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

Secured autonomic traffic management for a Tera of SDN flows

D5.3: Final demonstrators and evaluation report

Deliverable type	R (Report)
Dissemination level	PU (Public)
Due date	30.06.2023
Submission date	07.07.2023
Lead editor	Carlos Natalino (CHAL)
Authors	Lluis 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 Diaz, Óscar González de Dios (TID), Georgios P. Katsikas (UBI), Jose Juan Pedreño, Achim Autenrieth (ADVA), Sergio González, Javier Moreno (ATOS), Carlos Natalino (CHAL), Sebastien Andreina (NEC), Min Xie, Jane Frances Pajo, Abdelhakim Cherifi, Håkon Lønsethagen (Telenor), Mika Silvola (Infinera), Michele Milano, Nicola Carapellese (SIAE), Sébastien Merle, Peer Stritzinger (Stritzinger), Thomas Zinner (NTNU)
Reviewers	Paolo Monti (CHAL), Ricard Vilalta (CTTC)
Quality check team	Daniel King (ODC)
Work package	WP5

Abstract

This deliverable presents the performance assessment work done with TeraFlowSDN release 2.1 in the three scenarios considered in the project, i.e., Autonomous Network Beyond 5G, Inter-domain, and Cybersecurity. Four key aspects are reported: the processes adopted for the integration of TeraFlowSDN; the integration and adoption of the metrics collection framework; refinements of the scenario descriptions and workflows; and finally, the performance assessment. The document provides an overview of the Release 2.1 Architecture, details the metrics collection framework, and discusses the relevant KPIs. We present a brief context and motivation for each scenario, followed by the alignment with the TeraFlowSDN architecture, performance assessment, and scenario conclusions. The deliverable concludes with a summary of the KVIs and KPIs achieved and final remarks.

[End of abstract]

Disclaimer

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	24.01.2023	Editor	Initial structure of document
0.2	25.04.2023	Ricard Vilalta	Improvements to the structure of the document
0.2.1	26.04.2023	Georgios P. Katsikas	Contributions to the content
0.2.2	02.05.2023	Thomas Zinner	Contributions to the content
0.2.3	03.05.2023	Editor	Contributions to the content
0.2.4	04.05.2023	Ricard Vilalta	Contributions to the content
0.2.5	09.05.2023	Amit K. Batra	Contributions to the content
0.2.6	09.05.2023	Lluis Gifre	Contributions to the content
0.3	09.06.2023	Alberto Mozo	Contributions to the content
0.4	14.06.2023	Ricard Vilalta	Contributions to the content
0.4.1	19.06.2023	Lluis Gifre	Contributions to the content
0.4.2	22.06.2023	Editor	Contributions to the content
0.4.3	23.06.2023	Sébastien Merle	Contributions to the content
0.5	26.06.2023	Editor	Contributions to the content
0.6	04.06.2023	Reviewers	Reviewed version
0.7	07.07.2023	Quality Check	
1.0	07.07.2023	Editor	Submission

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

EXECUTIVE SUMMARY

This deliverable reports the achievements of WP5 during the last six months of the project, focusing on the performance assessment of TeraFlowSDN. The objective of this document is to summarize the results achieved in *(i)* processes adopted by TeraFlowSDN integration, *(ii)* integrating and adopting the metrics collection framework for performance assessment, *(iii)* refining the scenario descriptions and workflows, and *(iv)* collecting and summarizing the performance assessment results. The latter refers to the collection and analysis of KPIs and KVIs that 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, adopted by all the scenarios later described in the document. The third section offers an overview of the metrics collection framework for TeraFlowSDN, consolidating the performance assessment of all the components in a single solution and enabling in-depth scalability and performance analysis. Section 3 also presents the definitions of the metrics adopted in the performance evaluation, and the relevant scenario(s) where they are evaluated.

The second half of the document includes sections 4, 5 and 6. These sections are devoted to the three scenarios used to evaluate TeraFlowSDN. Each one contains a short introduction to the scenario's motivation and challenges. The alignment with TeraFlow architecture specifies how a scenario utilizes TeraFlowSDN and which components and use cases are relevant. For each scenario, a performance assessment section contains the description of the infrastructure adopted for the measurements, followed by the workflows that have been tested. Each workflow details its step-by-step performance assessment methodology, followed by the results obtained. We summarize each scenario introduction and its conclusions. Finally, this deliverable provides a summary of all the KVIs and KPIs achieved and final remarks.

Autonomous Network Beyond 5G

The TeraFlowSDN controller supports operator-driven use cases and workflows, addressing the objectives of this scenario by enabling the programmability of network elements and technology-based SDN controllers with the necessary north-bound and south-bound interfaces.

KPI	Target	Validation results
Device on-boarding time	< 50ms	100-400 ms. The target was too optimistic, but it does not have a real impact since on-boarding is performed only during the initialization phase. The current on-boarding procedure implies multiple interactions with the underlying database. In the next releases, we plan to optimize those interactions further to reduce the onboarding time.
Service setup delay	< 50ms	The measured overhead using emulated devices is 100ms. This deviation is detailed at the end of this section.
Service teardown delay	< 50ms	The measured overhead using emulated devices is 90ms. Similarly, as with service setup delay, in future releases, we plan to review the overall Service teardown workflow to improve the internal finite state machine, identify unneeded database interactions, and add support for parallel device deconfiguration.

Data rate	100G	Data rate is able to be shown in Grafana. Available data rates are dependent on the topology and network equipment, so we are limited to the transponders available in whiteboxes (e.g., 100G).
End-to-end service latency	5ms (indicative)	The actual value depends on the topology setup. For example, hardware switches are faster than software switches, while software switches perform better on high-end Commercial off-the-Shelf (COTS) hardware. Therefore, this value may vary. 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.
Reaction time to ensure SLA	~4s	The reaction time can be further analyzed on RPC calls (~1s) and path recalculation time (~3s). It should also be noted that the path recalculation time is dependent on the topology. Larger topologies may require more time than smaller topologies to find new paths.
Resource efficiency reduction factor	2	L2VPN is of 2,32 at offered load of 5k Erlang (peak resource efficiency). L3VPN is of 6,4 at offered load of 7k Erlang (peak resource efficiency).
Energy	< 30%	The devised EAR relies on a heuristic which favours the routing through active network elements rather than powered up devices and/or ports as much as possible. The conducted study paves the way to continuing working in the TeraFlow SDN controller to become a controller which adopts energy-efficiency objectives. In this regard, the next steps are envisaged to tackle more advanced algorithms, such as re-allocation of the established services to enforce powering-down network elements and/or exploiting the benefits of using machine learning trained models to attain better trade-offs between service provisioning and energy reduction.

Inter-domain

When different domains belong to separate network operators, mechanisms for inter-domain slicing while maintaining the privacy of internal network details become essential. The TeraFlowSDN controller incorporates a Distributed Ledger Technology (DLT) component based on blockchain technologies. This ensures that data exchanged between per-domain TeraFlowSDN instances can be kept confidential, if required, while enabling inter-operator collaboration.

KPI	Target	Validation results
Multi-tenancy	> 100 tenants	TeraFlowSDN can support more >100 tenants with reasonable service latencies. The response time might increase at high loads due to database latency and SQL query contention. Future releases might study how to enhancement the database schema.
Trust/privacy	100% secured connections	All connections related to the DLT component are secured and authenticated.

DLT transaction delay	10s	The average latency is between 2.2 and 3.3 seconds, c.f. Figure 75.
Positioning	100% vehicles	We have validated location-awareness in end-to-end connectivity services and network topologies. TeraFlowSDN has been extended with an augmented data model for topology and connectivity services to include GPS coordinates and Regions into service endpoints and connectivity service constraints. The proposed architecture considers the requested end-to-end connectivity service provisioning and update, considering that location-aware connectivity services might need service endpoint migration due to the dynamic nature of the joint edge-cloud continuum.
Social	< 20% cost	Green path computation (gPC) policies are proposed with a reward system, allowing greener states to correspond to higher rewards for lower performance. Considering that some green states might be unavailable (due to the aforementioned trade-off) for a given SLA requirement and link utilization at certain time intervals, introducing DLAs enables the customer to unlock greener states by allowing a certain level of performance degradation. By unlocking greener states with allowable degradation levels, the savings increased by around 47% with respect to **gPC$_{SLA}$**.

Cybersecurity

We developed a Cybersecurity module to address the broad range of cybersecurity threats present in this scenario. The Cybersecurity scenario has validated several components and use cases. The Cybersecurity module within TeraFlowSDN comprises three components: *Centralized Attack Detector (CAD)*, *Attack Inference*, and *Attack Mitigator*.

KPI	**Target**	**Validation results**	
		Layer 3	**Optical**
Security	> 99% accuracy (known attacks)	- Accuracy Score: 0.99966 - False positive: 0 - False negative: 144 - True positive: 1494 - True negative: 421968 - F1 Score: 0.95402	- Accuracy Score: 0.996 - F1 Score: 0.996
	> 90% accuracy (unseen attacks)	N/A	- Accuracy Score: 0.99 - False Positive Rate: 0.00063
Reliability	> 90% accuracy in detecting and avoiding known adversarial attacks.	- 99% accuracy in detecting and avoiding known adversarial attacks (i.e., 1% evasion rate).	N/A

Energy	> 25% resource consumption	- 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	N/A
Scalability	< 20% increase in the mean loop time between low and high traffic load	The mean loop time varies 15.7% (between 0.0247 and 0.0293) between low and high traffic load conditions.	N/A
	< 5 minutes violation of the targeted monitoring cycle time	N/A	The module is able to stabilize from a drastic increase in the number of services (in the order of 10x) in less than 5 minutes.

Table of contents

List of Figures

List of Tables

Abbreviations

AE	Adversarial Example
AI	Artificial Intelligence
API	Application Programming Interface
B5G	Beyond 5G
BBR	Blocked Bandwidth Ratio
BSS	Business Support System
CCAM	Cooperative, Connected, and Automated Mobility
CDF	Cumulative Distribution Function
CE	Customer Edge
CLI	Command-Line Interface
COTS	Commercial off-the-Shelf
CSGW	Cell-Site Gateways
DC	DataCenter
DLT	Distributed Ledger Technology
DNN	Deep Neural Network
DPI	Deep Packet Inspection
E2E	End-to-End
GAN	Generative Adversarial Network
gNMI	gRPC Network Management Interface
gRPC	gRPC Remote Procedure Call
HPA	Horizontal Pod Autoscaler
HT	Holding Time
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPM	Integrated Performance Monitoring
L2NM	Layer 2 Network Model
L2VPN	Layer 2 Virtual Private Network
L3	Layer 3 (Packet)
L3NM	Layer 3 Network Model

L3VPN	Layer 3 Virtual Private Network
ML	Machine Learning
MOS	Mean Opinion Score
MPLS	Multiprotocol Label Switching
MTU	Maximum Transfer Unit
MW	MicroWave
NBI	North-Bound Interfaces
NFV	Network Function Virtualization
NS	Network Service
OLS	Optical Line System
ONF	Open Networking Foundation
OPM	Optical Performance Monitoring
OSS	Operational Support System
OTA	Over-the-Air
PE	Provider Edge
PCEP	Path Computation Element Protocol
PINS	Public interconnected networks and services
PoP	Point of Presence
RPC	Remote Procedure Call
RTT	Round Trip Time
SASE	Secure Access Service Edge
SBI	South-Bound Interfaces
SDN	Software-Defined Networking
SLA	Service Level Agreements
SLE	Service Level Expectation
SLO	Service Level Objective
SNI	Server Name Indication
SSL	Secure Socket Layer
TAPI	Transport API
TCAM	Ternary Content Addressable Memory
TLS	Transport Layer Security

TR	Technical Reference
TSTAT	TCP STatistic and Analysis Tool
VIM	Virtual Infrastructure Manager
VPN	Virtual Private Network
WAN	Wide Area Network
WIM	WAN Infrastructure Manager

1. Introduction

TeraFlowSDN delivers a state-of-the-art, open-source, cloud-native Software Defined Networking (SDN) controller. TeraFlowSDN provides scalable, efficient, reliable, and flexible control for B5G (Beyond 5G) networks. In version 2.1, these characteristics are extended by providing a performance-validated SDN solution with integration and deployment procedure improvements. In this context, it is crucial to ensure that TeraFlowSDN correctly and efficiently integrates with networking devices and can handle the different protocols necessary to do so. WP5 is the work package that performs the TeraFlowSDN integration, followed by experimentation, validation, and evaluation using a range of Key Value Items (KVIs) and Key Performance Indicators (KPIs).

