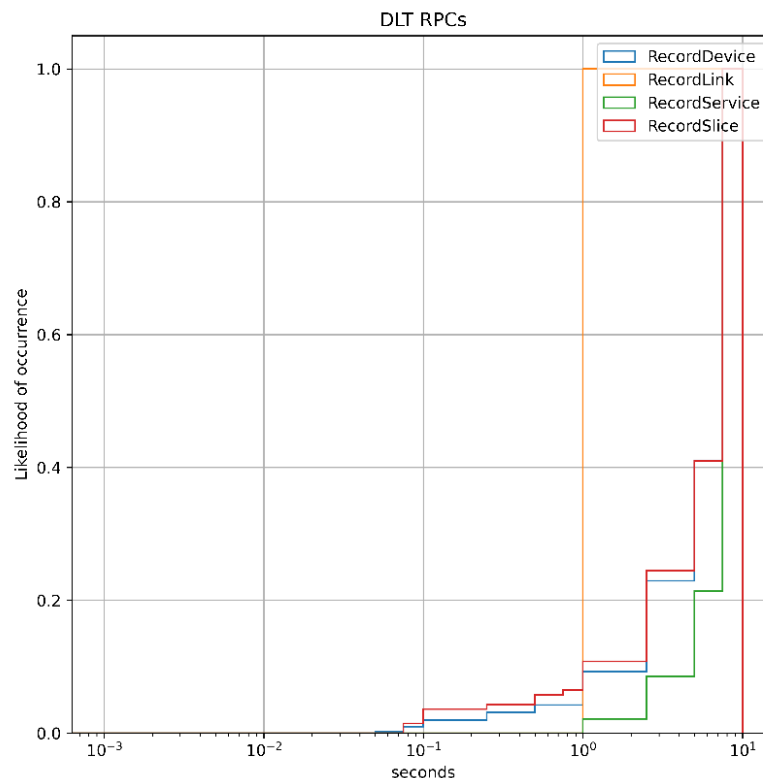


Figure 58. CDF for the DLT Delay

Figure 59 shows the complete information for an inter-domain transport network slice as shown in TeraFlowSDN User Interface. It may be observed that multiple sub-slices have been required.

Context: admin		
UUID: idc-slice	Endpoints	Device
	int	DC1
Owner:	int	DC2
Status: ACTIVE		
Constraints:		
Kind	Type	Value
Custom	bandwidth[gbps]	10.0
Configurations:		
Key	Value	
/settings	• vlan_id: 400	
/device[DC1]/endpoint[int]/settings	• vlan_id: 400	
/device[DC2]/endpoint[int]/settings	• vlan_id: 400	
Service Id	Sub-slices	
	D2 / idc-slice	
	D4 / idc-slice	
	admin / b14fe094-5adf-4b56-901d-498dfaf94cd8	

Figure 59. Inter-domain Transport Network Slice that includes sub-slices

Figure 60 provides the details of the local (from the initial domain perspective) requested sub-slice.

Slice b14fe094-5adf-4b56-901d-498dfaf94cd8		
Back to slice list		
Context: admin		
UUID: b14fe094-5adf-4b56-901d-498dfaf94cd8	Endpoints	Device
	int	DC1
Owner:	D2	R4@D1
Status: ACTIVE		
Constraints:		
Kind	Type	Value
Custom	bandwidth[gbps]	10.0
Configurations:		
Key	Value	
/settings	• vlan_id: 400	
/device[DC1]/endpoint[int]/settings	• vlan_id: 400	
/device[DC2]/endpoint[int]/settings	• vlan_id: 400	
Service Id	Sub-slices	
	admin / b14fe094-5adf-4b56-901d-498dfaf94cd8	

Figure 60. Sub-slice information details

After scenario validation, in D5.3 we will analyze the following Metrics for this workflow.

Name	Value	Comment
Trust/privacy	-	This KPI will be analysed in D5.3
DLT transaction delay	10s	This KPI will be further analysed in D5.3

6.6.3. Service/Slice Request Scalability

Only context evaluation has been provided with regard to scalability in D3.2. We aim to evaluate the KPI in D5.3, following the approach described in the related workflow.

Name	Value	Comment
Multi-tenancy	>100 tenants	This KPI will be analysed in D5.3

6.6.4. Location-aware Service Updates

This use case has not been validated and tested. Details of the use case and its evaluation will be provided in D5.3.

Name	Value	Comment
Positioning	-	This KPI will be analysed in D5.3

6.7. Pending Work and Summary

The Metrics Collection Framework has allowed us to start the recollection of KPIs, but until now, we are fixing several integration issues that preclude us from providing accurate results. To this end, we are fixing the reported issues, and the measurements will be provided in D5.3. and Table 5 summarizes their status.

Table 5. Target and achieved KPIs and KVs for Scenario 2

KPI	Target	Validation results
Multi-tenancy	> 100 tenants	This KPI will be completed in D5.3.
Trust/privacy	100% secured connections	This KPI will be completed in D5.3.
DLT transaction delay	10s	This KPI will be completed in D5.3.
Positioning	100% vehicles	This KPI will be completed in D5.3.
Social	< 20% cost	This KPI will be completed in D5.3.

7. Scenario 3: Cybersecurity

This section presents the third scenario in TeraFlow, focusing on cybersecurity. It can be considered an operator-led scenario, as it focuses on analysing and mitigating security attacks on multiple network layers, spanning data and control planes. Firstly, we introduce the scenario. Secondly, we present its alignment with the TeraFlow architecture. Thirdly, we present the scenario setup in the laboratories. Fourthly, we present the metrics and KPIs that are relevant for the scenario. Fifthly, we introduce the designed workflows and the current deployments. Sixthly, preliminary performance evaluation numbers are presented, where available. Finally, pending work and summary are provided.

7.1. Scenario Introduction

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

In this context, the 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.

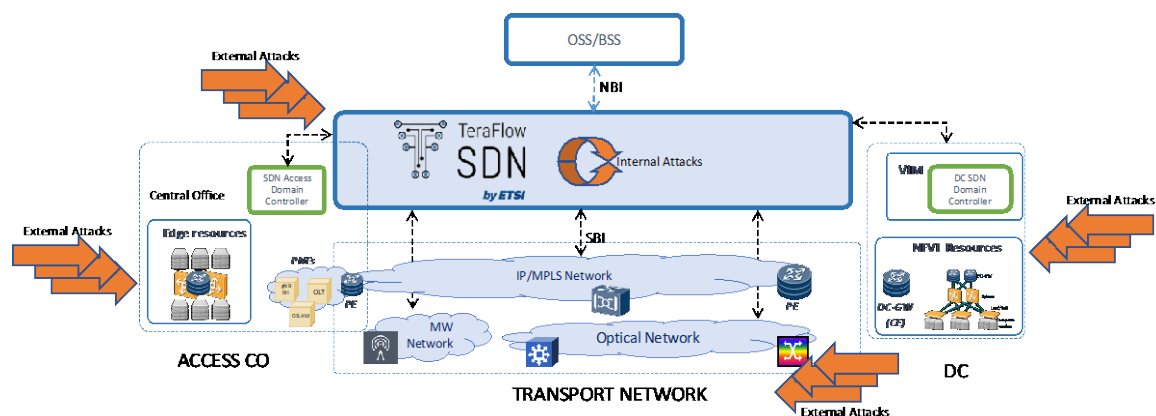


Figure 61. Cybersecurity scenario and threats

Figure 61 depicts an example of the envisioned Cybersecurity scenario and of the threats in the context of an automated network. Attacks may target the IP or the optical layers at the data plane. Attacks exploiting the IP layer traverse or target devices located in the access segment (e.g., edge DCs), the core network, or core DCs. In this case, per-packet inspection is necessary to detect and identify attacks, enabling their mitigation. However, inspecting packets is a demanding operation. Executing this process at a central packet inspector instance is impractical. Packets must be transported from the remote site, e.g., Central Office (CO) or DC, to a central location, incurring significant traffic and computing loads. Therefore, 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. Therefore, designing accurate, fast, and scalable optical attack detection, identification, and mitigation mechanisms becomes critical to avoid or minimize data losses and breaches. We focus on the scalable attack detection problem in this deliverable, while mitigation aspects will be tackled in the next deliverable.

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 the ML models are not exposed or vulnerable to these attacks. To this end, Generative Adversarial Networks are combined with ML-based models using a Black-Box approach to generate variations of attacks that help in training ML models immune to such adversarial attacks.

7.2. Alignment with TeraFlow Architecture

The Cybersecurity scenario will validate several components, use cases, NBI/SBI interfaces, and protocols. Three components compose the Cybersecurity assessment within TeraFlowSDN: Centralized Attack Detector (CAD), attack inference, and attack mitigator. The main components involved in this scenario are highlighted in Figure 62. They are deployed in different containers to take advantage of the scalability and reliability features of cloud-native applications. The Cybersecurity components integrate with TeraFlowSDN core components in several ways, as illustrated in Figure 62. The Service component is used for provisioning and (re)configuration tasks necessary to mitigate detected attacks. Integration with the Device component is also needed to perform changes to specific devices when mitigation actions are needed. The Context component is used to detect service updates (i.e., creation and deletion) and retrieve service details. The Monitoring component is used both to retrieve monitoring data as well as to store the result of the security assessment process.

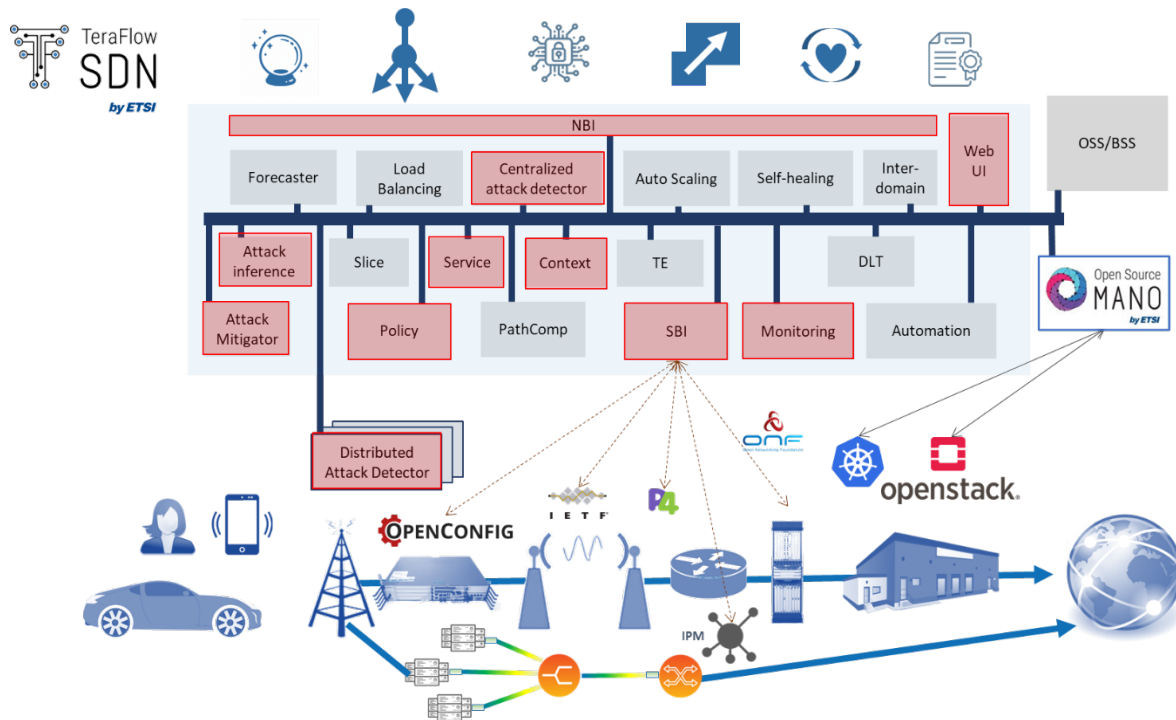


Figure 62. TeraFlow components used in the cybersecurity scenario

In addition to the components deployed within TeraFlowSDN, a Distributed Attack Detector (DAD) located at remote sites interacts with the TeraFlowSDN Controller. Note that the DAD is an external component running outside the TeraFlowSDN. We refer to D4.1 for a complete description of the components responsible for the Cybersecurity assessment. Use cases of interest for testing the validity of these components and apps are *monitoring*, *service*, *context*, *device*, *NBI*, and *path computation*. More details about these use cases are provided in D2.2.

7.3. Scenario Setup

In the following, we describe the setup used to validate the implementation of Scenario 3. As Scenario 3 has two main targets (i.e., optical and IP layers), we present two separate setups. First, for the Layer 3 cybersecurity experiments, the target is to prepare a setup that allows us to reproduce previously recorded cryptomining attacks. We first capture packets from the cryptomining attack that are reproduced in the setup environment. Second, the objective of the optical physical layer attacks is to reproduce previously-capture Optical Performance Monitoring (OPM) data from malicious attack conditions.