Given the distributed nature of the TeraFlowSDN development process, WP5 enumerated, evaluated and selected suitable techniques, processes, and tools used to assist partners during the development of TeraFlowSDN components and scenarios. This setup supported the collaboration among all partners, while ensuring consistency and reliability of the resulting software. The supporting software infrastructure developed during the H2020 TeraFlow project will continue to be leveraged within the ETSI OSG TeraFlowSDN.

To build the testbed setups, the project leveraged and extended existing infrastructure at partners' premises, realizing the three scenarios described in this deliverable. The scenarios are shortly described, highlighting their context and motivation. Then, details of the scenario related to the setup, metrics, workflows, deployments, and performance assessment are presented.

1.1. Purpose

The purpose of D5.3 is threefold. First, we aim to describe version 2.1 of the TeraFlowSDN controller, summarizing the overall architecture while highlighting specific developments made since version 2.0. The second objective is to report the performance assessment carried out in the three scenarios considered so far in the project. Each scenario is shortly described, followed by the testbed setup, workflows, and collection/analysis of the results. Finally, this deliverable summarises all the measurements across all the scenarios, consolidating the KVIs and KPIs achieved in the project.

1.2. Relationship with other Deliverables

This deliverable builds upon D5.2, where KVIs, KPIs and testbeds were initially described. To avoid repetition of content, when convenient, while maintaining consistency, we refer to D5.2 in some parts of this deliverable. Moreover, some of the performance assessment works reported in this deliverable are extensions of the initial results reported in D3.2 and D4.2.

1.3. Structure

This deliverable is structured as follows. Section 2 presents an architectural overview of TeraFlowSDN, comprising the latest components and interfaces added in version 2.1. Section 3 provides an overview of the metrics collection framework and defines the metrics reported in this deliverable. Sections 4, 5, and 6, report the performance assessment of each of the scenarios, respectively. In each section, a summary of the context, motivation, and challenges is presented, followed by the performance assessment of each workflow, concluded with some final remarks. Section 7 presents a summary of all the KVIs and KPIs measurements reported in the deliverable, followed by a few final remarks related to the activities of WP5.

2. Architecture Overview

A detailed description of the TeraFlowSDN release 2.0 architecture was provided in D2.2. Even though no major architectural changes were introduced in version 2.1, we briefly describe the overall TeraFlowSDN architecture to make the deliverable self-contained.

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

Figure 1 shows the proposed micro-service-based architecture. Following the design principles of cloud-native applications, each component is implemented as a micro-service that exports a set of Remote Procedure Call (RPC) services to other components. Each micro-service can be instantiated once or with multiple replicas, allowing 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 to receive all the service requests. 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 addition/deletion and updating its internal list of replicas. Depending on the RPC implementation, we may use the built-in Kubernetes load-balancer or adopt an external one (e.g., the one reported in Section 3.1). Each replica comprises a Pod, i.e., a collection of containers that the Kubernetes platform manages as a single entity. More information is provided in Sec. 3.1.

Context is a stateful component. It stores the network configuration (e.g., topologies, devices, links, services) and the status of all elements managed by TeraFlowSDN. Context leverages a database management system optimized for cloud-native scenarios, providing replication and high-availability deployments. Internally, it implements a Database API allowing for seamless switching between different backends (i.e., database management systems). The *Service* component is responsible for selecting, configuring, and deploying the requested connectivity service through the *South-Bound Interface (SBI)* component. To this end, *SBI* interacts with the network equipment through pluggable drivers. In addition, the Driver Application Programming Interface (API) – part of the SBI – defined in version 2.0 has been extended in version 2.1 to add new network protocols and data models (e.g., gNMI) to the *SBI* component, and enhance scalability. The *Automation* component implements several Event-Condition-Action (ECA) loops defining the automation procedures in the network. The *Monitoring* component manages the collection of different metrics configured for the network equipment and services, stores monitoring data related to selected KPIs. It 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.

The TeraFlowSDN controller uses the *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 *NBI* component is the interface from internal gRPC (gRPC Remote Procedure Call) and protocol buffers towards external Representational State Transfer (REST)-like requests. It provides a REST-API–based NBI to 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. The *WebUI* also provides a load generator that can be used to quickly set up services for performance assessment purposes.

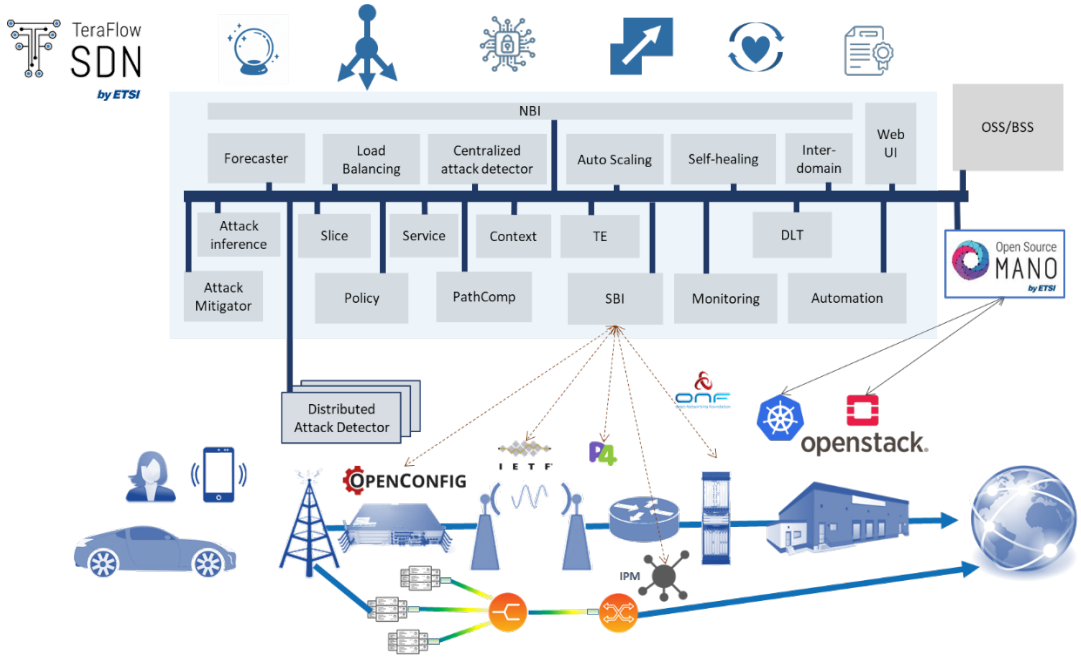


Figure 1. TeraFlowSDN architecture for release 2.1

Compared to release 2 (described in D5.2), TeraFlowSDN release 2.1 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 2.1 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 and the Path Computation component have also been introduced. This component allows new use cases, such as energy-aware service placement.

The Cybersecurity mechanisms were also updated, including novel components for attack detection (either distributed or centralized), attack inference, and attack mitigation. The scalability of the components has been improved. In addition, several novel use cases are supported, such as the optical layer attack detection using supervised or unsupervised learning. The *Distributed Ledger Technology (DLT)* component has also been extended to interact with the *Inter-domain* component and use the deployed Hyperledger Fabric.

3. Metrics Definition and Collection

TeraFlowSDN offers a built-in metrics collection framework that can benchmark and monitor the performance of the internal components. A preliminary version of this framework was introduced in D5.2. In this deliverable, for the sake of completeness, we revisit the metrics collection framework, and then we introduce the improvements made since D5.2. Moreover, we include a list of metrics used throughout this deliverable.

Unlike the *Monitoring* component, which collects 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 perform a given task (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 presents several metrics definitions that are relevant for the scenarios addressed in the experiments. These metrics are detailed in the next sections, on a 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 2, two main platforms are used:

1. *Prometheus*: a solution for exposing and collecting metrics about software performance at run time. Its adoption has two main steps: *(i)* instrumenting a component to capture the relevant metrics and *(ii)* configuring the main Prometheus server to extract the exposed metrics periodically. Step *(i)* is the responsibility of the component developer. Prometheus offers libraries for integration with all the programming languages adopted by TeraFlowSDN so far. For step *(ii)*, one challenge when using cloud-native solutions is to maintain a configuration consistent with the replicas that exist in the system. These replicas may change dynamically depending on the load offered to each TeraFlowSDN component. To address this task, we leverage Prometheus Operator, an open-source software responsible for monitoring the Kubernetes replication process and configuring Prometheus accordingly. The use of the Prometheus Operator guarantees that metrics are collected from all the replicas.
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). This allows, for instance, to create dashboards that summarize network load and its impact on the TeraFlowSDN components. An example of this will be shown in Section 6.3.1.3, in the scalability performance evaluation of scenario 3 monitoring the cybersecurity of L3VPNs.

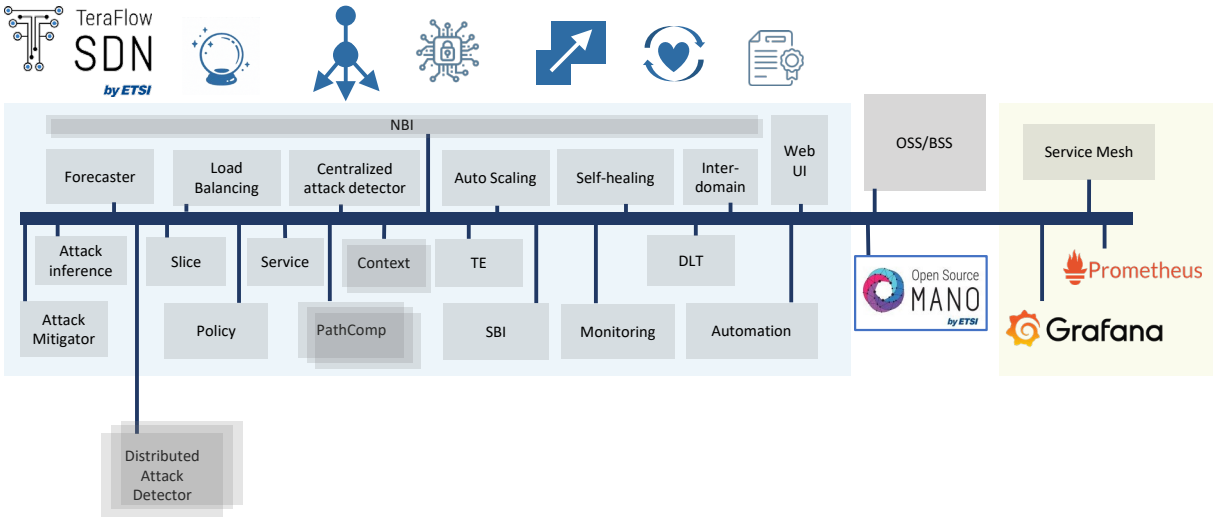


Figure 2. TeraFlowSDN extended architecture encompassing the metrics collection framework

In addition to these two main pieces of software, we rely on a *Service Mesh* software capable of load-balancing gRPC requests. 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.

3.1. 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 protobuffers, *(ii)* a binary format that lowers communication overhead, and *(iii)* easy use and interoperability across programming languages. However, the gRPC protocol is built on top of HTTP/2, which unlike HTTP/1.1, 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 reducing signaling overhead (e.g., connection handshaking time), it makes it harder to provide load balancing when gRPC is used with multiple replicas of the same service, i.e., 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 object 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 have built-in monitoring capabilities, keeping track of the health and detailed parameters of the connections among all components within a deployment. For instance, service mesh monitoring can measure how many requests a component/service or replica receives per second and the response time distribution for such requests.