7.3.1. MouseWorld Setup for Layer 3 Cybersecurity Experiments

Classical VPN services provided by network operators are not aware of cybersecurity attacks, because such capability would require additional appliances or solutions (Firewalls, Intrusion Detection System - IDS, etc.) to cope with attacks, in client facilities or through traffic engineering (redirection to a cleaning center) on the network operator side. This has been considered a disadvantage if we compare it with Software-Defined WAN (SD-WAN) or Secure Access Service Edge (SASE) overlay solutions. This demonstration setup aims to represent a common situation where TeraFlowSDN can monitor MPLS VPN traffic and apply ML techniques to detect and mitigate a complex representative attack such as

cryptomining. The setup considered for this demonstration is illustrated in Figure 63. A Level 3 VPN service (L3VPN) is deployed using the TeraflowSDN controller in the Telefónica facilities (details in D5.1 section 6.2), including Mouseworld Lab for traffic attack generation and Future network Lab for IP devices and SDN deployment. The TeraFlowSDN Controller activates this service using provisioned templates over the standardized IETF NETCONF southbound interface against the different Provider Edge (PE) routers from ADVA manufacturer. In this demonstration, branch and central office, are implemented with Mouseworld OpenStack resources through virtual machines that replay a mix of normal traffic with a cryptomining malware activity. Also, the central office provides internet access.

As part of the VPN service provisioning process done by the TeraFlowSDN Controller, a mirror of the traffic in the logical interfaces that conform to the L3VPN is also enforced to copy the traffic towards the distributed attack detector co-located with the ADVA router. This distributed attack detector component will extract and calculate statistical features from network flows to be delivered to the TeraFlowSDN Controller for further processing. The Cybersecurity components will identify the attack as a cryptomining activity and propose a mitigation solution to the TeraFlowSDN core components that will trigger the mitigation. This mitigation will be instantiated as a new customized Access Control List (ACL) rule in the ADVA router with specific parameters (transport protocol, destination IP address and destination port). This rule can be enforced in additional PE routers that are part of the L3VPN to increase the mitigation capacity.

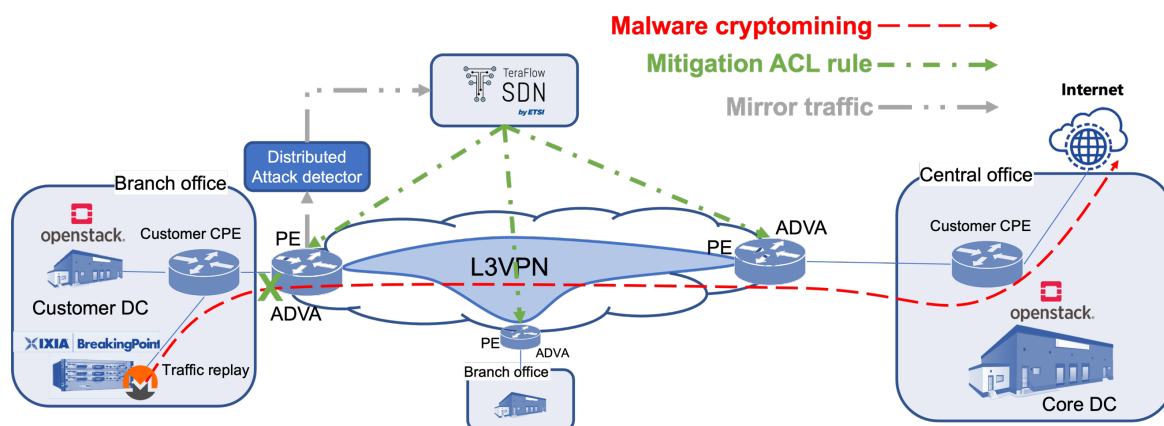


Figure 63. Deployment of the cybersecurity scenario focusing on L3

7.3.2. Emulated Optical Setup for Optical Cybersecurity Experiments

The objective of this setup is to enable us to reproduce OPM data from optical physical layer attacks captured in a real-world testbed. Since scalability is a key concern in this scenario, we need to be able to quickly create a high number of optical services being operated with the help of TeraFlowSDN. Then, TeraFlowSDN is responsible for its optical cybersecurity assessment.

Due to the high complexity, time constraints, and cost associated with reproducing experiments with real optical devices, we decided to use an emulated optical infrastructure. The high complexity comes from the fact that imposing attacks on the physical layer of optical networks require special equipment, and very specific configurations. Moreover, there are several time constraints. For instance, once (re)configured, optical devices may require a few minutes to a few hours to reach a stable working condition, making it impractical for experiments to be reproduced several times, as required in our case.

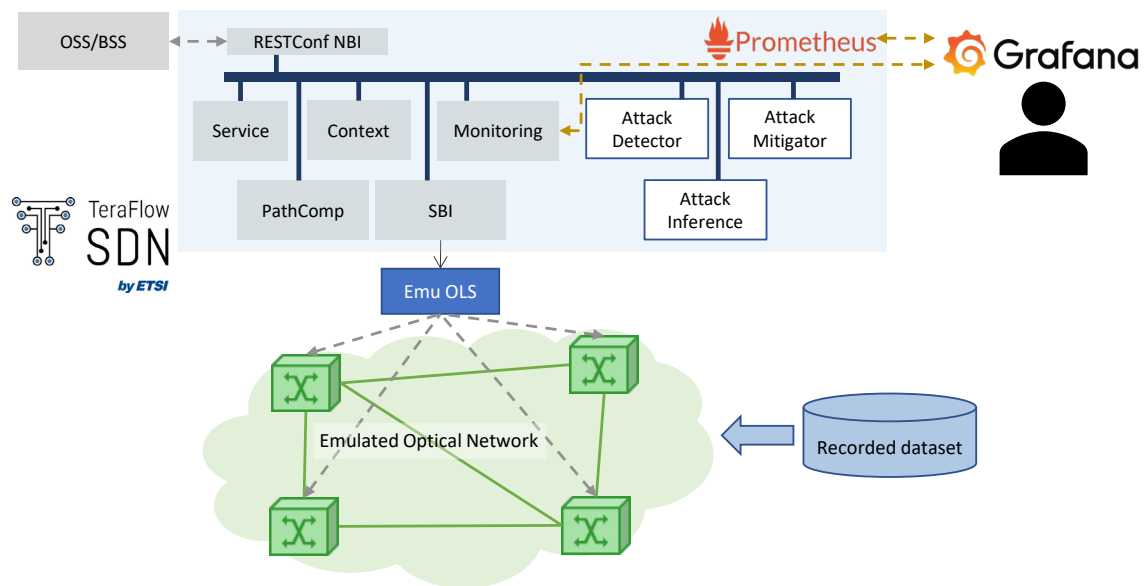


Figure 64. Simplified view of the emulated deployment

Figure 64 presents a simplified illustration of the setup considered. We use TeraFlowSDN at the control plane. The components tested are the Centralized Attack Detector (referred to hereinafter as Attack Detector in the context of the optical layer), Attack Inference, and Attack Mitigator. The Monitoring component and Prometheus are used as data sources for the visualizations, which are created using Grafana. We also created a custom script that acts as an OSS/BSS and can be configured to perform optical service requests to TeraFlowSDN's SBI.

The emulated optical network was configured to replay the dataset reported in [JLT2019]. The dataset consists of OPM samples collected from a real testbed using commercial equipment. The equipment consisted of coherent transceivers, able to report detailed OPM parameters with a frequency of once per minute. The data was collected using a custom-made agent and consolidated into a dataset. The OPM features captured are:

- Chromatic dispersion
- Differential group delay
- Optical signal-to-noise ratio
- Polarization-dependent loss
- Q-factor
- Block errors before FEC
- Bit error rate before FEC
- Uncorrected blocks
- Bit error rate after FEC
- Optical received power
- Optical received frequency
- Loss of signal

In addition to normal operating conditions, the setup was configured to emulate three types of attack: in-band jamming, out-of-band jamming, and polarization scrambling. For each type of attack, a light and a strong intensity were imposed, forming seven attack conditions:

1. Normal operating conditions
2. Light in-band jamming attack
3. Strong in-band jamming attack
4. Light out-of-band jamming attack
5. Strong out-of-band jamming attack
6. Light polarization scrambling attack
7. Strong polarization scrambling attack

Each attack condition was captured for 24 hours, which accounts for any transition period that the transmission might undergo (i.e., instability of the channel due to changes).

The dataset was used by a custom-made OLS that communicates with TeraFlowSDN emulating the optical network. The emulation happens in the optical service provisioning and the optical service monitoring phases. The emulated OLS makes it easy to accommodate any request without resource constraints during optical service provisioning. This enables us to perform stress tests and validate the scalability properties of the cybersecurity component.

During optical service monitoring, the OLS reports OPM values to TeraFlowSDN according to a configurable setting. By default, new optical services will replay data pertaining to normal operating conditions. However, each channel can be associated with a particular attack condition, upon which the emulated OLS will start replaying, for that specific service, the data captured from the attack condition. This allows us to validate the attack mitigator's scalability and actions, which are planned for D5.3.

7.4. Scenario Metrics

Scenario 3 integrates several components and implements several workflows that need to be evaluated. Table 6 summarizes the KPIs and KVs to be evaluated, their relevance, and the definition of how they are measured.

Table 6. KPIs and KVs for Scenario 3

Name	Description	Relevance		Definition of measurement	Component
		Layer 3	Optical		
Security	> 99% accuracy (known attacks)	Attacks need to be detected with high accuracy to make sure they do not remain undetected or unaddressed in the network		Measuring the performance of the trained model over a testing dataset	CAD and Attack Inference. Offline measurement based on the ML model characteristic and collected dataset

	> 90% accuracy (unseen attacks)	- (Not relevant due to specific focus on cryptomining attacks)	Detecting attacks that are not present in the current catalogue of attacks improves network preparedness	Measuring the performance of the model over attacks that are not present in the training dataset	Attack Inference. Offline measurement based on the ML model characteristic and collected dataset
	> 30% reduction of attack response latency	Attacks should be detected as early as possible to minimize the damage they can cause	- (Not relevant due to the significant time required to reconfigure optical devices)	*	Attack mitigator
Reliability	> 90% accuracy in detecting and avoiding known adversarial attacks	Adversarial L3 attacks generated by malicious actors need to be detected with high accuracy to prevent attackers being able to bypass the detection system	- (Not relevant due to information flow traversing only core components)	Measuring the performance of the model on detecting unseen adversarial attacks	Offline measurement based on the ML model characteristic and collected dataset

Energy	> 25% resource consumption	The machine learning model responsible of detecting L3 attacks should be optimized to be as energy efficient as possible with minimal degradation in accuracy	- (Not relevant due to low frequency of inferences with respect to the L3)	Measuring the reduction in total average energy consumption and average resource utilization metrics of the machine learning model responsible for detecting L3 attacks using thirteen different combinations of state-of-the-art optimization techniques	Offline measurement based on the ML model characteristic and collected dataset
<p>* Measuring the latency between the connection start and the detection of the attack. We will measure the latency reduction with and without using an ML model deployed at the edge (i.e., at the Distributed Attack Detector). The measurements will include the mean, minimum, maximum, and standard deviation of the time required to detect a cryptocurrency mining attack in different repetitions with and without an ML model deployed at the Distributed Attack Detector.</p>					

7.5. Workflows and Current Deployment

In the case of Scenario 3, two complimentary yet distinct workflows need to be implemented. One is related to monitoring Layer 3 flows, which work with a monitoring cycle that depends on the (user) traffic under exam. The second is the monitoring of optical connectivity services, which work with a monitoring cycle that the TeraFlowSDN administrator can define.

This section presents a few general workflows that illustrate how the Cybersecurity component interacts with other TeraFlowSDN core components. Later, the specifics of the Layer 3 and Optical workflows will be detailed.

Figure 65 shows the general communication among the core and cybersecurity components when a new service is created. Firstly, during start-up, the Cybersecurity component subscribes to service events from the Context component. Then, when a service request is received, the service setup stage is triggered, performing the necessary changes involving several components of TeraFlowSDN. A detailed workflow of the service setup can be found in D3.2. After the service is set up, the service identifier is returned to the customer who requested the service. Then, the KPI setup stage starts. At this stage, the Cybersecurity component is notified by the Context component about creating the new service. Then, the Cybersecurity will create relevant KPIs in the Monitoring component. The specifics of this workflow for Layer 3 and Optical Cybersecurity will be detailed later in this section.