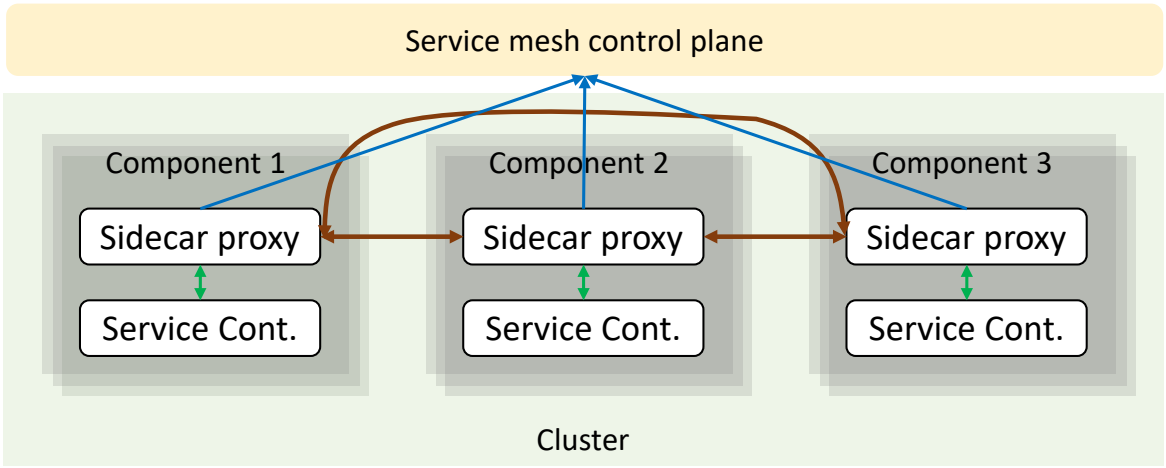


Figure 3. Architecture of the service mesh with sidecar proxy and service container

Figure 3 illustrates the architecture of a service mesh deployment. Each component is configured to let the service mesh control plane act over it. The service mesh control plane's main actions are to include a new container in the Pod called *sidecar proxy*. The sidecar proxy intercepts outgoing communication to other components and routes 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 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 service mesh deployment are the *Istio service mesh*, and the *Linkerd service mesh*. Both tools are free to use and open source. Their core functionalities are very similar. Ultimately, Linkerd was selected due to its seamless integration with MicroK8s and lightweight sidecar Pods.

3.2. 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, upon request, exposes the current state of the metrics to a specific URL.

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

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

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*: a time-series database responsible for persisting/storing the collected metrics;
- *Web-based UI and API*: a user interface where users can query and visualize data stored in the database. The API allows access to the same information without the GUI, which becomes ideal for extracting only the data.

In the TeraFlowSDN controller, we also adopted the Prometheus Operator, which complements the functionalities of Prometheus with automatic detection and configuration of replicas. Figure 4 illustrates one case when the Kubernetes Pod Autoscaler adds new replicas, and Prometheus Operator automatically detects the replicas, and configures Prometheus to collect metrics from them.

Figure 4 shows the total computation time a component spends during a load increase. The important aspect in the figure is that we have initially a single replica, represented by a single (yellow) line. Around minute 10:36, new replicas are added, represented by the new (red, blue, and green) lines being added to the time series. With the ability to monitor several replicas, the metrics collection framework represents a flexible solution for monitoring and observing TeraFlowSDN's internal performance. Moreover, the automatic inclusion/deletion of data collection endpoints as the number of replicas change characterize a robust internal monitoring solution. Enabling Prometheus in a component after it has been instrumented is relatively easy. Adding one new element in the component YAML configuration file is enough.

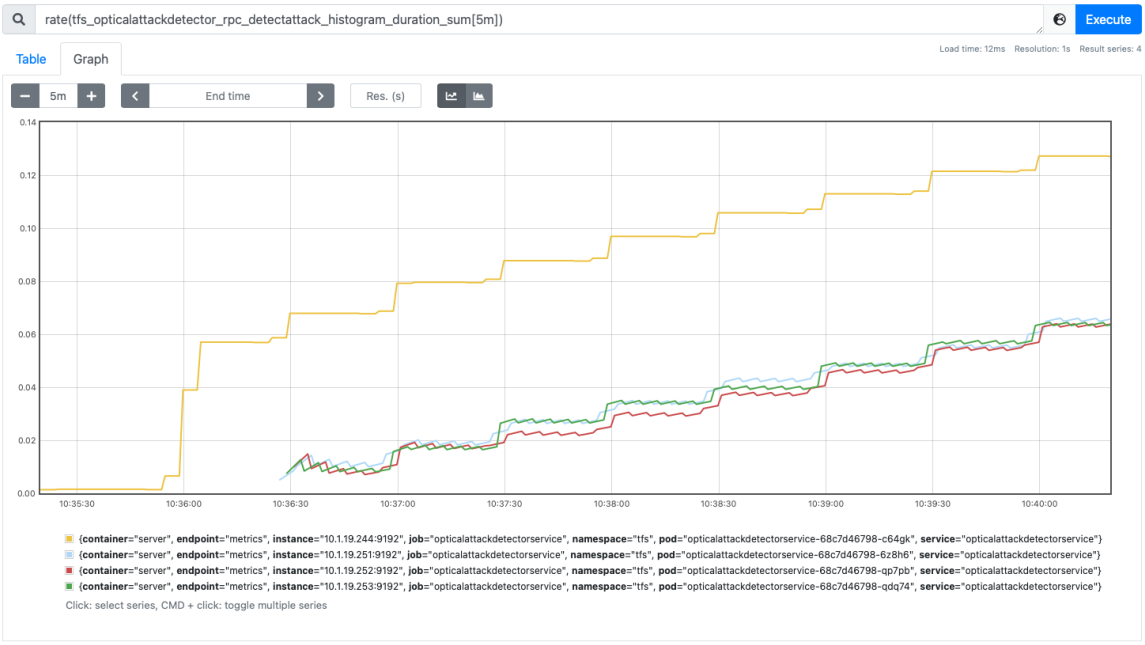


Figure 4. Screenshot of Prometheus metrics when new replicas are created

3.3. Grafana

Grafana is the final open-source software used in the metrics collection framework. Grafana is focused on visualizing different KPIs for their concurrent analysis. This is done through dashboards. An important feature of Grafana is the ability to create of dashboards combining data from different sources. For instance, in the case of TeraFlowSDN, the Monitoring component is responsible for monitoring services and devices currently active in the network. In addition, we 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. We refer to D5.2 for a more complete description, including screenshots.

3.4. Load Generator

One of the key tasks when benchmarking TeraFlowSDN is the ability to generate load for the components in a well-controlled and reproducible manner. For this purpose, we developed the *Load Generator* component. The Load Generator can be accessed through a Command Line Interface (CLI), or, more conveniently, a WebUI.

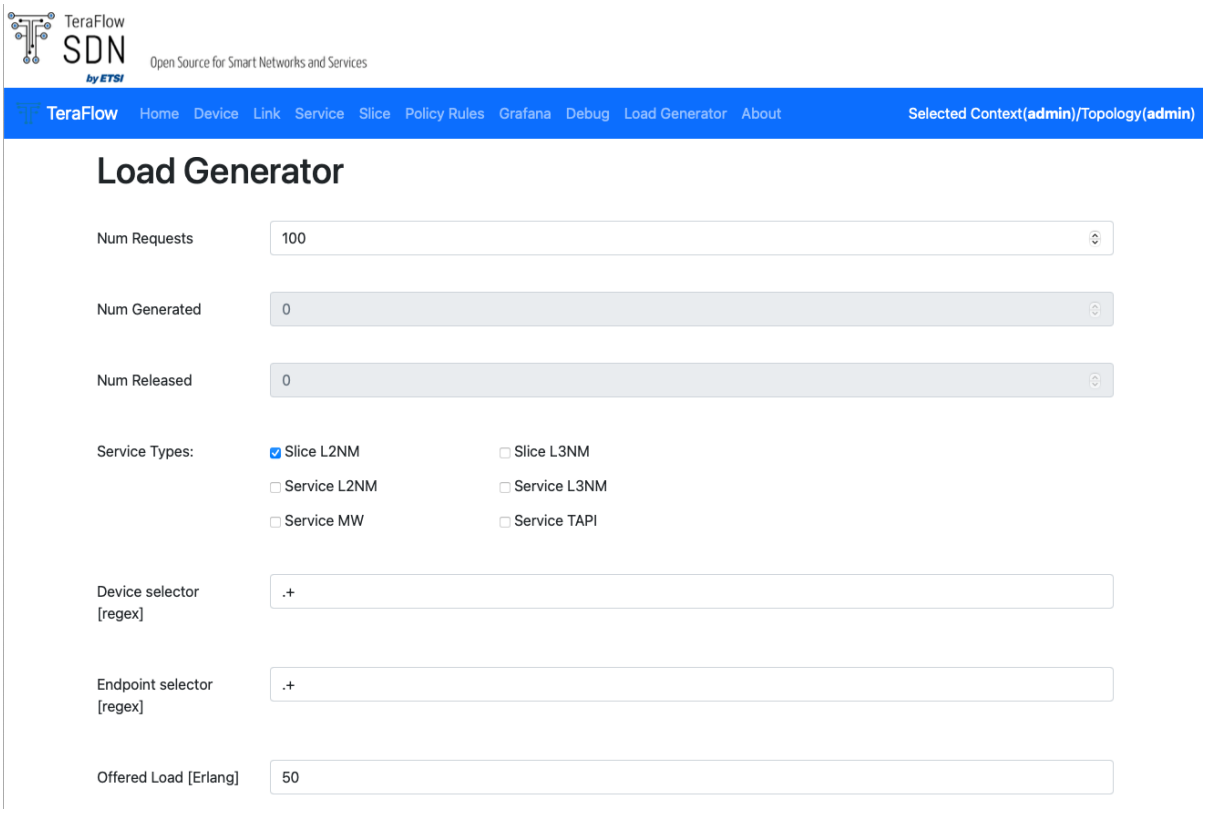


Figure 5. Screenshot of the load generator configuration through the WebUI

Figure 5 shows a screenshot of the web form provided by the WebUI for invoking the Load Generator. The Load Generator provides several settings that allow for flexible traffic generation. They are:

- Number of service requests to be generated;
- Service type(s) (allows to define which service types will be generated in the service requests);
- Device selector (allows to define which devices should be considered in the service requests);
- Endpoint selector (allows to define which device/service endpoints should be considered when provisioning the generated service requests);
- Offered load (defines the load incurred by the generated service requests, in Erlang);
- Availability (defines the range of availability required by the generated service requests);
- Capacity (defines the range of capacity required by the generated service requests, in Gbps);
- End-to-end latency (defines the range of latency required by the generated service requests, in ms);
- Maximum number of workers (defines how many concurrent processes will be employed to generate the desired load);

- Teardown (sets whether services are torn down after the holding time expires);
- Record to DLT (enables or disables the recording of the services to the DLT).

The Load Generator also takes advantage of the Metrics Collection Framework, exporting internal KPIs to Prometheus. These KPIs can be visualized in a custom Grafana dashboard provided with TeraFlowSDN.

4. Scenario 1: Autonomous Network Beyond 5G

In this section, we discuss a scenario which specifically explores the advancement of autonomous networks beyond the capabilities of 5G. It primarily emphasizes the progression of transport networks by integrating SDN and NFV (Network Function Virtualization) technologies hierarchically. It is an operator-led scenario, as it centers around the evolution of transport networks with the involvement and guidance of network operators. Firstly, we introduce the scenario. Secondly, we present its alignment with TeraFlowSDN architecture. Thirdly, we present the performance evaluation. Finally, we provide a summary of scenario conclusions and future steps.

4.1. Scenario Introduction

In 5G, network operators rely on pre-defined templates for services and network slides embedded within the systems. However, it became evident that this approach is not scalable beyond 5G (B5G) scenarios, where networks need to dynamically adapt to the changing demands of end-users. In B5G, the network, operated by the network slice controller, must compute a deployment plan that considers relevant network service functions and align it with a service provisioning and configuration plan. This dynamic and intelligent process ensures that the requested service is matched appropriately, provides adaptation capabilities during service operation, and relates the services to the available underlying network resources. In cases where services require resources from multiple technology domains, orchestrating these resources is necessary to provide multi-layer and multi-domain services. Network automation is the key to addressing such adaptive environments.

While SDN promised network programmability, the challenge lies in integrating different tools with their own APIs and varying data models, which can be costly and time-consuming. The TeraFlowSDN controller supports operator-driven use cases and workflows, addressing the objectives of this scenario by enabling the programmability of network elements and technology-based SDN controllers with the necessary north-bound and south-bound interfaces.