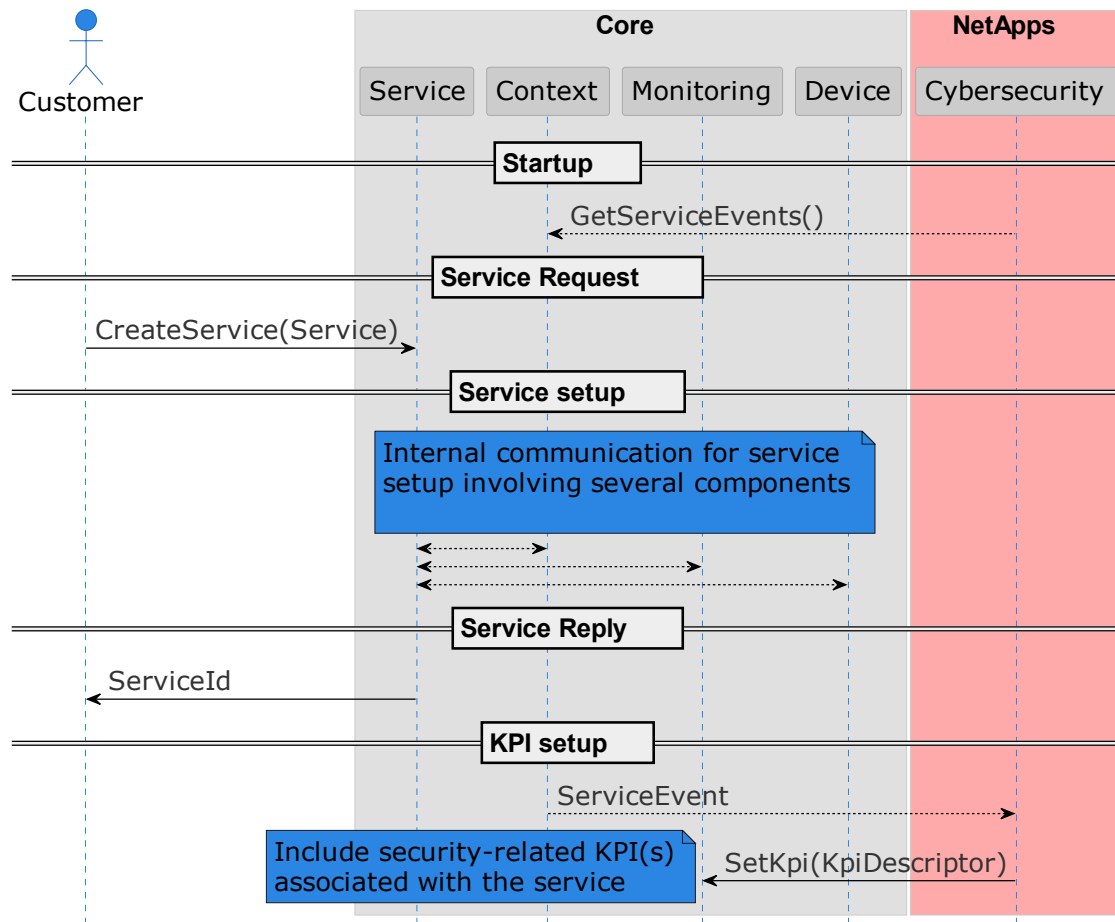


Figure 65. Scenario 3 workflow: General communication when creating a new service

7.5.1. Layer 3 Cybersecurity

In this section, we will describe the specific workflows that implement the detection and mitigation of network attacks at the IP layer.

7.5.1.1. Traffic Capture and Feature Extraction at the Network Edge

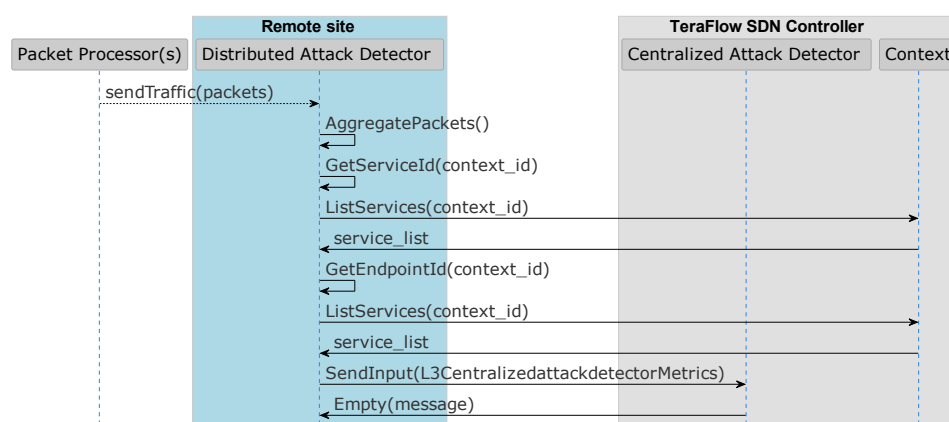


Figure 66. Scenario 3 workflow: Traffic Capture and Feature Extraction Workflow

We assume that the DAD receives a copy of the traffic (i.e., all the packets) traversing the endpoint being monitored for L3 attacks. After the DAD receives the traffic, it is grouped into flow-level statistics using the TSTAT files. Figure 66 shows that the DAD communicates via RCP methods with the Context component to obtain the service_id and endpoint_id attributes, so the connection is traceable in the TeraFlowSDN, and the mitigation strategies can later be implemented on the correct devices. Once all the connection data is grouped into an L3CentralizedattackdetectorMetrics object, it is sent via the RCP method SendInput to the CAD.

7.5.1.2. Detect Known Attacks using Supervised ML

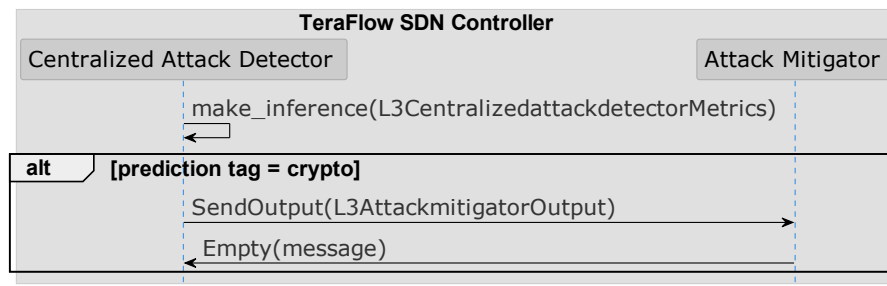


Figure 67. Scenario 3 workflow: Attack Detection Workflow (Layer 3)

Figure 67 shows the workflow for the detection of known attacks. The CAD component receives and stores flow statistics from L3CentralizedattackdetectorMetrics objects. A function is then called with these objects as the input to perform the Machine Learning inference that will classify the data as either belonging to a cryptomining attack or not. If the flow statistic has been classified as a cryptomining attack, the SendOutput RCP method will be called. It will send the necessary flow data and inference data to the Attack Mitigator component in an L3AttackmitigatorOutput object.

7.5.1.3. Mitigate Detected Attacks

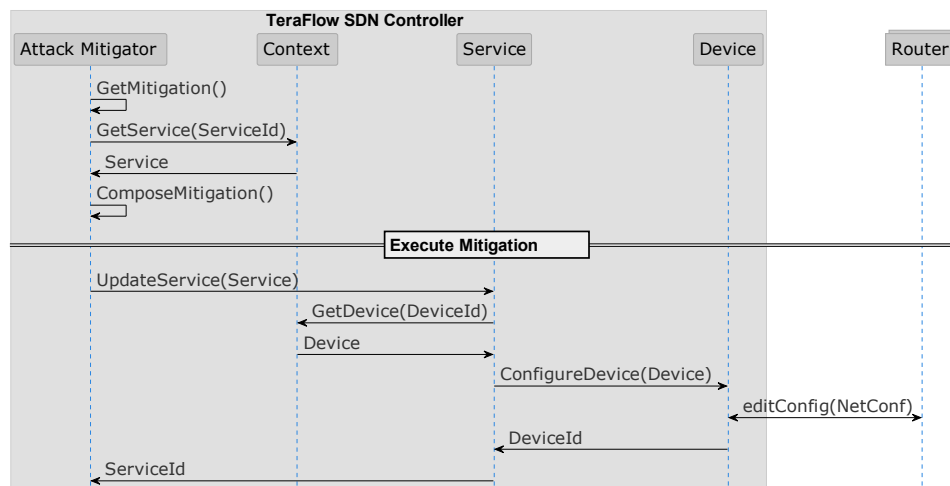


Figure 68. Scenario 3 workflow: Attack Mitigation Workflow (Layer 3)

Figure 68 shows that after the AM component receives the connection data belonging to a cryptomining attack, it will create a mitigation strategy. As of now, that mitigation strategy is to generate a rule to drop the connection. AM will then need to communicate with the Context component to receive the Service instance belonging to the service_id that is included in the connection data. After receiving the Service object, the ComposeMitigation method will add the new rule to drop the connection to it. After calling the RCP method UpdateService with the modified

service instance, the Teraflow OS will propagate the changes to the Device component, and it will modify the ACL rules in the Router to drop the connection, thus finishing the current mitigation strategy.

7.5.1.4. Monitor Relevant Cybersecurity-related Metrics

The Centralized Attack Detector monitors five relevant KPIs for each active service. Below, we list the cybersecurity KPIs that are observed and recorded and their associated KPI sample type:

- Cryptomining detector confidence in security status over the last time interval (KPI_ML_CONFIDENCE);
- Security status against cryptomining attacks of the service in a time interval (KPI_L3_CRYPTO_SECURITY_STATUS);
- Number of attack connections detected in a time interval (KPI_UNIQUE_ATTACK_CONNS);
- Number of unique compromised clients of the service in a time interval (KPI_UNIQUE_COMPROMISED_CLIENTS);
- Number of unique attackers of the service in a time interval (KPI_UNIQUE_ATTACKERS).

The values of KPI_L3_ML_CONFIDENCE are collected for predictions that take place during a specific time interval (e.g., 5 seconds). This is done separately for predictions that correspond to an attack and predictions that correspond to normal traffic. At the end of each time interval, the values of both lists are aggregated independently calculating the average. If an attack connection occurred during that time interval, the average confidence of the predictions corresponding to an attack are sent to the Monitoring component as KPI_L3_ML_CONFIDENCE and "1" as KPI_L3_SECURITY_STATUS_SERVICE. Otherwise, the average confidence of the predictions corresponding to normal traffic is sent to the Monitoring component as KPI_L3_ML_CONFIDENCE and "0" as KPI_L3_SECURITY_STATUS_SERVICE.

The KPI_L3_UNIQUE_ATTACK_CONNS counts the number of unique attack connections that were detected in each time interval. Like the previous KPIs, these values are collected during each time interval. Once the interval is over, these values are aggregated and sent to the monitoring component. Note that the packet aggregator running in the Distributed Attack Detector component aggregates the new packets from the same connections as soon as they are received, and the characteristics are sent to the ML model. For this reason, if subsequent packets are received from the same connections, the Decentralized Attack Detector will produce new statistics that the ML model will also ingest. For this reason, connections may be detected as an attack more than once. However, in KPI_L3_UNIQUE_ATTACK_CONNS we will only count these repeated connections once.

Similar to KPI_L3_UNIQUE_ATTACK_CONNS, KPI_UNIQUE_COMPROMISED_CLIENTS measures the number of compromised cryptocurrency clients in each time interval by counting the number of flows that correspond to the same source IP. On the other hand, KPI_UNIQUE_ATTACKERS measures the number of unique attackers in each time interval by counting the number of flows that correspond to the same destination IP. KPI_L3_UNIQUE_ATTACK_CONNS provides a measure of the intensity with which compromised clients attack the network. KPI_UNIQUE_COMPROMISED_CLIENTS and KPI_UNIQUE_ATTACKERS extend this information by revealing the scale of the compromised network and quantifying how many attackers are involved in attacking the network.

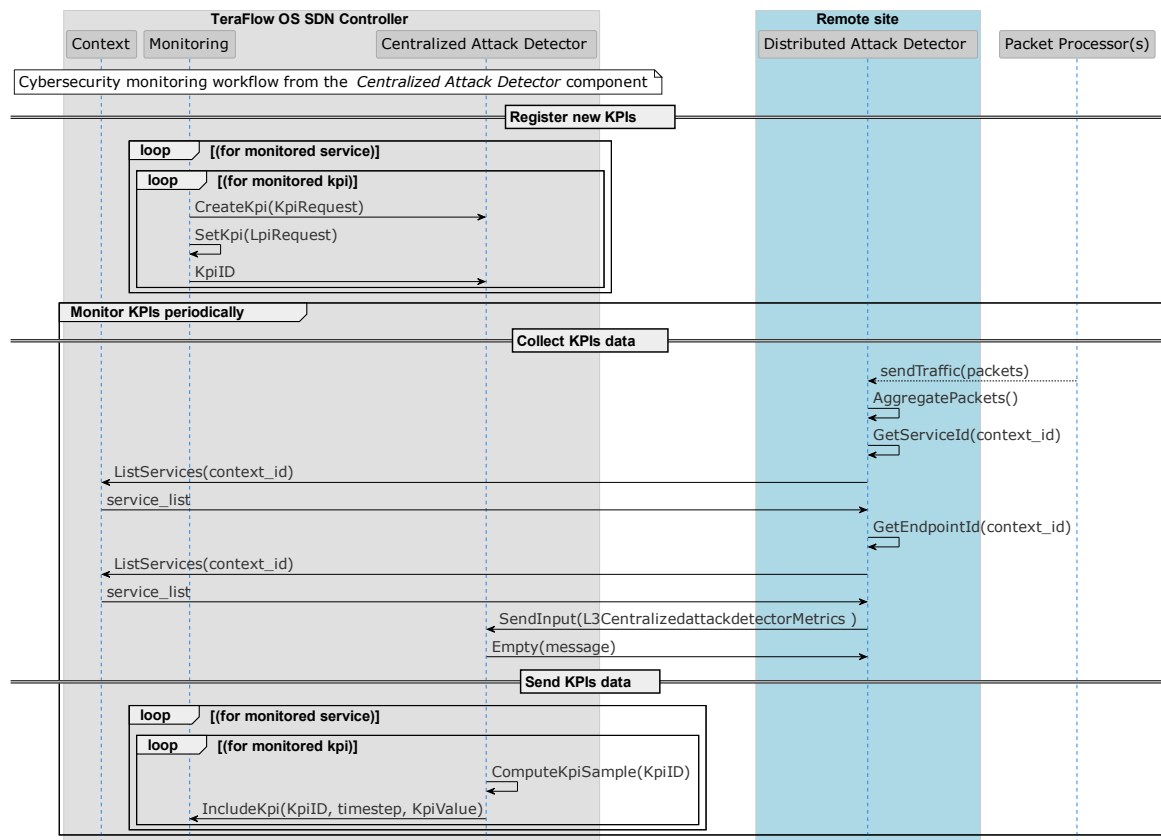


Figure 69. Scenario 3 workflow: Cybersecurity KPIs Monitoring Workflow (Layer 3)

Figure 69 shows that the Centralized Attack Detector creates these KPIs at launch time by registering `KpiRequest` for each KPI through the Monitoring client and requesting the Monitoring service process to create and add them to the Management Database (DB). For each `KpiRequest`, a `KpiDescriptor` includes service information, device and endpoint identifiers, and the description and KPI sample type of each KPI. After successful creation, the KPIs can be effectively monitored by sending samples to the Monitoring service via the `IncludeKpi` RPC method. When the Monitoring service receives each sample, they are introduced into the Metrics DB to be accessible through the Grafana dashboard.

7.5.2. Optical Cybersecurity

For the optical cybersecurity, the Centralized Attack Detector (hereinafter denoted simply as Attack Detector) has three main tasks:

1. To maintain a list of the currently active optical services, their identifiers, and relevant KPI identifiers;
2. To periodically trigger the cybersecurity assessment loop for each active optical service;
3. To coordinate the cybersecurity assessment of each service.

These responsibilities are too extensive for a single component and diverge in terms of how they are triggered and processed. For instance, to maintain an updated list of the currently active services, the cybersecurity app needs to subscribe to Context events related to services (i.e., service creation, service update, service deletion). These events will be reported whenever changes happen. On the other hand, the cybersecurity assessment loop needs to be executed periodically, irrespective of other events. Moreover, this task needs to be done by a single instance, i.e., the component responsible for

this task cannot be replicated. Finally, the execution of the assessment for each service needs to be a scalable process, meaning that the component performing this task needs to scale.

These facts translate into the need for one stateful component and another stateless one. Stateful components maintain the internal state necessary for their correct functioning. Nevertheless, on the other hand, they are hard (if not impossible) to replicate due to the need to establish a protocol to share the state. On the other hand, Stateless components handle each request as an isolated process, and no state is saved across requests.

Due to these reasons, we divided the Attack Detector into two variations:

- Attack Manager: a *stateful* component with a single replica (i.e., does not scale) responsible for maintaining a list of current active optical services in the network, and for triggering the assessment of each optical service;
- Attack Detector: a *stateless* component that can have multiple replicas, where each call refers to the task of performing the assessment of a single optical service.

These two components cooperate in order to realize the optical cybersecurity assessment loop. For example, during the initialization of the Attack Manager, as illustrated in Figure 70, the Attack Manager queries the Context component for a list of current services. Note that this initialization procedure allows the Cybersecurity app to be resilient to restarts, which means that it can be started or restarted whenever needed. Furthermore, it means that the Cybersecurity app can be put into operation at any point in time (as opposed to having to activate it only during the startup of the entire TeraFlowSDN).

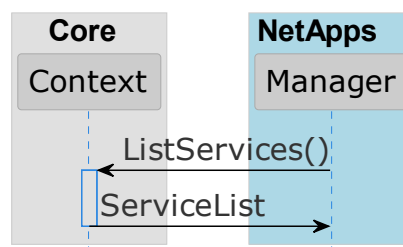


Figure 70. Scenario 3 workflow: Initialization of the optical cybersecurity components

Naturally, as the network is operated, new services will be created, and old services will be terminated. Therefore, the optical cybersecurity component needs to have an updated list with the active services. This could be obtained by repeating the workflow in Figure 70 for every new loop that is starting, i.e., querying the Context for a list of services. However, this approach would incur in substantial added load to the Context. To have keep the load of the optical cybersecurity loop as low as possible, we took advantage of the streaming capabilities of the Context to receive events related to services.

Figure 71 shows the workflow used to maintain an updated list of active services. For the sake of space, we focus on the service creation part, with the service deletion being very similar except for the triggering event. The figure shows that the Attack Manager is notified upon the creation of a new service. This allows the Attack Manager to include the newly created service in its internal list of active services and create relevant KPIs in the Monitoring.

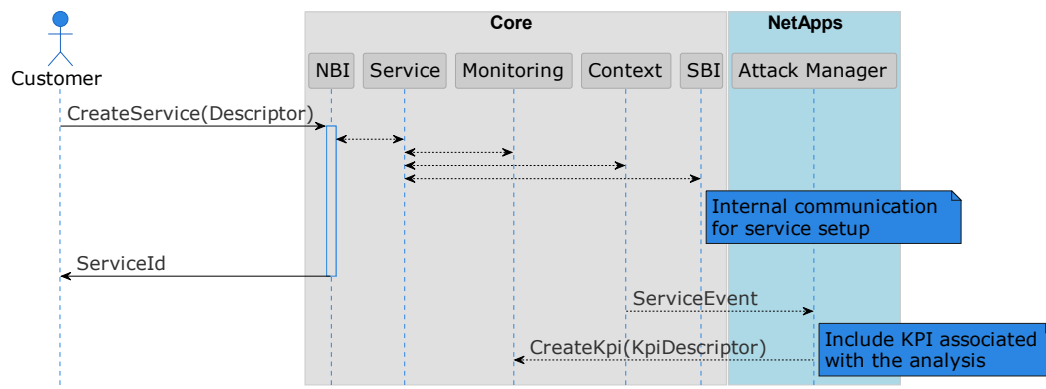


Figure 71. Scenario 3 workflow: Receiving service events from the Context component

Once the Attack Manager has a list with the optical services currently under operation in the network, the periodical cybersecurity assessment loop can take place. The specific steps and components involved in the loop will depend on the type of ML model used by the Attack Inference component. Supervised learning algorithms learn the properties of the system during training, which means that for the inference only the new(est) sample(s) are needed, i.e., the ones that were not assessed so far. On the other hand, unsupervised learning models do not have a training step, which means that they need a substantial number of samples at each inference to be able to determine which one(s) of the samples, if any, are anomalies (or attacks, as it is in our case).

One potential solution for both cases would be to rely on the Monitoring component to provide all the samples for all the inferences, regardless of the ML model used. However, this would incur a substantial load on the Monitoring component, since it would need to retrieve many samples at each loop for each service.

Another potential solution would be to cache the latest samples within the Attack Detector component, but this approach would make it stateful (i.e., bound to specific services). In addition, this would increase the complexity of managing replicas, making the scalability more complex.

The third approach is the one adopted by TeraFlowSDN. In this approach, a cache is deployed as an external component. The cache stores the latest samples of all the active services in the network. By adopting an in-memory cache, the response time can be orders of magnitude lower than the Monitoring component (which uses an in-disk persistent database).

Figure 72 illustrates the communication among components for the case where the Attack Inference uses a supervised learning model. For each service, the Attack Manager invokes the Attack Detector. The Attack Detector, responsible for a single service at a time, queries the Monitoring for the latest OPM sample(s), i.e., the ones that have not yet undergone the attack detection. The Monitoring returns the list of samples used to build a detection request sent to the Attack Inference. Once the inference result is received, the relevant KPIs are included in the time series related to this service. The Attack Detector then returns an empty message to the Attack Manager, representing that the assessment for this service has been completed. Finally, after receiving the completion message from all the services, the Attack Manager computes how much time the loop took and reports it to the Monitoring. This value is also reported to Prometheus.

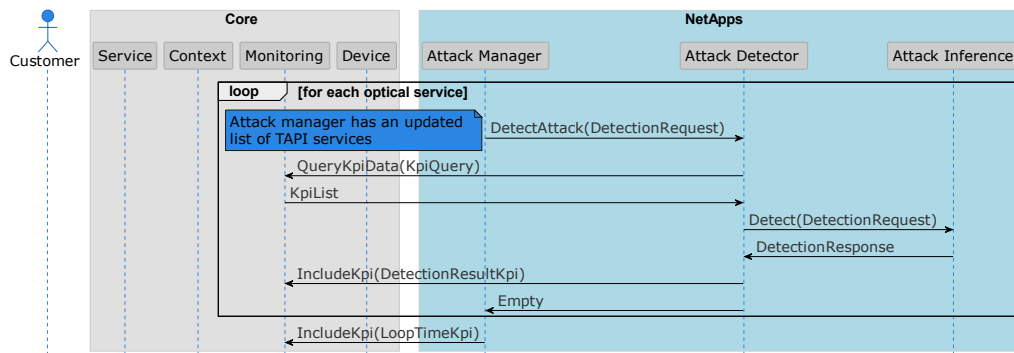


Figure 72. Scenario 3 workflow: Periodical optical cybersecurity monitoring using supervised learning

Figure 73 shows the communication among components for the case where an unsupervised learning algorithm is used by the Attack Inference. We focus only on the differences from the previous workflow. The first difference is that, upon receiving a detection request, the Attack Detector gets the latest samples from the cache. Then, based on the latest OPM sample available in the cache, it queries the new(est) OPM sample(s) from the monitoring component. The number of new samples may change depending on the ratio between the monitoring cycle and the cybersecurity cycle. For instance, if the monitoring cycle is executed at every 30 seconds, but the cybersecurity cycle runs at every 1 minute, each cybersecurity cycle will process 2 new OPM samples. The n new samples are added to the array of samples obtained from the cache, while discarding the oldest n samples. The new array of samples is sent back to the cache to be used in the next cycle.

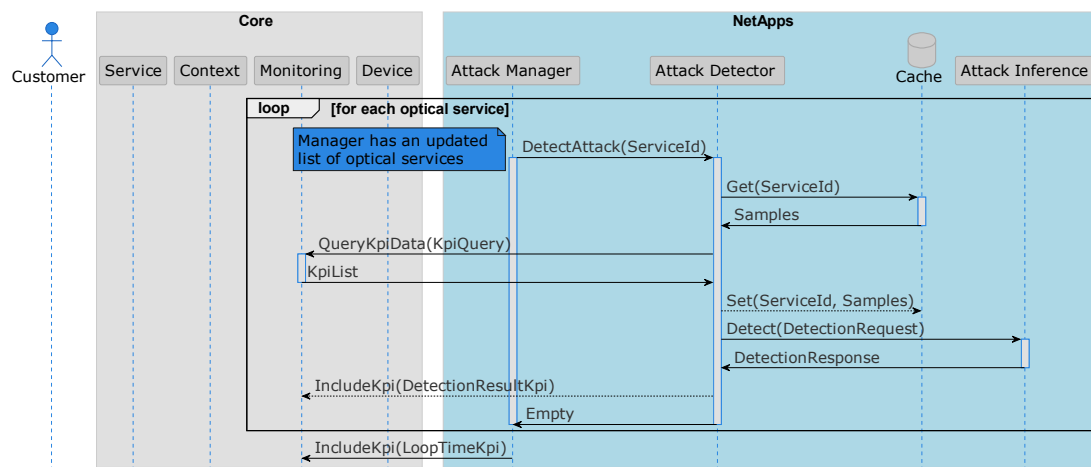


Figure 73. Scenario 3 workflow: Periodical optical cybersecurity monitoring using unsupervised learning

Once the needed samples are gathered, the Attack Detector composes a detection request to the Attack Inference and sends it. The Attack Inference executes the unsupervised learning model and returns the detection response with an array of integers with the same cardinality as the samples. Each item in the array represents whether or not that sample was considered an anomaly (i.e., an attack in our case).

Once an attack is detected, an attack mitigation strategy must be triggered. We leave the attack mitigation part for the next iteration of the Cybersecurity component to be reported in D5.3.

7.6. Preliminary Performance Evaluation

In this section, we report the preliminary performance of the Cybersecurity components for the two types of attacks investigated: Layer 3 attacks represented by cryptomining and optical physical layer attacks represented by six different attacks described in Section 7.3.2.

7.6.1. Layer 3

This section describes the performance and energy efficiency evaluation of the cybersecurity components that address attack detection and mitigation at Layer 3 of the seven-layered Internet model.

7.6.1.1. Security

In this section, the performance evaluation will focus on the machine learning model integrated in the Centralized Attack Detector responsible for identifying malicious cryptocurrency traffic in the network.