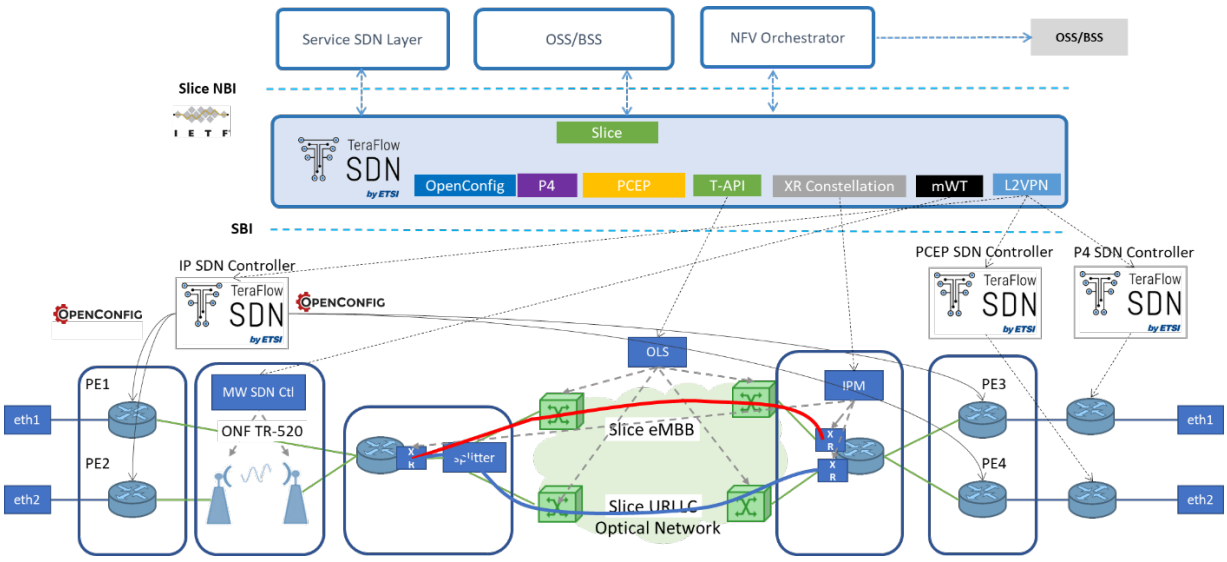


Figure 6. Scenario 1: Autonomous Network Beyond 5G

The high-level architecture depicted in Figure 6 illustrates the envisioned scenario. It includes integrated network elements within different technological domains, enabling the autonomous provisioning, configuration, and management of transport network slices. These slices consist of

various Virtual Private Network (VPN) services like Layer 2 (L2VPN) and Layer 3 (L3VPN) services with dedicated Service Level Agreements (SLAs). The interaction between the NFV Orchestrator (e.g., ETSI OpenSource MANO) and TeraFlowSDN North-Bound Interface (NBI) allows L2/L3VPN connectivity provisioning.

The optical network domain is managed using the Open Networking Foundation (ONF) Transport API (TAPI), which serves as a South-Bound Interface (SBI) towards an Optical Line System (OLS) or an optical SDN controller responsible for optical network elements. The microwave transport network follows the ONF Technical Reference (TR) 352 and Internet Engineering Task Force (IETF) Network Topology data models. A dedicated microwave SDN controller interacts with TeraFlowSDN via the appropriate SBI based on ETSI mWT 024 specifications.

Layer 3 (L3) routers are controlled using OpenConfig data models. The TeraFlowSDN Controller is instantiated as an IP SDN controller, interacting with a parent TeraFlowSDN controller that functions as an End-to-End (E2E) Orchestrator. Alternatively, the L3 routers can be controlled using the Path Computation Element Protocol (PCEP). In this case, the TeraFlowSDN Controller is instantiated as a dedicated PCEP SDN controller, interacting with the parent TeraFlowSDN controller as the End-to-End Orchestrator. Additionally, P4 switches can be controlled by a dedicated instance of the TeraFlowSDN controller. This controller, along with the parent TeraFlowSDN controller acting as the End-to-End Orchestrator, enables effective management of P4 switches.

4.2. Alignment with TeraFlowSDN architecture

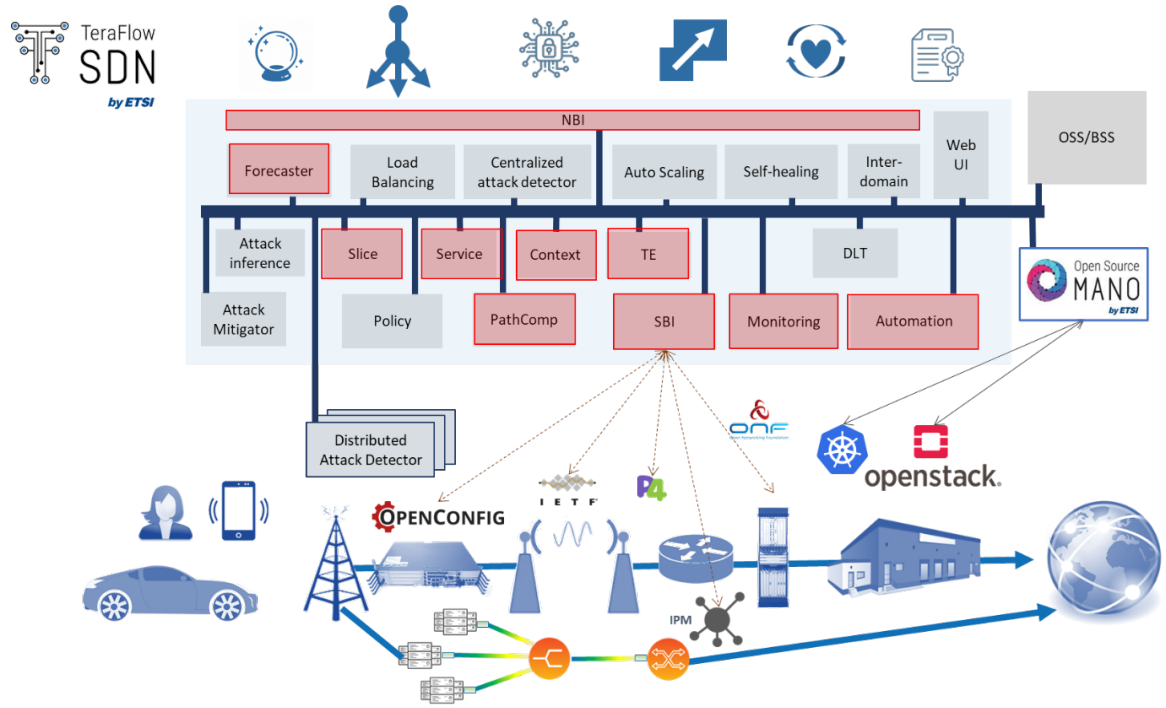


Figure 7 Scenario 1 E2E TeraFlowSDN instantiation

Figure 7 shows the instantiation (configuration and TeraFlowSDN 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 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 TeraFlowSDN controller.

Scenario 1 involves the following components:

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

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

- Zero-touch Device Automation
- L2/L3VPN Service Management and Integration with ETSI OpenSource MANO
- Slice Grouping
- Policy-driven Service Restoration with P4 devices
- Energy-Efficient Path Computation

4.3. Performance Evaluation

This section presents the performance evaluation performed for the different workflows composing scenario 1. The workflows have been introduced and detailed in D5.2, Section 5.5. In this deliverable, we focus on the performance evaluation of the workflows. First, we introduce the testbed setup used by the partners to evaluate the performance of the workflows. Then, we present the performance evaluation of each workflow.

4.3.1. Testbed Setup

The performance evaluation for Scenario 1 has been carried out in a geographically distributed testbed interconnecting CTTC premises with Telefonica I+D premises through a legacy VPN tunnel. Moreover, a testbed located at UBITECH has been used to evaluate the P4 workflow.

At CTTC premises, the testbed consists of a hybrid cluster composed of 3 machines interconnected by means of a GbE switch. The technical specifications of the machines are detailed below:

- 1 machine featuring an Intel Core i9-10900K CPU @ 3.70GHz (10 cores / 20 threads) with 64 GB of RAM, a 2 TB Intel NVME disk, and 1x 10GbE NIC + 1x 1GbE NIC.
- 2 machines featuring an AMD Ryzen Threadripper 3960X 24-Core Processor (24 cores / 48 threads) with 128 GB of RAM, a 2 TB SSD SATA disk + a 4 TB HDD SATA disk, and 2 NIC cards (2x 1GbE ports + 1x 1GbE port).

Each machine runs an Ubuntu 22.04.2 LTS Server Edition Operating System. The machines are clusterized by means of MicroK8s v1.24.13 installed through the official Canonical Snap packages and

configured according to the instructions available in the ETSI TeraFlowSDN Deployment Guide [DepG]. To improve the cluster's performance, the software has been installed directly on the physical server without any Virtual Machine, and the CockroachDB database has been deployed in cluster mode to incorporate data replication, distribution, and consistency enforcement in the performance assessment. Thus, all the performance assessments in this scenario, especially those related to the Context component, account for such data management features. However, the Horizontal Pod Auto-Scalers (HPA) of Kubernetes for all the TeraFlowSDN controller components have been deactivated to assess the components in a non-scalable environment. In section 5.3.4, where we evaluate the performance of attending Service and Slice requests in a scalable manner, the HPAs for the TeraFlowSDN controller components have been enabled.

At Telefonica I+D premises, the testbed consists of some HL-5 Edgecores, Ubuntu virtual machines, SIAE MW devices, and a SpirentTest center with the following compute specifications:

- 2x Infinera DRX-30 router running the ADVA NOS-OPX-B-21.5.1. These devices are an open cell site gateway platform that provides a combination of 1GE, 10 GE, 25 GE and 100GE interfaces using commercial silicon to provide a cost-effective, software-centric, and flexible solution for network routing. These devices support Netconf/Openconfig, gNMI, BGP-LS, PCEP and OSPF.

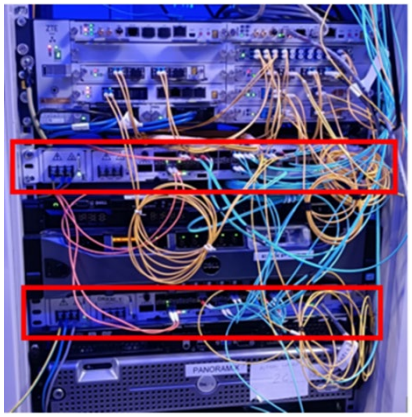


Figure 8. Edgecore DRX-30 chassis layout

- 1x Edgecore AS7315-30X router running the ADVA NOS-OPX-B-21.5.1. This device 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 of mobile networks. This device supports Netconf/Openconfig, gNMI, BGP-LS, PCEP and OSPF.

Figure 9. AS7315-30X chassis layout

- 1x Spirent N12U [SPI22]. This device 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.

- 1x Dell PowerEdge R730. This device has 56 CPU intel Xeon E5-2690 v4 @ 2.60GHz cores capable to boos 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.

Figure 10. Dell R730

- 1x Dell PowerEdge R720. This device has 32 CPU intel Xeon E5-2680 @ 2.70GHz cores capable to boos up to 3.50GHz, 125GB of DDR4 RAM and 100GB of SSD storage. It runs Ubuntu 20.04.5 LTS and 10Gbps network interfaces.

Figure 11. Dell R720xd

- For the virtualization environment we use Openstack version Xena and Stein, KVM and VMware versions 6, 6.7, and 7, respectively. We have two Openstack clusters running Stein and Xena respectively. The KVM and VMware hosts run independently of one another and are managed individually. This offers us a great balance between flexibility and scalability. We can effortlessly deploy a large number of virtual machines using the Openstack clusters, running a total of 808 vCPUS, 1512GB of RAM and 23TB of hybrid storage. But we can also use KVM and VMware to deploy machines that may need specific configurations tailored to their unique requirements, such as specialized software installations, customized network settings, or intricate network connectivity with external systems.
- The SIAE equipment at Telefonica premises includes a virtualized server with 4 vCPUs, 32GB of RAM, and a 350GB hard drive, running SUSE Linux 12 SP4 as Operating System. On top of it, SIAE intermediate MW controller version SM-DC 8.3.2 is in charge of managing a radio link terminated by two SIAE AGS20 terminals.
- The Infinera local TeraFlowSDN emulation environment uses 3 Dell PowerEdge R420 servers, one for running TeraFlowSDN, second running for IPM installation and third running emulated XR modules. Servers have Intel Xeon ES-2470 CPU@2.4GHz (20cores/40threads), 64GB RAM, 1T-2T SSD, 6x1G and 2x10G ethernet ports and running Ubuntu 22.04 and using Edgecode AS5812-54X router between IPM server and emulated XR modules servers to mimic as realistic network configuration and behavior as possible on emulated setups. For TeraFlowSDN XR constellation pluggable development we have simpler pure emulated setups where Integrated Performance Monitoring (IPM) and emulated XR modules can be run on the same server (used for example Telefonica lab installation). In addition of this, physical XR modules were hosted on 2 x Edgecore DCS240/AS7926-32DB with modified Edgecore Enterprise SONiC, where SONiC was extended to support XR modules, supporting different 400G port modes, including 1x400G and 4x100G breakout modes. From the TeraFlowSDN viewpoint the emulated or physical devices gets abstracted behind IPM allowing setups to use different level emulated or physical XR devices supporting targeted test use case.