A major change from the initial design was the replacement of the Random Forest algorithm with a Deep Neural Network for detecting malicious cryptocurrency traffic. This decision was made because the Deep Neural Network (DNN) provided higher accuracy than the Random Forest. In addition, a DNN is easily parallelizable and can scale to adapt to different network environments with different data throughputs, allowing for better scalability than the Random Forest algorithm. In addition, another deciding factor for this change was the fact that the energy-efficient optimization of Random Forest algorithms is not yet well established, making them less suitable for the implementation of a centralized attack detector.

7.6.1.1.1. Analysis of the Cryptomining Detector

In this section, we describe in detail the model we used to address the detection of cryptomining attacks at the network layer. First, we describe the setup we used to collect the training data. Next, we present the structure of the model and the procedure that was followed to train it. Finally, we evaluate the model using several standard performance metrics.

7.6.1.1.2. Training of the Cryptomining Detector

The dataset used to train the DNN model for the task of cryptomining detection has been developed for the precise task of detecting cryptomining attacks [PAS20]. This dataset is provided by Telefónica R&D as part of ML research for defences against network traffic attacks generated in their Mouseworld lab.

The experiments that can be deployed in the Mouseworld Lab [PAS18] allow the capture, storage, and processing of network traffic representative of the attacks to be reproduced. The processing performed on the network traffic captured in the Mouseworld Lab is oriented toward the training and validation of ML models for detecting network attacks. To this end, the Mouseworld Lab provides a way to launch clients and servers and collect their traffic, even if they interact with clients and servers outside Mouseworld on the Internet. In this way, the Mouseworld lab can be used to set up and emulate attack scenarios in a controlled way and to generate and collect in a PCAP file all packets of the attack and normal traffic to be used later for the training and testing of ML algorithms. This emulation environment allows configuring and executing specific attacks mixed with normal traffic instantiating virtual machines that deploy specific attack clients connected to real servers located at different points on the Internet. In addition, the emulation environment allows configuring other

virtual machines on which normal traffic clients and servers (e.g., web, file hosting, streaming) are deployed. Finally, a commercial tool called BreakingPoint from Ixia allows a wide variety of realistic traffic types to be configured and injected into the network. Once a configuration is deployed, all packets exchanged by the clients and servers with each other and other servers on the Internet can be captured. The captures are stored in PCAP format files to be used later for training and validation of ML-based attack detectors.

We used the Mouseworld lab to emulate a cryptomining attack scenario over a 5G network. In this scenario, the attack consists of several cryptomining clients sending cryptomining traffic to a real server located on the Internet. Normal traffic was also injected into the network using the BreakingPoint tool, and several virtual machines were configured to emulate the normal traffic (e.g., web, file hosting, streaming). The captures of both attack and normal traffic were stored in PCAP files. The PCAP files were then used to generate the dataset used in this case study.

The data collected in the Mouseworld lab contains traffic samples represented by flow statistics derived from network packets using the Tstat tool. This traffic data was labelled to create the dataset used to train the cryptomining detector. In particular, two types of traffic can be found in the dataset, samples (rows) corresponding to normal traffic, and samples corresponding to cryptomining attacks. In this case, each row of the dataset was tagged as either 0 (normal traffic) or 1 (cryptomining attack traffic) using the IPs and ports of the known attack connections.

We used the TensorFlow library to train a Fully Connected Neural Network (FCNN) classifier to predict whether a connection corresponds to cryptomining activity or not according to all features derived from Tstat statistics except IPs and ports, as they are used to label the dataset (class labels) and therefore cannot be used to train the model.

The structure of each of the FCNN model that was used as baseline is specified below. In particular, the model consists of a stack of three fully connected layers with 20, 30 and 10 with Rectified Linear Unit (ReLU) activation followed by a fully connected layer with two neurons and SoftMax activation as output layer. The training hyperparameters are as follows. We use a batch size of 4096. We also use Adam optimizer with a learning rate of 0.001. Furthermore, we use the early stopping technique to automatically terminate the training process if the validation loss does not improve for 20 epochs, restoring the model weights to those obtained in the epoch with the lowest validation loss after training is complete. For the validation procedure, we reserve 20% of the training data for the validation split. Finally, as a loss function, we use the categorical cross-entropy function.

Although the accuracy of the model using all these features is already high, it was observed that many of them do not contribute significantly to the prediction performance and can be ignored to improve the training efficiency and model inference. Therefore, we decided to make a random selection of the most commonly used features and managed to reduce the required input to ten features, while the F1 score was still high (> 95%). We list the features that we selected in Table 7. Note that if a feature has a CS (Client-Server) and SC (Server-Client) identifier, it is because it has been measured in both directions. However, if a feature has only one identifier, it is because it has been measured in the direction indicated by the identifier type (CS or SC).

Table 7. Selected features of the Crypto dataset to train the cryptomining detector.

CS ID	SC ID	Name	Type	Description
13	27	SYN count	Numeric	Number of SYN segments observed (including rtx).
70	93	flow control	Numeric	Number of retransmitted segments to probe the receiver window.
71	94	unnece rtx RTO	Numeric	Number of unnecessary transmissions following a timeout expiration.
72	95	unnece rtx FR	Numeric	Number of unnecessary transmissions following a fast retransmit.
73	96	!= SYN seqno	Binary	1 = retransmitted SYN segments have different initial seqno.
74	97	HTTP Request count	Numeric	Number of HTTP Requests (GET/POST/HEAD) seen in the C2S direction (for HTTP connections).
76	98	First HTTP Response	Numeric	First HTTP Response code seen in the server->client communication (for HTTP connections).
77	99	PSH-separated C2S	Numeric	Number of push separated messages C2S.
78	100	PSH-separated S2C	Numeric	Number of push separated messages S2C.
-	90	reordering	Numeric	Number of packet reordering observed.

CS: Client to server traffic.

SC: Server to client traffic.

All data were standardized to ensure that the mean of the sample was 0 and the standard deviation was 1. This was done so that the scale of each variable did not cause one variable to dominate the results. We found that standardisation significantly improved the results.

7.6.1.1.3. Performance Evaluation of the Cryptomining Detector

The trained model was converted to an ONNX format. ONNX is an open-source format for representing deep learning models in an intermediate format that allows for interoperability between different frameworks, such as TensorFlow, PyTorch, and Caffe2. ONNX is a well-suited format for deploying deep learning models in production, since it enables faster performance and a smaller file size. In addition, this conversion process allows the model to be deployed and used in various environments, including web services and mobile devices, and a variety of hardware platforms. The conversion process of the DNN model from the TensorFlow/Keras format to ONNX was done using the tf2onnx library. Once the conversion is complete, the model can be deployed using the ONNX Runtime library, which provides an execution engine for the ONNX models.

Once the model was successfully converted to ONNX, it was evaluated in an offline fashion. This was done by first selecting a test set of data and then running inference on it using the ONNX Runtime library. The model's performance was then evaluated by comparing the predicted results to the actual labels of the data. Once the model was successfully converted to ONNX, it was evaluated offline. To do this, a test data set representing 20% of a reserved portion of the total data set that was never used for model training was first selected, and then inference was run on it using the ONNX Runtime library. The performance of the model was then evaluated by comparing the predicted results with

the actual labels on the data. The metrics used to measure model performance include three well-known metrics: precision, balanced accuracy, F1 score and confusion matrix. We incorporated balanced precision among the evaluation metrics to account for imbalances that exist in the data set. A brief explanation of each metric is provided below.

- **True Negative (TN):** number of cases in which the model correctly predicted a negative outcome. The True Negative Rate (TNR) measures the rate of negative outcomes correctly predicted as negative;
- **False Positive (FP):** number of cases in which the model incorrectly predicted a positive outcome. The False Positive Rate (FPR) measures the rate of negative samples that were mislabeled as positives;
- **False Negative (FN):** number of cases in which the model incorrectly predicted a negative outcome. The False Negative Rate (FNR) measures the rate of positive samples that were mislabeled as negative;
- **True Positive (TP):** number of cases in which the model correctly predicted a positive outcome. The True Positive Rate (TPR) measures the rate of positive samples that were correctly labeled as positive;
- **Accuracy:** rate of correct predictions made by the model. It is calculated by taking the ratio of true positives and true negatives to the total number of predictions. The formula is given by: $\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$;
- **Balanced Accuracy:** accuracy of the model in predicting both positive and negative classes. The formula is given by: $\text{Balanced Accuracy} = (TP/P + TN/N) / 2$ where P is the total number of positive examples, and N is the total number of negative examples;
- **Precision:** true positive rate of all positive predictions made by the model. The formula is as follows: $\text{Precision} = (TP) / (TP + FP)$;
- **Recall:** true positive rate of all true positive examples in the data set. The formula is as follows: $\text{Recall} = (TP) / (TP + FN)$;
- **F1 Score:** it is calculated by taking the harmonic mean of precision and recall. The formula is given by: $\text{F1 Score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$;
- **Confusion Matrix:** The confusion matrix is a visual representation of the model's performance and is used to analyze the model's ability to correctly classify the data into different classes.

The results of the evaluation are shown below.

- Accuracy: 0.99996
- Balanced Accuracy: 0.99543
- Precision: 0.99998
- Recall: 0.99543
- F1 score: 0.99541
- Confusion matrix:

	Predicted Negative Predicted Positive	
Actual Negative	97120	0
Actual Positive	4	434

From the evaluation results, it can be seen that the ONNX DNN model achieved excellent performance, with an accuracy of 0.99996, a balanced accuracy of 0.99543, a precision of 0.99998, a recall of 0.99543,

and an F1 score of 0.99541. This shows that the model is capable of accurately predicting the labels of the data set with a high degree of accuracy.

Once the offline evaluation was successfully completed, the ONNX model was integrated into the Centralized Attack Detector (CAD) component. In this context, we validated the performance of the model by re-injecting the same packets contained in the test dataset and observed that the performance of the ONNX model was consistent with the offline evaluation results.

7.6.1.2. Energy Efficiency

In this section, we evaluate the energy efficiency optimization of the deep neural network deployed in the Centralized Attack Detector component responsible for the cryptomining detection task. We present a comparison of the energy efficiency achieved with different state-of-the-art techniques, discuss the energy efficiency trade-offs arising from the model optimization, and identify the best performing approaches for the task at hand according to a variety of criteria considering different energy efficiency and accuracy requirements.

7.6.1.2.1. Experimental Framework for the Cryptomining Detector Energy Efficiency Optimization

In this section, we describe the experimental framework we applied for analysing the energy efficiency and resource utilization of DNN-based systems deployed in production environments. First, we explain the energy measurement process followed to collect energy consumption data and discuss the main statistics collected to analyse the resource utilization of the resulting models. Next, we describe how the most appropriate optimization strategy for the problem at hand is selected. Next, we describe the energy optimization techniques that were applied.

7.6.1.2.2. Measuring Energy Consumption

To measure the energy consumption obtained with the different optimization approaches, we use the Running Average Power Limit (RAPL) interface, which estimates power consumption based on the Power Management Controller (PMC) values that can be collected from Intel family processors. In particular, we use the RAPL interface through the *powerstat* command line profiling tool to collect the CPU power consumption of the ML models during the training, model optimization, inference, and model loading phases.

For each combination of techniques to be applied, the baseline model that was analysed in section 7.6.1.1.1 is trained and then the optimization techniques are applied sequentially depending on the order specified in the optimization strategy defined by the particular combination to be applied.

Once the model has been trained and optimized, its inference performance is evaluated. To evaluate the inference performance of the model obtained with each combination of techniques, the model is converted from TensorFlow/Keras to TensorFlow Lite format. Considering that the inference phase is the most energy-consuming in ML applications, converting the model to TensorFlow Lite format to speed up and optimize the inference process has proven to be a great gain in terms of both energy and performance.

In the inference stage, various batch sizes are tested to evaluate the variation in power consumption and resource utilization. By default, the small (32), medium (256) and large (1024) batch sizes are tested, although they can be configured by the user depending on the application requirements.

In addition, the predictive performance of the optimized ML model obtained with each combination of techniques is also measured. In this case, since the model to be optimized is a classifier, we use the F1 accuracy and F1 score, as well as the balanced accuracy to account for the class imbalance that exists in our data.

Once all repetitions have been performed, the metrics obtained at each time step among all repetitions are aggregated using the mean, standard deviation, and maximum value.

In addition, a second aggregation is also performed, but on this occasion on the time axis to show the mean value, standard deviation, and maximum of each statistic measured throughout the test. as a summary of the results. At this point, the total energy consumed in the test is obtained by multiplying the average energy consumption by the average duration of the test. Furthermore, the percentage of reduction in total average energy consumption is also calculated by computing the difference between the average total energy consumption of each test with that obtained for the same test performed with the baseline model and dividing the result by the average total energy consumption of the test performed with the baseline model. In this way, a percentage is obtained that can be used as a metric of the improvement in energy efficiency achieved by the proposed optimizations. The obtained value can be positive or negative depending on the change in the energy consumption of the optimizations with respect to the baseline. A positive value shows that the optimizations perform better than the baseline, while a negative value shows that the optimizations consume more energy than the baseline.