Figure 12. Dell PowereEdge R420 server

Figure 13. Edgecore DSC240/AS9726-32DB

UBITECH provides a testbed where the P4-based Zero-touch provisioning (Section 4.3.2) and Policy-driven service restoration (Section 4.3.5) scenarios are tested. The testbed consists of an Intel NUC device with the following compute specifications:

- CPU: 4 physical Intel(R) Core(TM) i7-5557U CPU @ 3.10GHz
- DRAM: 16GB of Kingston DDR3 DRAM at 1600 MT/s
- Storage: 1TB Samsung SSD 850
- Network interfaces: 2x 1GbE through an onboard NIC

For this testbed, UBITECH deploys the ETSI TeraFlowSDN controllers as a Kubernetes managed service and a software-based P4 topology with 8 bmv2 P4 switches through Mininet. A set of Mininet hosts (client and server) is used to inject traffic into Mininet.

4.3.2. Zero-touch Device Automation

For the Zero-touch Device Automation workflow, we consider the evaluation case of the overhead consumed by the TeraFlowSDN controller to onboard new devices, without considering the time consumed by the device to self-configure. A similar strategy was adopted in [OFC22]. Therefore, the results reported in the following are obtained using emulated device driver.

First, we evaluate the overhead introduced by the TeraFlowSDN controller in onboarding devices emulated through the Emulated device driver in the SBI. This emulated driver is mainly devoted to debugging and assessing the TeraFlowSDN controller's logic. It stores and retrieves the configuration rules from internal memory and does not interact with external devices. Using this driver, we can measure the overhead introduced by the TeraFlowSDN controller while onboarding new devices regarding how much time is consumed by data base accesses, (de)configuration rule processing, etc.

This experimental assessment has been carried out using the network topology depicted in Figure 14. The topology consists of 7 emulated packet routers interconnected by means of 18 unidirectional links. Each router has 9 ports (also named as endpoints in TeraFlowSDN terminology). A subset of these endpoints (i.e., 36, two for each unidirectional link) is used to interconnect the routers, while the rest are used as client ports for the performance assessment.

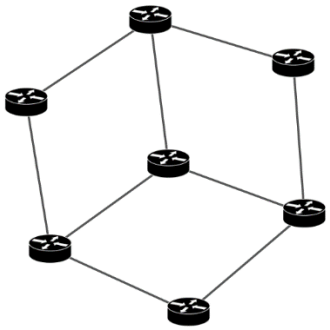


Figure 14. Emulated network topology

The performance assessment results are depicted in the following figures, each plotting the cumulative distribution function of the execution times of each operation and/or method. The results are obtained by measuring the time taken by each stage of the onboarding process when onboarding the topology depicted in Figure 14. Detailed descriptions of each method evaluated in this section is provided in D3.2, Section 4.1.4.

Figure 15 illustrates the execution time for a single *AddDevice* RPC method execution in the SBI component. In this case, *Device* and *Device Driver* refer to the specific parts within the SBI component responsible for interacting with devices. Detailed information about the SBI component design can be found in D3.2, Section 4.1.2. The minimum execution time of this RPC method is around 100ms; almost 60% of executions are completed in less than 300ms, and it never exceeds 400ms.

To illustrate how this time is distributed, Figure 16 showcases how the *total* time (olive-colored) is distributed among the different internal operations done by the *AddDevice* RPC method. This *total* time is the composition of two main operations: *wait_queue* (cyan-colored), which accounts for how much time the operation is waiting on the internal queue (note that two requests on the same device needs to be executed sequentially), and *execution* (orange-colored) that accounts for the sum of times required by the different operations. As it can be expected, the most time-consuming operations are those related to quering the Context component to check the existence of the device in the database (*get_device*, green-colored) and the storage of the updated device information in the database (*set_device*, gray-colored).

The time consumed by the *GetConfig* and S*etConfig* operations in the Emulated Driver are shown in Figure 17. These operations always consume less than 1ms given they perform in-memory storage and retrieval of configuration rules.

Finally, Figure 18 shows the distribution of time for the Context RPC methods that are involved in the onboarding of new devices, in particular, the *ListDevices*, *SelectDevice*, and *SetDevice* RPC methods. *SelectDevice* (orange-colored) is the method that consumes less time; around 50% of executions take less than 10ms and always it takes less than 100ms. In contract, *SetDevice* (green-colored) takes 20-60ms given the database updates need to check the database schema constraints are fulfilled. *ListDevices* takes longer given it needs to retrieve all the devices in the database, but it never exceeds 100ms.

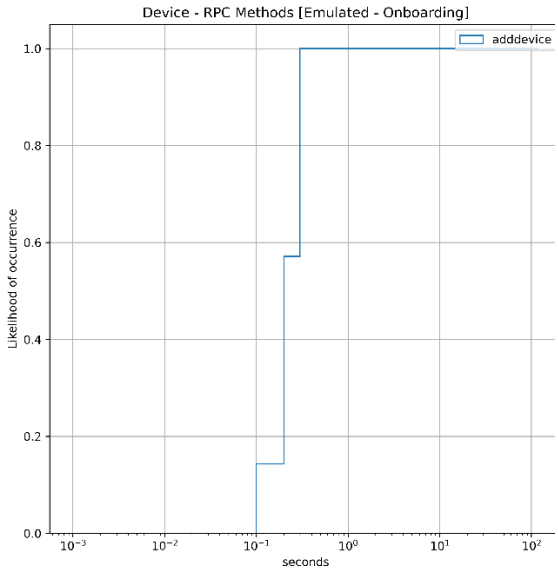


Figure 15. Device component – AddDevice RPC

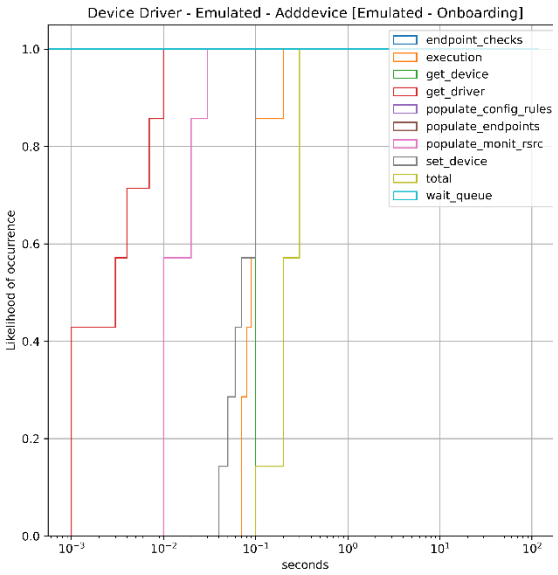


Figure 16. Device component –AddDevice RPC internal operations

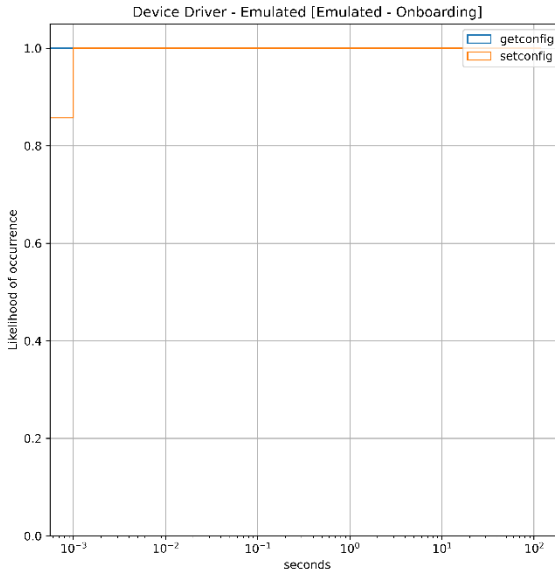


Figure 17. Emulated Device Driver – GetConfig/SetConfig Methods

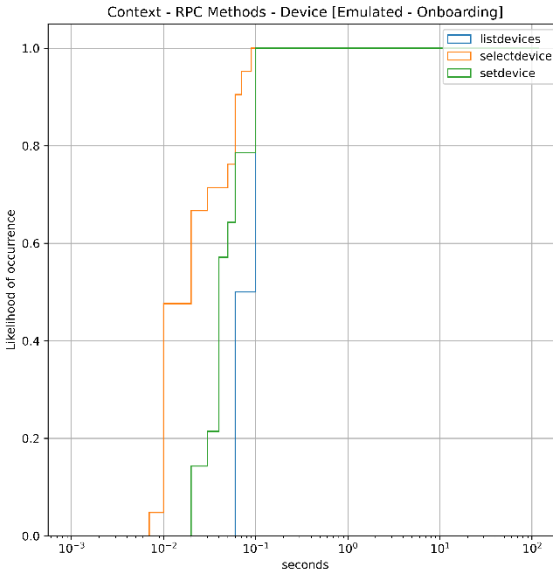


Figure 18. Context component – Device-related RPC methods

Table 1 provides the summary of obtained results.

Table 1. Scenario 1 - Device Onboarding KPIs

Name	Metric	Value	Comment
Device on-boarding time	On-boarding time contribution by internal TeraFlowSDN components	100-400 ms	Measured using Emulated Driver

4.3.3. L2/L3 VPN Service Management and Integration with ETSI OpenSource MANO

The evaluation of L2/L3VPN Service Management and Integration with ETSI OpenSource MANO is split into three parts. The first part assesses the time required to setup and teardown L2 VPN services. The second part is devoted to evaluating the L3 VPN services. Finally, the third part validates the creation of connectivity services from ETSI OpenSource MANO.

4.3.3.1. L2 VPN Service Management

This section reports the performance results to setup and teardown 1,000 L2VPN service requests. For the assessment, we used the topology illustrated in Figure 14 and described in Sec. 4.3.2 and the load generation tool described in Sec. 3.4. The load generator has been configured with an offered load of 50 Erlang and 10 seconds of holding time per L2VPN service. The requests are sent directly to the Service component through the TeraFlowSDN gRPC internal API, where the setup and teardown time are measured directly.

The performance evaluation results are depicted in the following figures as the cumulative distribution function of the execution times of each operation and/or method. Detailed descriptions of each method evaluated in this section are provided in D3.2, Section 4.2.3.1.

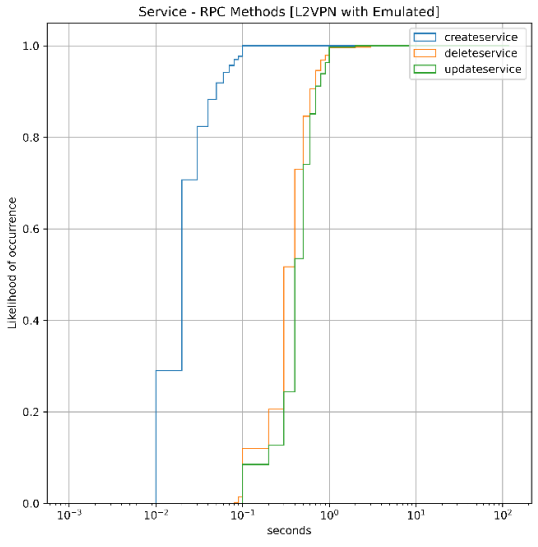


Figure 19. Service component RPC methods for L2VPN w/Emulated

Figure 19 illustrates the execution times of the *CreateService*, *UpdateService*, and *DeleteService* RPC methods of the Service component for L2VPN services. The services are created internally to obtain a unique identifier, and then populated with the required endpoints, constraints, and configuration rules. Therefore, the *CreateService* time accounts only for the time required to create the service in

the database. This operation takes less than 30ms in 80% of the requests, with the exception of a few cases where it can take up to 100ms. The *UpdateService* method requests the path computation and the setup of the required configuration rules for the services. The *DeleteService* method retrieves the service from the database and deconfigures the rules related to the service. The execution time of the *UpdateService* method is slightly larger than the *DeleteService* method. This is because the former needs to wait for the Path Computation result. The *UpdateService* method always takes at least 100ms, and for 80% of the executions it does not exceed 150ms. The *DeleteService* method, takes at least 90ms, and for 80% of the executions it does not exceed 140ms. In very rare cases the *UpdateService* and the *DeleteService* methods might take a second or more to complete due to congestion in the database. Further investigation on database resources might be needed to clarify this congestion.

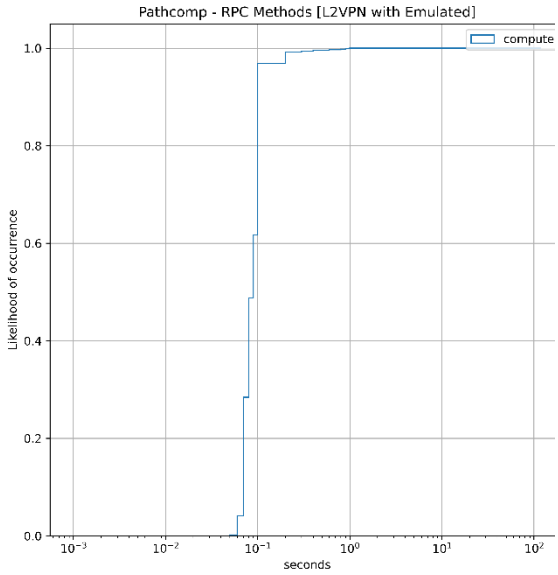


Figure 20. Path Computation component computation time

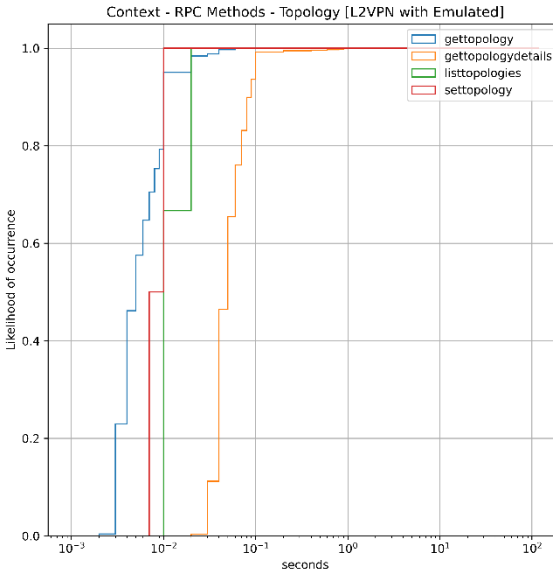


Figure 21. Topology-related methods of the Context component

Figure 20 depicts the time consumed by the Path Computation component to compute the paths for the new service requests. A detailed description of the Path Component interfaces can be found in D3.2, Section 3.4.3. It is worth noting that this time accounts for parsing the request received from the Service component, collecting the topology from the Context component, offloading the computation to the Backend Path Computation component, and composing and sending back the reply to the Service component. As it can be seen, the minimum consumed time is 60ms and almost 95% of times it does not exceed 100ms. In a few exceptional cases it might take up to 1 second. As previously commented, further research on distributed database resources might be needed to remove these outliers.

The Path Computation component uses the Context *GetTopologyDetails* method to retrieve the topology with the devices and links it needs to perform the path computation. The *GetTopologyDetails* method, takes, at least, 30ms to complete (Figure 21), in around 80% of times it takes no more than 70ms, and 90% of cases it takes less than 100ms. However, in some edge cases it can take hundreds of milliseconds depending on the congestion in the database, with the worst case measured in 1 second. This explains the performance of the Path Computation component in those edge cases.

The Service Handler triggers the actual rule configuration and deconfiguration in the devices specialized in the specific types of services. A detailed description of the Service Handlers can be found in D3.2, Section 4.2.2. Two important methods in the Service Handlers are *SetEndpoint* (responsible for configuring rules in the devices) and *DeleteEndpoint* (that performs the deconfiguration operations). The performance of both methods when considering L2VPN services with emulated devices is very similar (Figure 22). This behavior is expected given the Emulated driver stores and retrieves the rules directly from in-memory storage. It must be noted that the in-memory storage is to emulate the behavior of the device Network Operating system, thus, it does not replace the need for storing the configuration rules also in the Context database as the purpose of both databases is different. This experiment validates the stability of the rest of the involved components and the overhead introduced by the TeraFlowSDN controller. Figure 22 shows that the minimum consumed time is around 70-80ms and in 85% of the cases it does not exceed 100ms. Besides, in edge cases these methods can consume up to 1 second.

Figure 23 plots the performance of the *AddDevice* and *ConfigureDevice* methods of the Emulated Device Driver in the Device component. The *ConfigureDevice* is responsible for configuring and deconfiguring the rules on the devices through the device drivers, thus its performance directly affects the performance of *SetEndpoint* and *DeleteEndpoint* methods of the Service component. In the topology used for this validation, in general, the paths traverse, on average two devices. This explains why the time consumed by *SetEndpoint* and *DeleteEndpoint* methods is roughly twice the time consumed by the *ConfigureDevice* method (in average two devices need to be configured for each service). In this case, the *AddDevice* method is not relevant, since it is used only during the on-boarding of the devices, but is included in the plot for the sake of completeness.

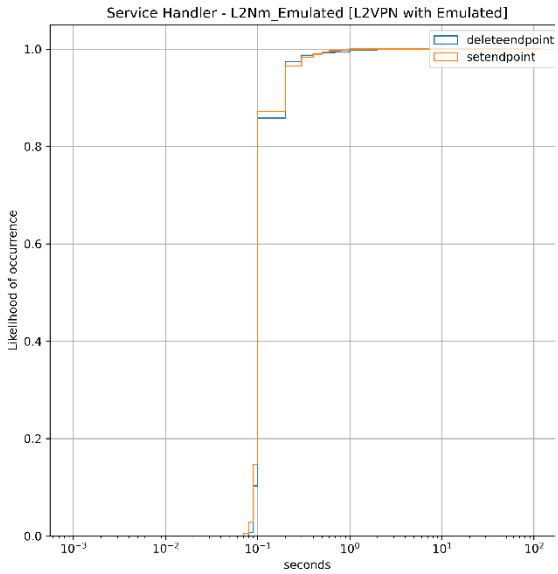


Figure 22. L2NM Emulated Service Handler – Set/DeleteEndpoint methods

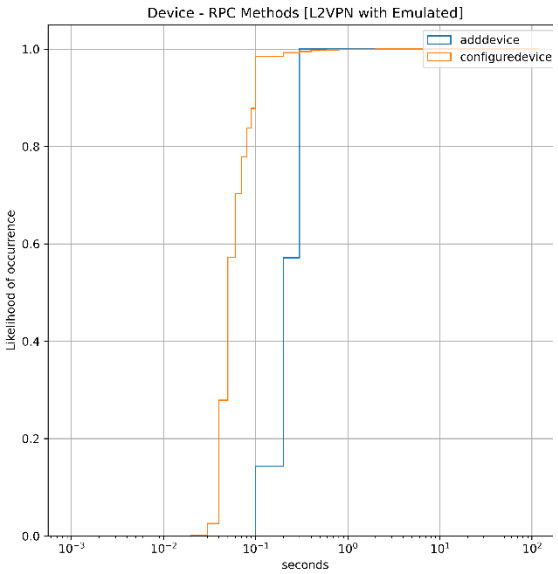


Figure 23. Device component – RPC methods

Next, we analyze the performance of the *ConfigureDevice* method of the SBI component. Figure 24 showcases the details of the internal operations of the *ConfigureDevice* method. The *total* time is the sum of the *wait_queue* and the *execution* time. The rest of the measurements are contributors to the *execution* time. As can be seen, the *wait_queue* time is negligible (in the order of microseconds). The time for *configure_rule* and *deconfigure_rule*, *i.e.*, the actual configuration of rules in the in-memory

storage of the Emulated driver, are also negligible. The execution time is dominated by the accesses done to the Context database to retrieve (*get_device*) and/or update the state (*set_device*) of the device in terms of known configured rules. These times are aligned with those reported in Figure 25 for the Device-related RPC methods in the Context component. Note that other components and operations in the TeraFlowSDN controller use the methods to *SelectDevice* and *SetDevice*; which explains the slight variations between the two plots.

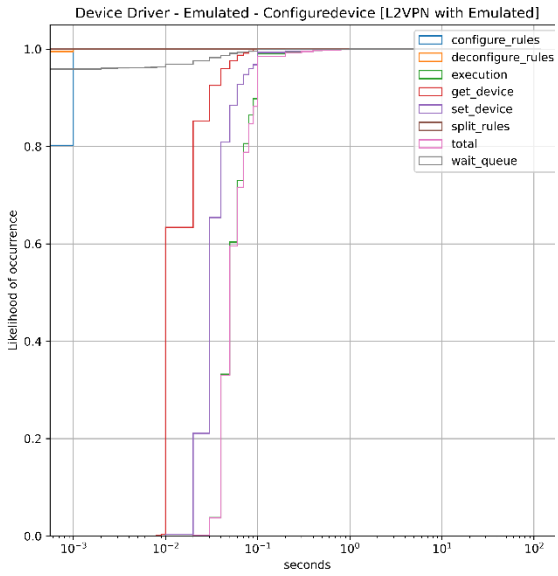


Figure 24. Device component – ConfigureDevice internal operations

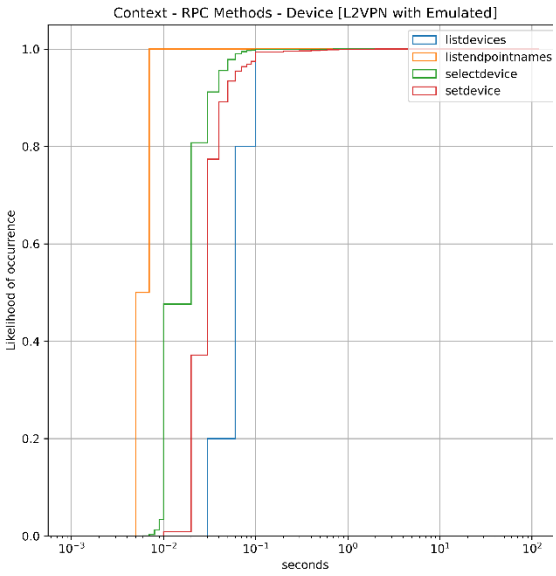


Figure 25. Context component – Device-related RPC methods

4.3.3.2. L3 VPN Service Management

This section reports the performance results to setup and teardown 1,000 L3VPN service requests. For the assessment, we used the topology illustrated in Figure 14 and described in Sec. 4.3.2 (page 33 of this document), and a load generation tool to automate the evaluation described in Section 3.4. The load generator has been configured with an offered load of 50 Erlang and 10 seconds of holding time per L3VPN service. The requests are sent directly to the Service component through the TeraFlowSDN gRPC internal API, and the setup and teardown time is measured directly in the Service component.

The performance evaluation results are presented in the following figures as the cumulative distribution function of the execution times of each operation and/or method. Detailed descriptions of each method evaluated in this section are provided in D3.2, Section 4.2.3.2 (page 112).

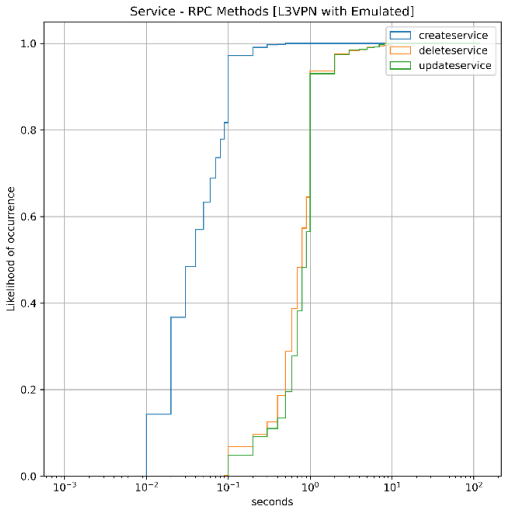


Figure 26. Service component RPC methods for L3VPN with emulated devices

Figure 26 illustrates the execution times of the *CreateService*, *UpdateService*, and *DeleteService* RPC methods of the Service component for L3VPN services. As it was the case for L2VPN services, internally, the services are first created to obtain a unique identifier per service. Then they are populated with the required endpoints, constraints, and configuration rules. For this reason, the *CreateService* time only accounts for the time required to add the service in the database. This operation takes less than 100ms in 80% of the cases, while in some exceptional cases, it can increase up to 400ms. The *UpdateService* method triggers the path computation and the setup of the configuration rules required for the services. The *DeleteService* method retrieves the service from the database and performs the deconfiguration of the rules related to the service. Since it needs to wait for the results of the Path Computation method, the execution time of *UpdateService* is slightly larger than the one of the *DeleteService* method. The *UpdateService* method always takes at least 100ms, and for 80% of the cases it does not exceed 1 second. The *DeleteService* method needs at least 100ms, and for 80% of the cases it does not exceed 1 second. In rare cases, the UpdateService and the DeleteService methods might take a few seconds to complete due to congestion in the database. Note that L3 Services require managing a larger number of configuration rules than L2VPN Services, thus increasing the number of rules to be updated and persisted by the Context component.