7.6.1.2.3. Selection of the Best Optimization Strategy

Three different optimization profiles have been considered in the optimization process, which determine the selection of the most suitable optimization technique according to the particular needs of the application. Details of these profiles are given in D4.2. A summary is given below.

- A. **Energy efficiency profile:** This profile is designed to minimize the power consumption of the optimized model as much as possible, while providing acceptable performance measured against a given metric and a specific threshold;
- B. **Performance profile:** This profile is designed to maximize the performance of the optimized model as much as possible, while providing an energy efficiency gain equal to or greater than a given threshold;
- C. **Balanced profile:** This profile is designed to balance performance and energy efficiency by applying the optimization strategy that provides the best balance between both metrics according to the parameters that control the importance of each one during the selection.

Regarding the selection of the most appropriate combination of techniques to apply, an exhaustive search of all combinations of optimization strategies is performed. More specifically, a set of all possible combinations of techniques that can be applied to the DNN model is first built and then each combination is applied and evaluated. At the end, the most appropriate combinations to apply according to each optimization criterion are selected.

7.6.1.2.4. Selected Model Optimization Strategies

Three different sets of optimization strategies were selected for the experimental evaluation. Each optimization strategy contains several different combinations of the most promising state-of-the-art optimization techniques that were identified. The tests were performed offline to evaluate the most effective approach to minimize the total energy consumption of an ML model during the training,

inference, and loading stages by performing a quantitative analysis of the energy consumption metrics.

The first set contains combinations of quantization techniques that can be applied as a post-processing step after training the ML model. The second set contains various methods of compressing the physical representation of an ML model (number of total model parameters or the in-memory size of each parameter), in order to reduce the amount of energy consumed by an ML model during the inference stage and in subsequent retraining that might be necessary due to a change in the underlying data distribution during the operational stage. Finally, the third set contains combinations of the individual techniques included in the other two sets.

The specific combinations of techniques in each set are listed in Table 8. It should be noted that, in order to reduce the computational cost and time spent on the third test set, only the optimal post-training quantization technique according to the results of the first test set in terms of the selected optimization profile is applied to the models of the third set. The reason for this choice is that all the post-training quantization techniques have a negligible computational cost compared to that of the techniques found in the second set. Therefore, by evaluating them beforehand and selecting the optimal one as the one used for the combinations present in the third test set, the total evaluation time is considerably reduced. After preliminary validation, we conclude that this approach does not affect the results obtained with the third set in any meaningful way. Specific details of the application of these techniques are described in D4.2.

Table 8. Energy efficiency optimization strategies considered in the experimental framework.

Set	Opt. Strategy Id.	Opt. Strategy
N/A	0	No optimizations (baseline)
Post-Training Optimization Techniques	1	1) Full 8-bit Integer (INT8) Weight Quantization
	2	1) Half-precision Floating-point (FP16) Weight Quantization
	3	1) Full Integer Weight Quantization with 16-bit Integer (INT16) Activations and 8-bit Integer (INT8) Weights
Training-aware Optimization Techniques	4	1) Pruning-aware Model Fine-tuning
	5	1) Quantization-aware Model Fine-tuning
	6	1) Neural Architecture Search 2) Knowledge Distillation
Combined Optimization Techniques	7	1) Pruning-aware Model Fine-tuning 2) Quantization-aware Model Fine-tuning
	8	1) Neural Architecture Search 2) Knowledge Distillation 3) Pruning-aware Model Fine-tuning
	9	1) Neural Architecture Search 2) Knowledge Distillation 3) Quantization-aware Model Fine-tuning
	10	1) Neural Architecture Search 2) Knowledge Distillation

		3) Pruning-aware Model Fine-tuning 4) Quantization-aware Model Fine-tuning
	11	1) Pruning-aware Model Fine-tuning 2) Optimal post-training Quantization
	12	1) Neural Architecture Search 2) Knowledge Distillation 3) Optimal post-training Quantization
	13	1) Neural Architecture Search 2) Knowledge Distillation 3) Pruning-aware Model Fine-tuning 4) Optimal post-training Quantization

7.6.1.2.5. Experimental Evaluation

In this section, we present the results of the experimental evaluation that was performed to optimize the energy efficiency of the postmining detector implemented in the Centralized Attack Detector component. First, we describe the experimental setup that we have defined to test the different optimization strategies to be evaluated, including the main parameters of the optimization process to be applied to our target model, as well as the hardware platform that we have used to perform the experiments. Next, we analyse the experimental results obtained in the model inference state for each of the optimization strategies that were applied. Finally, we provide a summary of the main conclusions of our experimental evaluation.

7.6.1.2.6. Experimental Setup

We have performed an experimental evaluation in which we have applied all combinations of optimization techniques defined in 7.6.1.2.4 to the cryptomining detector described in Section 7.3.1. In addition, we repeated the experiments 5 times with a 1-second time interval for sample measurements to collect energy efficiency metrics.

To carry out the optimization process, the three optimization profiles defined in 7.6.1.2.3 to select the most appropriate optimization strategy are considered. We establish as a performance threshold a minimum acceptable reduction in energy consumption concerning the non-optimized model of 25% and a minimum balanced accuracy of 0.9. Furthermore, to apply the balanced profile, we set the ratio of these two factors as 0.5 for both to obtain the optimization strategy that leads to the most balanced results between the two objectives and analyse its comparison with those obtained with the other two profiles. We will use balanced accuracy as the objective performance metric for our optimization process because, as explained above, due to the class imbalance that exists in our data it is a more restrictive and more reliable metric to evaluate the effectiveness of the different optimization strategies that have been applied than the other metrics that are commonly used a classification problem (e.g., accuracy, F1 score, etc.). Furthermore, we use a balanced accuracy of 0.9 as a threshold because it provides an acceptable accuracy in our case while maintaining adequate energy efficiency.

Three different batch sizes (small: 32, medium: 256 and large: 1024) were tested to analyse the influence of the batch size used to perform the prediction on the results obtained

The hardware platform used to validate the proposed methodology is a system with an Intel(R) Core(TM) i7-2600 CPU (Sandy Bridge microarchitecture; base clock 3.40 GHz; turbo boost 3.80GHz; 8

MB cache) with Intel RAPL support. The system has 32 GB of RAM and Ubuntu 20.04.5 (with 5.15.0-48-generic Linux kernel) was used as the operating system.

The experimental framework was implemented using Python (version 3.10.6) as the main programming language and using the following dependencies: TensorFlow (version 2.9.2), TensorFlow Model Optimization (version 0.7.3), psutil (version 5.9.4) and powerstat (version 0.02.27).

7.6.1.2.7. Analysis of the Results Obtained

The complete analysis of the results obtained in the inference stage of the optimized models and the selection of the best optimization strategies that provide the best compromise between energy efficiency and performance according to the different optimization profiles considered can be found in D4.2. In Figure 74, we show the percentage of total average CPU power consumption obtained for each optimization strategy during the inference phase. The values represented were obtained from the aggregation of measured values collected during the duration of model inference at 1-second intervals and over 5 iterations for each optimization strategy using a batch size of 256 (medium size) to perform the prediction.

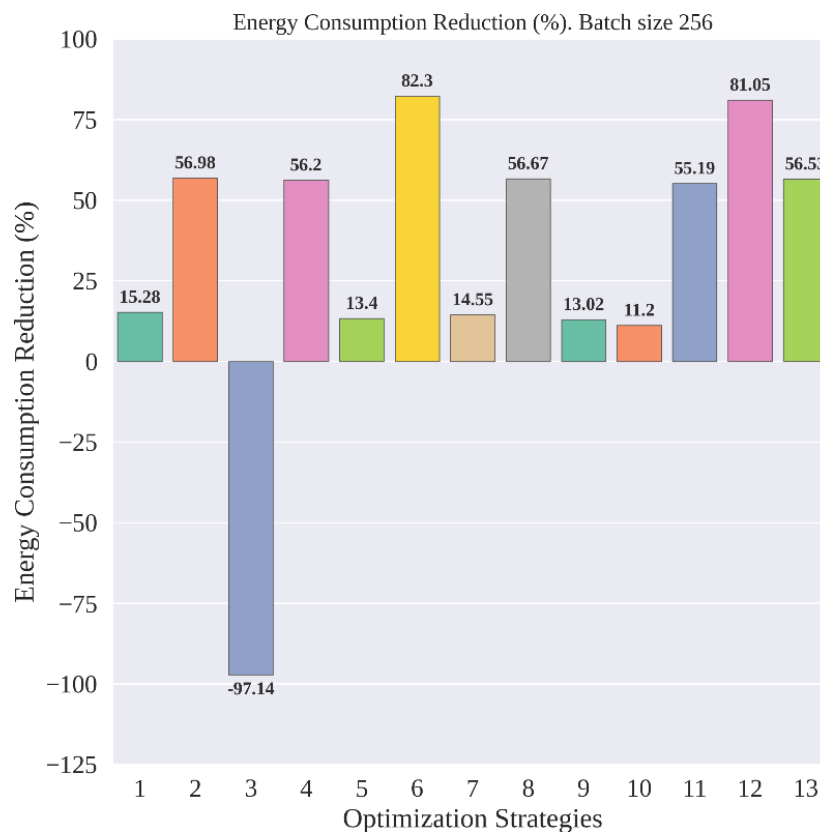


Figure 74. Energy consumption reduction obtained with each optimization strategy in the inference phase with respect to the non-optimized model using a batch size of 256.

A summary of the most important conclusions and observations is provided below.

- Almost all optimization strategies lead to a significant reduction in energy consumption, exceeding in most cases the threshold of reduction in energy consumption with respect to the non-optimized model that was set at the beginning of the experimental evaluation;

- The knowledge distillation technique provides the largest reduction in energy consumption in most cases studied, reducing the total average energy consumption by up to 82.304% with a minimal performance degradation of just 0.08% in the balanced accuracy, 0.016% in the accuracy and 0.11 in the F1 score;
- The optimization strategy based on the application of the half-precision floating-point weight quantization provides a reduction in energy consumption of up to 58.208%, with no performance degradation compared to the baseline model;
- Some of the techniques studied are not mutually exclusive and can be applied in conjunction with each other to further reduce the energy consumption of the model. In this regard, applying a weight quantization as final post-processing can potentially reduce the energy consumption of the model in the inference stage. In particular, we observed that the optimization strategy based on the application of the knowledge distillation technique followed by a half-precision floating-point weight quantization provides a reduction in energy consumption of up to 80.741%, with a negligible performance degradation of 0.08% in the balanced accuracy, 0.016% in the accuracy and 0.11 in the F1 score;
- Another technique that can be applied in conjunction with the knowledge distillation technique to further reduce the energy consumption of the model is the pruning-aware model fine-tuning. The optimization strategy based on the application of the knowledge distillation technique followed by a pruning-aware model fine-tuning provides a reduction in energy consumption of up to 81.046% but with a significantly higher performance degradation of 0.287% in the balanced accuracy. In addition, in some preliminary tests, we observed that performance degradation was unpredictable, as the same optimization strategy was applied on different occasions and provided different performance results. For this reason, we recommend caution when applying this optimization strategy;
- The results show that, regardless of the batch size used for inference, an optimal balance between energy and accuracy can be obtained with the application of the knowledge distillation technique, as it provides a very high energy savings with minimal degradation of performance. Finally, the results also demonstrate that, when performance degradation is not allowed, the optimization strategy based on the application of the half-precision floating-point weight quantization provides the best energy-consumption results;
- The difference in energy consumption reduction provided by optimization strategies using a large batch size and a small batch size does not result in a variation in the relative ranking of the optimization strategies. In most cases, optimization strategies that provide the greatest reduction in energy consumption for small batch sizes also provide the greatest reduction for large batch sizes, and when this is not the case, the difference in energy savings is small. However, it is clear from the results that the reduction in energy consumption obtained with the application of optimization strategies is greater for larger batch sizes, regardless of the optimization strategy applied. From this observation, we can conclude that, in order to obtain optimal energy savings, the use of large batch sizes should be preferred whenever possible, as the application requirements permit;
- The results obtained in terms of accuracy and balanced accuracy are also very favourable in general. With only two exceptions, the optimization strategies have managed to maintain a balanced accuracy above the threshold we set at the beginning of the experimental evaluation. In all cases, accuracy and the F1 score were almost unaffected. However, due to the significant class imbalance in the data, this result is irrelevant, so we continue to focus on

balanced accuracy. The largest balanced accuracy loss is obtained with the application of 8-bit integer weight quantization after training (0.5), followed by the balanced accuracy loss produced with the application of knowledge distillation and half-precision floating-point weight quantization (0.287) and knowledge distillation (0.008).

In summary, we observe that the optimization strategy based on the application of the knowledge distillation technique is the one that provides the largest reduction in energy consumption in all cases studied. However, in some cases, the optimization strategy based on the application of the knowledge distillation followed by the pruning-aware fine-tuning technique and the quantization-aware model fine-tuning provides a slight gain over the former but with a much greater performance penalty. In addition, the optimization strategy that provides the best energy efficiency gain with no degradation in performance is the half-precision floating-point weight quantization technique.

Based on the results obtained, we can conclude that the knowledge distillation technique provides the largest reduction in energy consumption in most cases studied, reducing the total average energy consumption by up to 82.304% with a minimal performance degradation of just 0.08% in the balanced accuracy, 0.016% in the accuracy and 0.11 in the F1 score.

In addition, it is clear from the results that the reduction in energy consumption obtained with the application of optimization strategies is greater for larger batch sizes, regardless of the optimization strategy applied. From this observation, we can conclude that, in order to obtain optimal energy savings, the use of a large batch size should be preferred whenever possible, as the application requirements permit.

Therefore, in our case the use of the knowledge distillation technique to optimize the DNN model implemented in the Centralized Attack Detector component is the most recommended strategy among the ones evaluated, as it provides the highest energy savings and minimal performance degradation. In addition, medium and large batch sizes should be used for inference, as they provide significantly higher energy savings than the small batch size.

7.6.2. Optical

In this section, we present the preliminary performance evaluation of the Cybersecurity component devoted to detecting physical layer attacks to optical networks. First, we present a quick summary of the results of the ML model for physical layer attack detection and identification (detailed results are present in D4.2). Then, we also show results related to the scalability properties of the Cybersecurity optical performance analysis loop designed and implemented in TeraFlowSDN.

7.6.2.1. Accuracy of ML models

One of the critical KPIs of the cybersecurity scenario is the accuracy of the ML model used in the presence of known and unknown attacks. Known attacks are attacks that have been included in the training dataset, therefore known by the ML model prior to the inference. Unknown attacks are attacks that have not been presented to the ML model during training, and are first presented to the ML model during inference. Naturally, both these attacks are known a priori by the ML/cybersecurity specialist who designed the experiments to collect the dataset, selected and trained the ML models, and evaluated their performance.

Let us focus first on the detection and classification of known attacks. In this case, the fact that the attacks are known by the ML model during training enable the ML model to classify them, i.e., the ML model is able to perform attack detection and identification. One of the most regarded ML models for

the classification task are the Artificial Neural Networks (ANNs). In our scenario, we adopted ANNs and performed a hyperparameter analysis (details are provided in D4.2). We split the dataset into 3 slices: training, validation, and testing.

Figure 75 shows the performance of the best ANN architecture over training for the training and validation datasets. As we can see, the accuracy and categorical cross-entropy (used as the loss function for training) start in a quite bad performance, but quickly progress to very good levels of performance. The accuracy reaches nearly 100%. We can see that after 400-500 epochs, the categorical cross-entropy of the validation set stabilizes, indicating that if we interrupt the training in around 500 epochs the model would still maintain the characteristics of the model trained with 1000 epochs. At the end, the ANN achieves a 98.2% accuracy over the testing set.

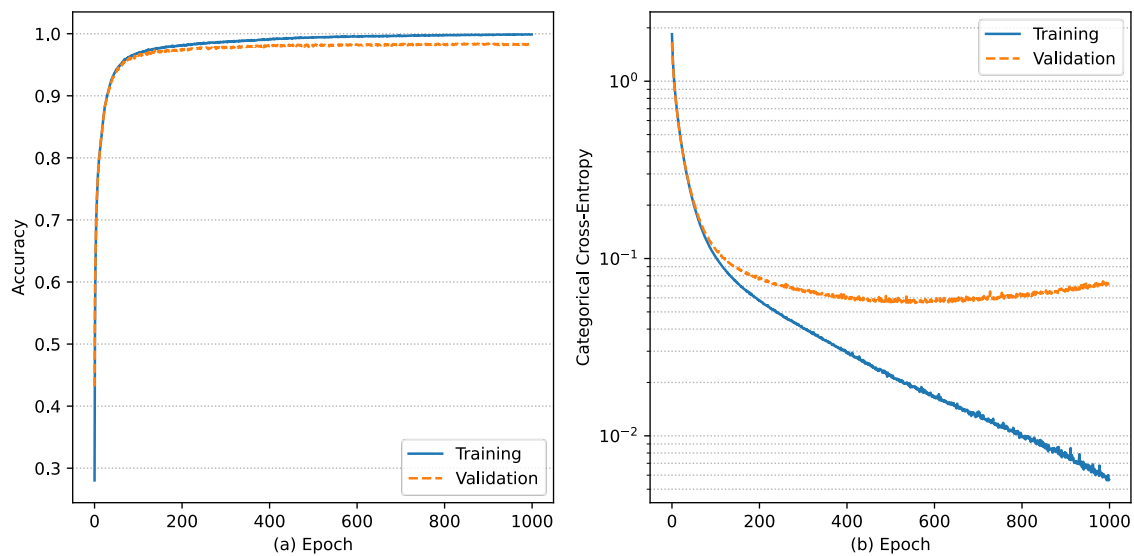


Figure 75. Training performance of the ANN for attack detection and identification

Figure 76 shows the confusion metrics for the training, validation, and testing data sets. In this case, true positives are all the attacks that are predicted as an attack, where false negatives are the attacks predicted as not attacks. Focusing on the testing dataset, only 0.4% are false negatives, where the true attack is a light in-band jamming attack, but the ANN is classifying them as normal operating conditions.

True negatives are the samples from normal operating conditions being classified as such, while false positives are the samples from normal operating conditions being classified as attacks. Only 0.2% of false positives are found, 0.1% being classified from the light in-band jamming attack, and 0.1% from the strong polarization attack.

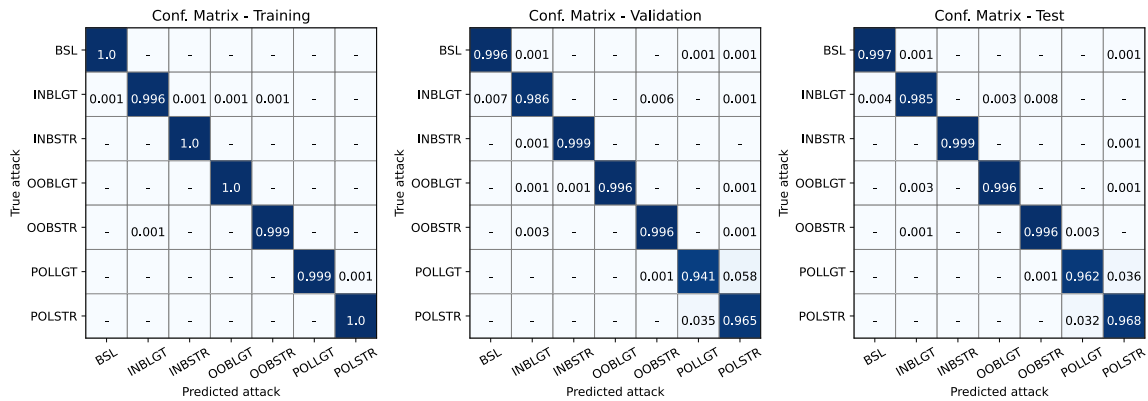


Figure 76. Confusion matrices for the ANN

When we move to the task of detecting unknown attacks, i.e., attacks that are not included in the training dataset, traditional ANNs are no longer suitable. In this case, we adopted DBSCAN, a well-known unsupervised learning algorithm used for anomaly detection. In our case, attacks should affect, at least mildly, the OPM parameters, just enough so that DBSCAN is able to detect them.

Table 9 shows a summary with the best results of the hyperparameter tuning performed for DBSCAN. The two parameters MinSamples and Epsilon were varied in the range of [3, 5, 8, 10, 12, 15, 20, 50, 80, 100] and [0.1, 0.5, 1, 2, 3, 4, 5, 10], respectively. We can see that the best configuration achieves an F1 score of 0.803, leading to 26% false positives and 13.8% false negatives. These numbers are not ideal but can be used in conjunction of root cause analysis strategies to provide indication that something is wrong with the channel.

Table 9. Summary of the results of unsupervised learning detecting unknown optical physical layer attacks

MinSamples	Epsilon	TNR	FPR	TPR	FNR	F1 score
3	1	0.819	0.18	0.76	0.239	0.77
5*	1*	0.733	0.266	0.861	0.138	0.803
8	1	0.601	0.398	0.912	0.087	0.786
10	2	0.998	0.0012	0.374	0.625	0.42
20	2	0.998	0.0017	0.55	0.449	0.623
50	2	0.991	0.008	0.586	0.413	0.659
80	2	0.891	0.108	0.653	0.346	0.687

* Configuration with the best overall F1 score.

7.6.2.2. Scalability Performance Evaluation

In this section, we focus on the performance evaluation of two components: the Attack Detector and the Attack Inference. The evaluation of the Attack Mitigator will be performed in the next deliverable (i.e., D5.3).

For the scalability performance evaluation, we designed a script that is able to generate a high number of optical service requests to TeraFlowSDN. Moreover, in order to be able to accommodate such a high number of services, we disabled the resource availability checks when performing the provisioning of new services. Finally, we configured the emulated optical data plane to replay data captured as detailed in Section 7.3.2. For the results in this deliverable, we only replay data from normal operating conditions, given that the scalability of the Attack Mitigator component is not assessed.

The results illustrated in this section are collected and plotted using Prometheus. We show the Prometheus query for each plot to serve as reference for reproducibility purposes. We varied the number of active optical services in the range [120, 240, 480, 960, 1440, 1920]. The OPM cycle is configured for 30 seconds, as well as the Cybersecurity assessment cycle (i.e., our components run the cycle at every 30 seconds). We use the Attack Inference component that leverages DBSCAN, and unsupervised learning algorithm that is able to detect anomalies (e.g., attacks in the case of our scenario). For each prediction, we use 330 samples. The cache is enabled and saves the 330 samples between monitoring loops.

First, let us visualize the number of active optical services in the network. Figure 77 shows the number of active optical services in the network, together with the Prometheus query used to generate it. For each number of services, we leave the experiments running for 30 minutes in order to be able to capture the stable scalability performance. Between each test, we remove all the active services and leave TeraFlowSDN without any services for 10 minutes.

`optical_security_active_services`

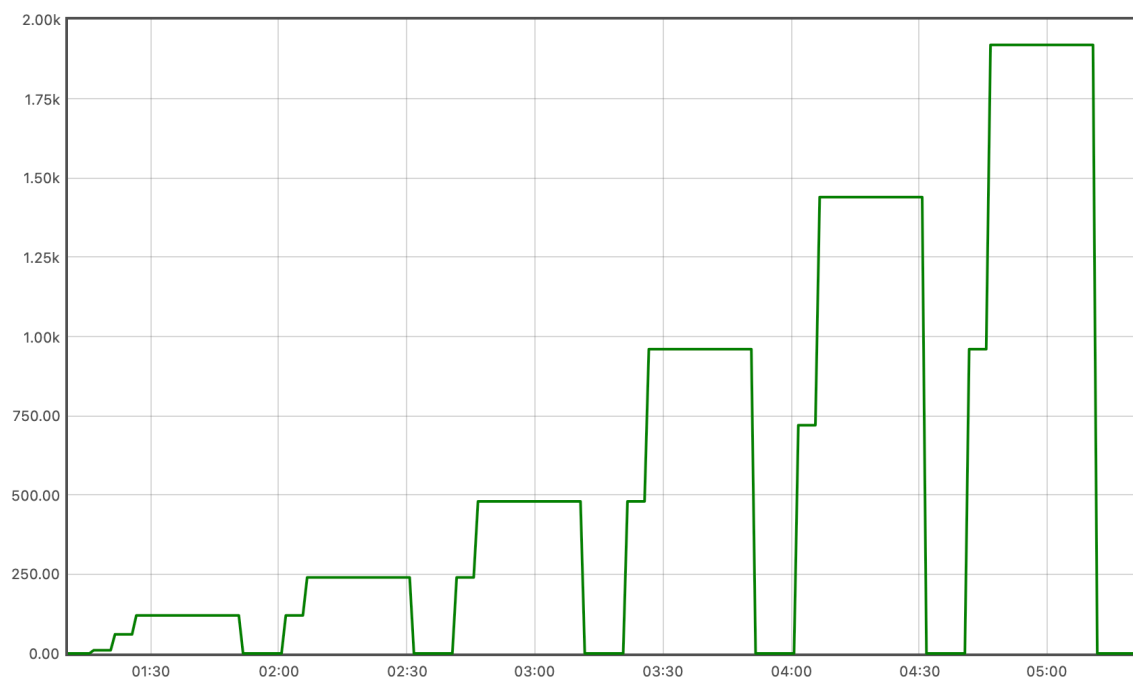


Figure 77. Number of active optical services (y axis) over time (x axis) in the network as collected by Prometheus

Next, we move our attention to evaluating how successful the Cybersecurity component is with respect to scaling to meet the monitoring cycle (i.e., configured for 30 seconds in this case). Figure 78 shows the Prometheus query and the plot generated for the average loop time. By analyzing Figure 78 in combination with Figure 77 we can note that the increases in the number of services coincide with increases in the loop time (i.e., x axis have the same time span). This represents the time that it took to perform the optical physical layer attack detection over all the services in the network. Note that the plot shows that at the beginning of a given experiment (e.g., right after adding 240 services) there is a increase in the response time, that later stabilizes towards the final value. This behaviour is explained by the fact that right after adding a high number of services, all at once, it takes a few minutes for TeraFlowSDN to scale to the necessary number of replicas. However, this represents a worst-case scenario, as in normal operating conditions the number of active services does not fluctuate so drastically in a short period of time.