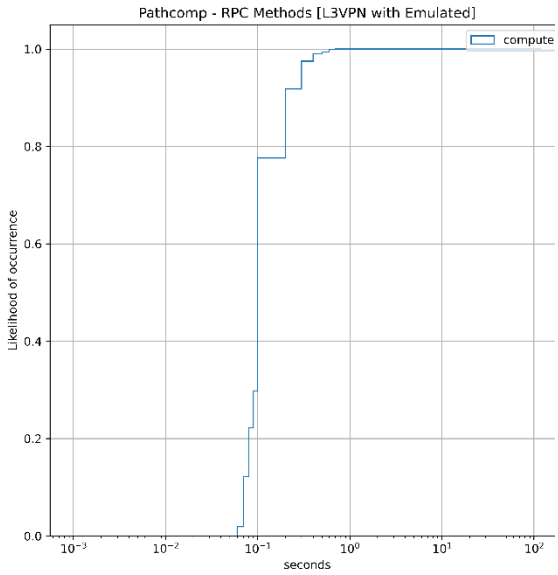


Figure 27. Path Computation component computation time

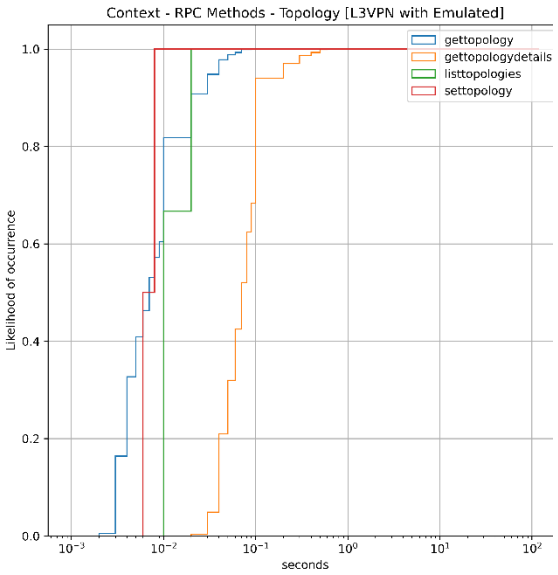


Figure 28. Topology-related methods for the Context component

Figure 27 depicts the time consumed by the Path Computation component to compute a path for new service requests. It is worth noting that this time accounts for parsing the request received from the Service component, collecting the topology from the Context component, offloading the computation to the Backend Path Computation component, and composing and sending back the reply to the Service component. As it can be seen from the figure, the minimum consumed time is 60ms, and almost 80% of the executions do not exceed 100ms.

The Path Computation component uses the Context *GetTopologyDetails* method to retrieve the topology with the devices and links it needs to perform the path computation. The *GetTopologyDetails* method, takes, in about 80% of the cases, between 30ms and 100ms to complete (Figure 28). However, in some special cases it can take hundreds of milliseconds depending on the congestion in the database, with the worst case being close to 1 second. This explains the performance of the Path Computation component in those edge cases.

The actual rule configuration and deconfiguration in the devices are triggered by the Service Handler specialized in the specific types of services. More details about the Service Handlers can be found in D3.2, Section 4.2.2. The two important methods in the Service Handlers are *SetEndpoint* (responsible for configuring rules in the devices) and *DeleteEndpoint* (that performs the rule deconfiguration). The performance of both methods for the Service Handler specialized in L3VPNs using Emulated devices is very similar (Figure 29). This behavior can be expected given the Emulated driver stores and retrieves the rules directly from an in-memory storage. This experiment validates the rest of the components' stability and the overhead introduced by the TeraFlowSDN controller. Figure 29 showcases that the minimum consumed time is 100ms, and 95% of the times it does not exceed 1 second.

Figure 30 plots the performance of the *AddDevice* and *ConfigureDevice* methods of the Emulated Device Driver in the Device component. The *ConfigureDevice* is responsible for configuring and deconfiguring the rules on the devices through the device drivers, thus its performance directly affects the performance of *SetEndpoint* and *DeleteEndpoint* methods of the Service component. In the topology used for this validation, the paths generally traverse, on average two devices. This explains why the time consumed by the *SetEndpoint* and *DeleteEndpoint* methods is roughly twice the time

consumed by the *ConfigureDevice* method (in average two devices need to be configured for each service). In this case, the *AddDevice* method is irrelevant because it is used only during the on-boarding of the devices, but included here for the sake of completeness.

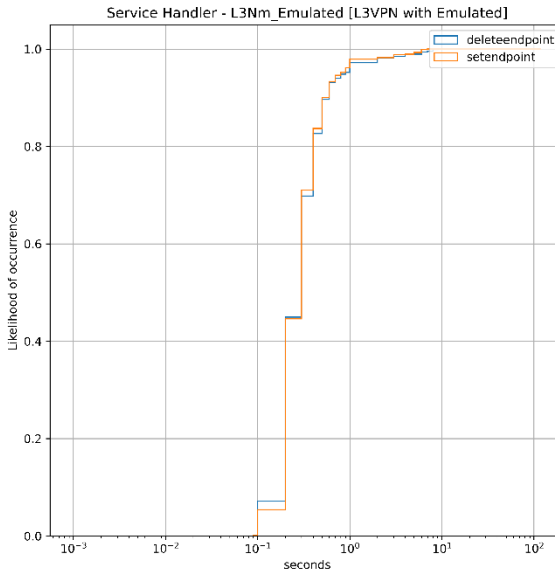


Figure 29. L3NM Emulated Service Handler – Set/DeleteEndpoint and DeleteEndpoint methods

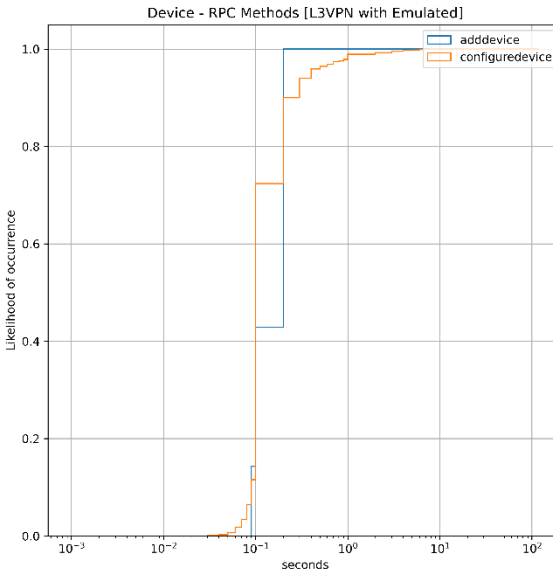


Figure 30. Device component – RPC methods

Next, we analyze the performance of the *ConfigureDevice* method of the Device component. Figure 31 showcases the details on internal operations performed by the *ConfigureDevice* method. The *total* time is the sum of the *wait_queue* time and the *execution* time. The other measurements are contributors of the *execution* time. As can be seen from the figure, the *wait_queue* time is in the order of microseconds in 90% of cases, and below 100ms in 95% of executions. In some edge cases it can reach a second when the device is blocked serving other (de)configuration requests. The time for *configure_rule* and *deconfigure_rule*, i.e., the actual configuration of rules in the in-memory storage of the Emulated driver, are also negligible. The time consumed is then dominated by the accesses done to the Context database to retrieve the known state of the device (*get_device*) and updating that state (*set_device*) in terms of known configured rules. These times are aligned with those reported in Figure 32 for the Device-related RPC methods in the Context component. Note that other components and operations in the TeraFlowSDN controller use the methods of *SelectDevice* and *SetDevice*; that explains the slight variations between both plots.

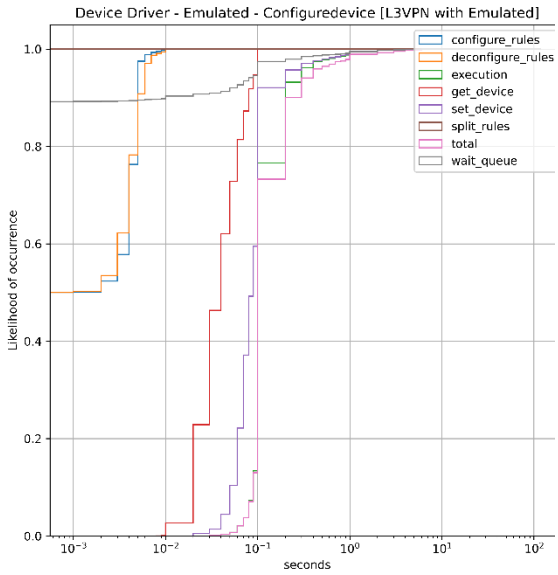


Figure 31. Device component – ConfigureDevice internal operations

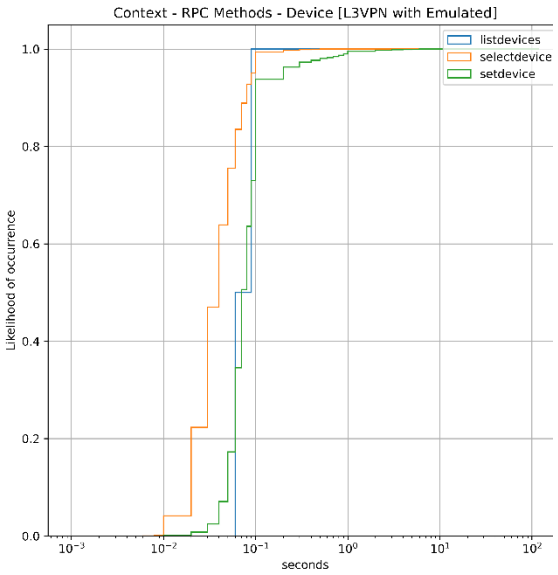


Figure 32. Context component – Device-related RPC methods

4.3.3.3. End-to-end Service Setup and IETF L2VPN SBI

This sub-section describes the setup of an end-to-end service in a multi-technology network scenario, as illustrated earlier in this section, in Figure 6. The control and management plane consists of a hierarchy of SDN controllers, each responsible for a sub-domain/sub-technology. Figure 33 illustrates the multi-domain and multi-technology topology seen at the parent TeraFlowSDN controller instance.

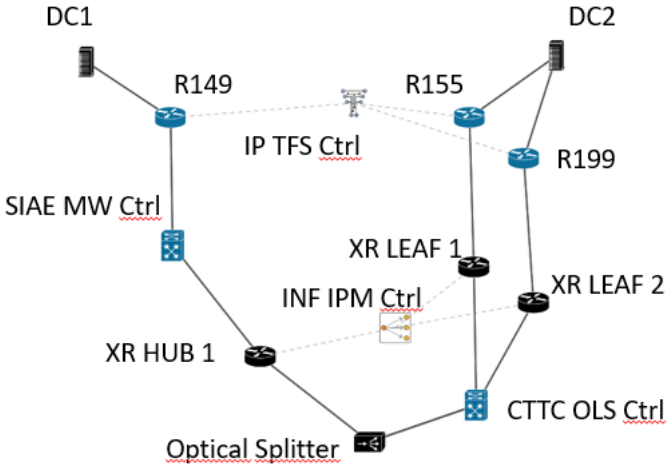


Figure 33. Multi-domain and multi-technology network topology seen at the parent TeraFlowSDN controller instance

The parent TeraFlowSDN instance manages: *(i)* a child IP TeraFlowSDN controller ("IP TFS Ctrl" in the figure) in charge of controlling the packet routers R149, R155, and R199; *(ii)* the SIAE MW controller in charge of controlling a microwave link; *(iii)* the Infinera IPM controller in charge of managing the XR transceivers; and *(iv)* the CTTC Open Line System (OLS) controller that manages the underlying optical mesh.

When a new DC-to-DC L2VPN service is requested, the parent TeraFlowSDN controller computes the overall end-to-end path and decomposes the end-to-end service into sub-services. Figure 34 depicts the DC-to-DC end-to-end service (the first one in the list) and the sub-services (the following four services). The parent TeraFlowSDN controller delegates the setup of the sub-services to the underlying child controllers.