We can see that between 120 and 960 services, the loop took between 10 and 15 seconds. Note that this time can be considered quite stable, even though the number of services in the network increased by a factor of 8. When the number of services is 1440, the loop takes longer than 20 seconds. When considering 1920 active optical services, the loop time reaches 30 seconds at first, but quickly stabilizes below that value.

```
rate(optical_security_loop_seconds_sum[5m]) /  
rate(optical_security_loop_seconds_count[5m])
```

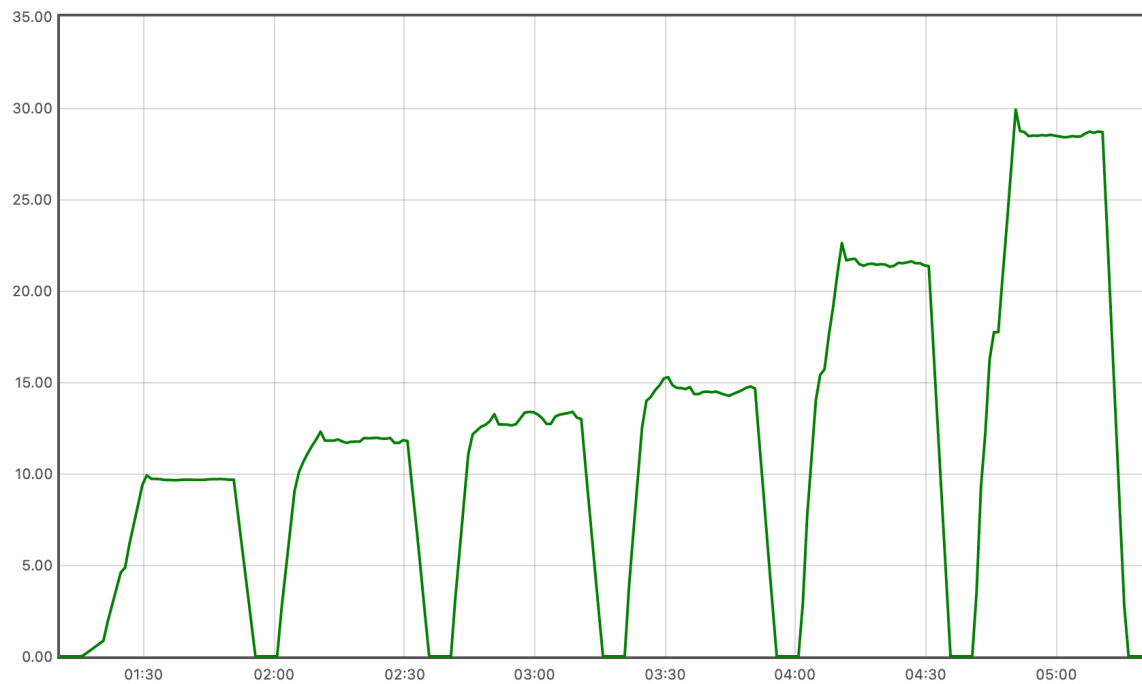


Figure 78. Time taken for the optical cybersecurity monitoring loop (y axis, in seconds) over time (x axis) as collected by Prometheus

In the following, we focus on the scalability properties of each individual component. Figure 79 shows the response time of the Attack Detector for performing the cybersecurity monitoring to a single service. The value is averaged over all the replicas. We can see that at the beginning of each experiment (i.e., right after adding a high number of services), the response time is high, reaching up to 230 milliseconds for the case with 1920 services. However, as the experiment progresses, more replicas are created and the response time stabilizes around a value between 80 and 120 milliseconds, which represents a very stable range given the great difference in the number of services.

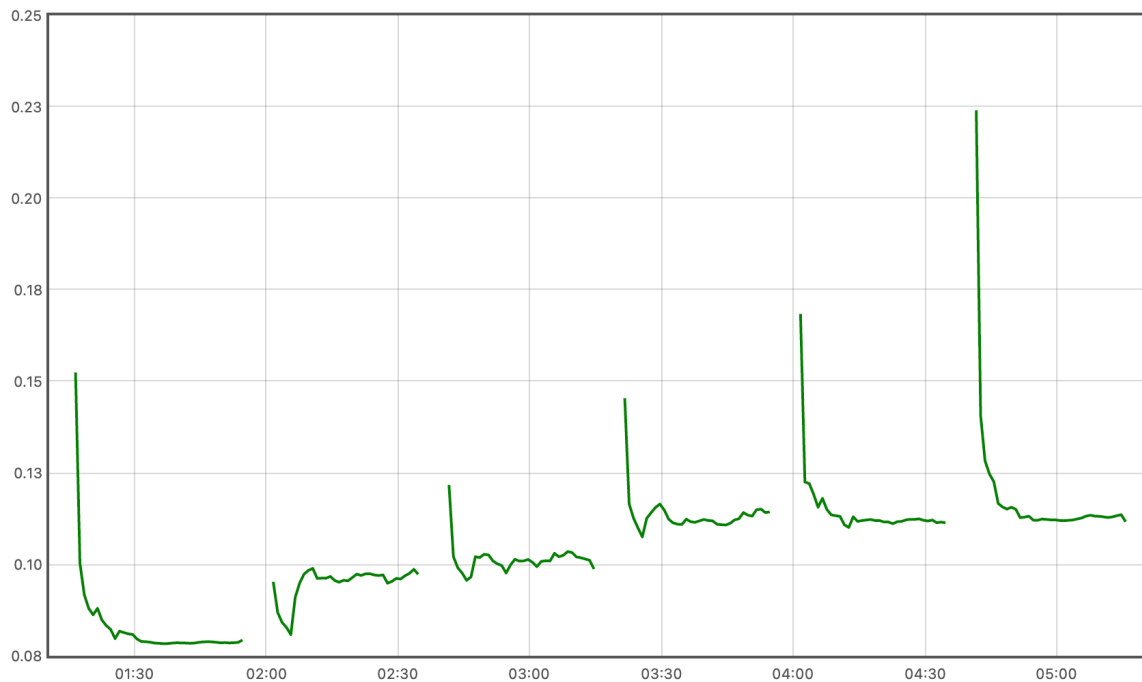


Figure 79. Average response time over all replicas (y axis, in seconds) of the optical attack detector over time (x axis) as measured by Prometheus

7.7. Pending Work and Summary

Table 10 summarizes the KPIs and KVs achieved by the current implementation of Scenario 3. Some of the KPIs and KVs are left for D5.3.

Table 10. Target and achieved KPIs and KVs for Scenario 3

KPI	Target	Validation results	
		Layer 3	Optical
Security	> 99% accuracy (known attacks)	<ul style="list-style-type: none"> Accuracy Score: 0.99966 False positive: 0 False negative: 144 True positive: 1494 True negative: 421968 F1 Score: 0.95402 	<ul style="list-style-type: none"> Accuracy Score: 0.982 False positive: 0.002 False negative: 0.004 True positive: 0.996 True negative: 0.997 F1 Score: 0.996
	> 90% accuracy (unseen attacks)	N/A	<ul style="list-style-type: none"> Accuracy: 0.817 False positive: 0.266 False negative: 0.138 True positive: 0.861

			<ul style="list-style-type: none"> • True negative: 0.773 • F1 Score: 0.803
	> 30% reduction of attack response latency	To be evaluated in D5.3	N/A
Reliability	> 90% accuracy in detecting and avoiding known adversarial attacks.	To be evaluated in D5.3	N/A
Energy	> 25% resource consumption	<ul style="list-style-type: none"> • Percentage of Total Average CPU Energy Consumption Reduction with respect to the original model in the inference stage (Knowledge Distillation, batch size: 256): 82.304%. • Loss in the accuracy: 0.016%. • Loss in the balanced accuracy: 0.008%. • Loss in the F1 Score: 0.011 	To be evaluated in D5.3

8. Conclusions and Next Steps

This deliverable reports the latest efforts on integration and performance evaluation of TeraFlowSDN. In this regard, three scenarios are leveraged to drive the integration and evaluation efforts. The main outcomes are a functional CI/CD environment and detailed documentation for new and experienced users. Moreover, a new metrics collection framework enables the internal monitoring and performance assessment of TeraFlowSDN components and workflows. Finally, each scenario has been detailed, including its motivation and challenges, alignment with the overall TeraFlowSDN architecture, the setup used to evaluate the performance in the context of the scenario, metrics relevant to the scenario, workflows, deployment, and preliminary performance evaluation.

Regarding integration, the next steps include the functional tests created for the scenarios in the CI/CD environment. This will enable the validation of modifications in terms of unitary tests (i.e., tests more focused on the individual functionalities of each component) and end-to-end workflows. Activities for dissemination will continue, and feedback will be considered for newer versions of the documentation offered by TeraFlowSDN.

The scenarios will continue to be integrated and evaluated, with several points to be reported in D5.3. For Scenario 1, all the KPIs and KVs have been detailed in this deliverable. However, due to issues in the integration of components, all measurements of the defined KPIs and KVs will be reported in D5.3. The same applies to Scenario 2, where the integration issues with the Context component prevented us from obtaining the preliminary performance results as expected.

Finally, in the case of Scenario 3, several KPIs and KVs have already been measured and reported in this deliverable. However, additional performance assessments remain to be performed. In particular, with respect to the energy efficiency optimization performed in Cybersecurity Layer 3, the best optimized model that was obtained in the energy efficiency assessment that was performed will be deployed in the Centralized Attack Detector component in the next iteration, and further energy efficiency and performance metrics will be collected in this environment to validate that they are consistent with the energy efficiency and performance metrics that were obtained offline. Furthermore, resilience to adverse attacks will also be analysed in the next iteration. In this sense, we will report the methodology and results of our study on the effectiveness of the defensive mechanisms put in place to protect the ML model deployed in the Centralized Attack Detector against this type of attack. In the case of Cybersecurity Layer 3 components, a comprehensive evaluation of the real-time performance of the different components under various stress conditions will be carried out, and different scalability mechanisms will be studied to ensure the correct operation of these components under high load. In the future, we will also analyse the attack mitigation response time of Cybersecurity L3 components in the next iteration and study different strategies to improve the performance of these services by reducing latency in inter-component communications. For optical cybersecurity, the next deliverable will include a measurement of the elapsed time from detection to end-to-end mitigation, as well as an analysis of the performance of the attack mitigation strategy. Finally, an assessment of the power consumption of the optical cybersecurity loop will be included, and optimizations will be proposed.

References

- [ECOC22] Ll. Gifre, et al, "Experimental Demonstration of Transport Network Slicing with SLA Using the TeraFlowSDN Controller", European Conference and Exhibition on Optical Communication, 2022.
- [EDG22] '300G CELL SITE ROUTER' [Online]. Accessed: 2022-12-15. Available: <https://www.edge-core.com/productsInfo.php?cls=291&cls2=342&cls3=343&id=955>
- [JLT2019] C. Natalino, et al., "Experimental Study of Machine-Learning-Based Detection and Identification of Physical-Layer Attacks in Optical Networks," in *Journal of Lightwave Technology*, vol. 37, no. 16, pp. 4173-4182, Aug. 2019. DOI: 10.1109/JLT.2019.2923558.
- [NFV22] Ll. Gifre, et al., "DLT-based End-to-end Inter-domain Transport Network Slice with SLA Management Using Cloud-based SDN Controllers", IEEE NFV-SDN, 2022.
- [OECC22] R. Vilalta, et al., "End-to-end Interdomain Transport Network Slice Management Using Cloud-based SDN Controllers ", OECC/PSC 2022.
- [OFC22] Ll. Gifre, et al., "Demonstration of Zero-touch Device and L3-VPN Service Management using the TeraFlow Cloud-native SDN Controller", OFC, 2022.
- [OFC23] Ll. Gifre, et al., "Slice Grouping for Transport Network Slices Using Hierarchical Multi-domain SDN Controllers", accepted demo paper at OFC, 2023.
- [PAS18] A. Pastor, A. Mozo, D. R. Lopez, J. Folgueira, and A. Kapodistria, 'The Mouseworld, a security traffic analysis lab based on NFV/SDN', in *Proceedings of the 13th international conference on availability, reliability and security*, 2018, pp. 1–6.
- [PAS20] A. Pastor *et al.*, 'Detection of encrypted cryptomining malware connections with machine and deep learning', *IEEE Access*, vol. 8, pp. 158036–158055, 2020.
- [SLK] ETSI TeraFlowSDN Slack channel. Subscription link: https://join.slack.com/t/teraflowsdn/shared_invite/zt-1lut4qg47-9XbNW4S3egOw7UYf6D3sdQ
- [SPI22] 'Spirent SPT-N12U Mainframe Chassis'. [Online]. Accessed: 2022-12-15. Available: [Spirent SPT-N12U Mainframe Chassis datasheet - Spirent](#)
- [RFC8466] G. Fioccola, et al., A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery, IETF RFC 8466, October, 2018.