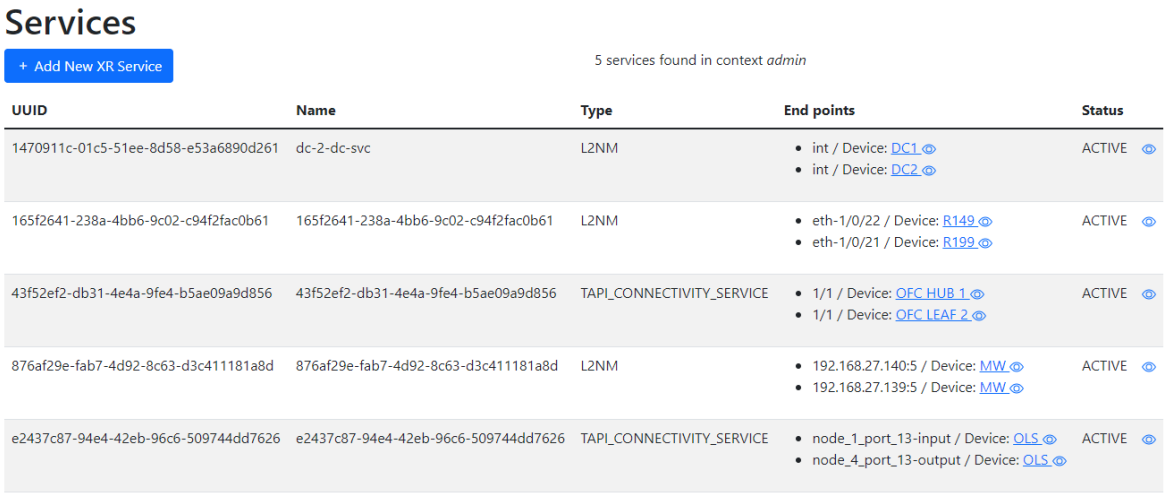


Figure 34. End-to-end service and sub-services

Specifically, the parent TeraFlowSDN controller makes use of the IETF L2VPN SBI driver to interact with the child TeraFlowSDN controller. The second acts as IP controller and is responsible for configuring the L2VPN between the packet routers, in this scenario, R149 and R199. The traffic of this L2VPN traverses the rest of the underlying multi-technology domains. The configuration messages used to delegate the L2VPN creation from the parent to the child TeraFlowSDN controller are listed in Figure 35.

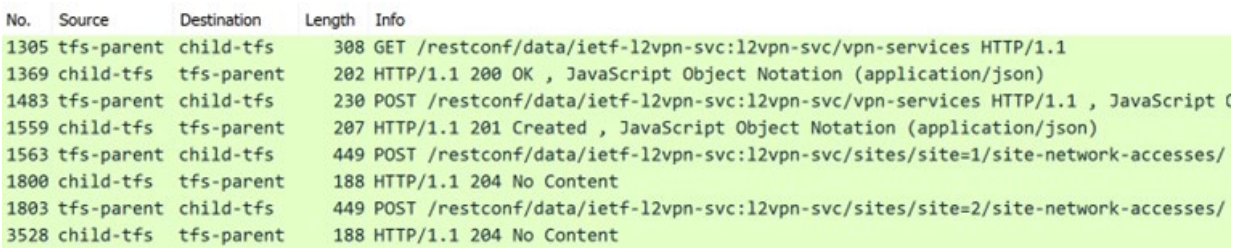


Figure 35. IETF L2VPN SBI messages

This message sequence starts with a GET request (messages 1305 and 1369) to identify existing VPN services. Figure 36 depicts the content of the reply. In this case, no VPN services exist.

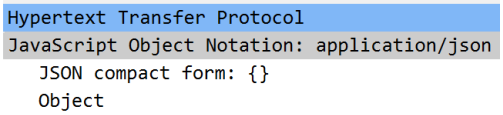


Figure 36. Detail of GET reply – No VPN Services created

When the SBI driver identifies there is no usable VPN service, it creates a new one in two steps. First, an empty VPN service is created to define a unique identifier for the new service (messages 1483 and 1559 in Figure 35). Figure 37 lists the content of the request to create the empty VPN service.

Figure 37. Detail of IETF L2VPN Create VPN service

When the unique identifier for the new VPN service is fixed, the required sites (i.e., the endpoints to be interconnected by the VPN) are added to the VPN service (messages 1563 to 1800 for site 1, and 1803 to 3528 for site 2 in Figure 35). Figure 38 and Figure 39 detail, respectively, the addition of VPN sites 1 and 2. Note that in this scenario, site 1 corresponds to router R149 and DC1, while site 2 corresponds to R199 and DC2.

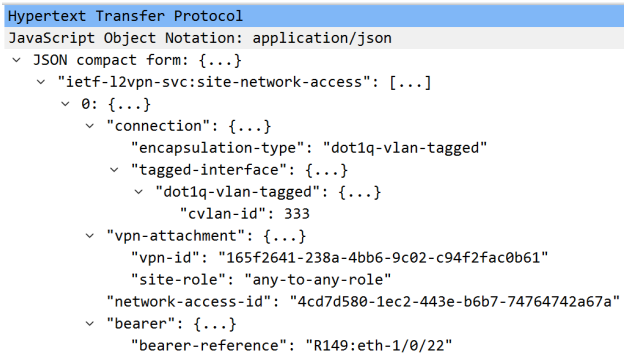


Figure 38. Detail of IETF L2VPN Add site 1 to VPN service

The execution time of this procedure depends on three main factors: *(i)* the execution times for L2VPN services, *(ii)* the time to configure real devices through the SBI drivers and the intermediate controllers (when needed), and *(iii)* the round-trip-time between the premises where the parent and child TeraFlowSDN controllers, intermediate controllers, and physical devices are deployed. For this reason, for the sake of avoiding repetitions and given the execution time components *(i)* and *(ii)* are deeply studied and reported in Sections 4.3.3.1 and 4.3.2, respectively. We point the reader to the respective sections to extract the performance evaluation components.

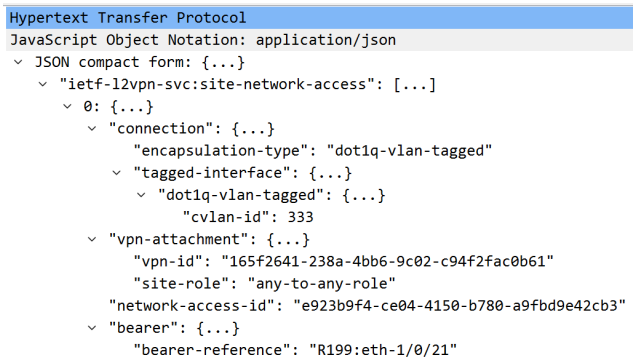


Figure 39. Detail of IETF L2VPN Add site 2 to VPN service

4.3.3.4. Integration with ETSI OpenSource MANO

The results of this workflow have already been reported in D5.2, Section 5.5.2. For completeness, we summarize the main results of this integration in this deliverable. The architecture used for this workflow is depicted in Figure 40. It shows two geographically-distant data centers (acting as Virtual Infrastructure Managers - VIMs) to be interconnected through a transport network slice using a WAN Infrastructure Manager (WIM). 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 the data net and DC2. A transport network slice is deployed over a network connectivity service to satisfy the communication requirements between the two data centres.

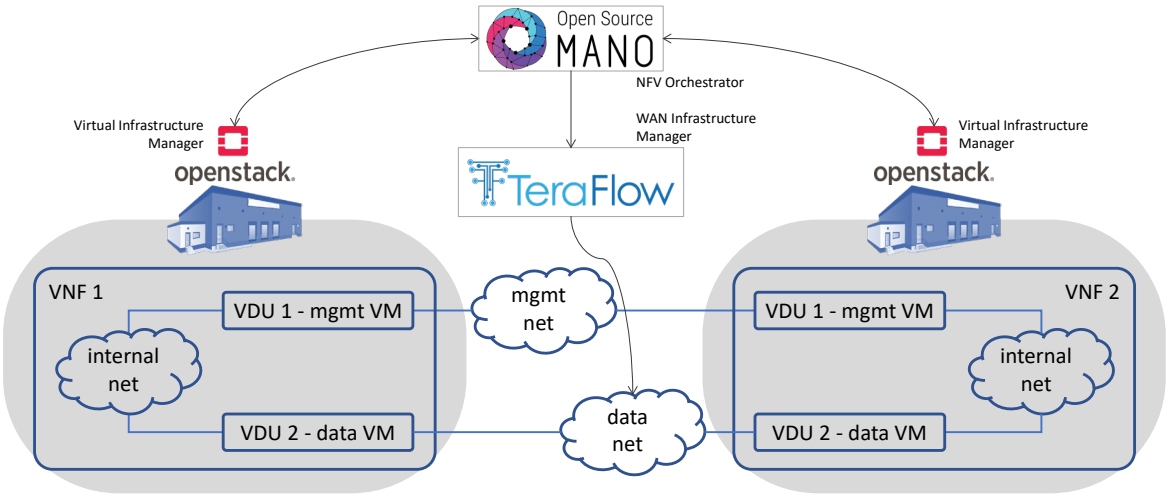


Figure 40. Integration of NFV-O and Transport SDN Controller

We implemented 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 it to the Service component, 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 the Service component 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.

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 41.


```
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 41. Example of IETF-L2VPN-svc:site-network-access

4.3.3.5. Packet layer traffic monitoring using Grafana

For the performance evaluation of L2/L3 VPN services, we adopted a simple emulated network topology composed of four routers forming a ring topology, and two clients connected to two different routers. The emulated devices are on-boarded in TeraFlowSDN, and a new service is created, allowing the two clients to exchange traffic. The two routers connecting the clients are denoted as R1-EMU and R2-EMU. The traffic generation follows a random cyclic pattern.

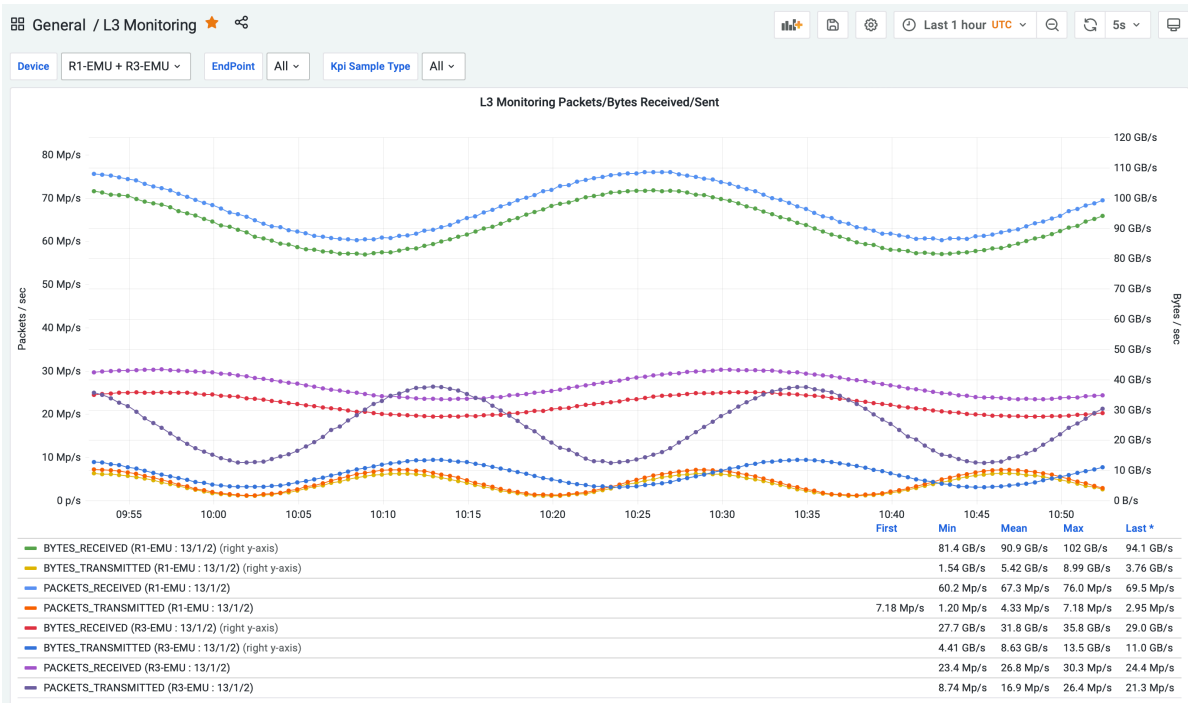


Figure 42. Grafana dashboard illustrating the traffic monitoring

Figure 42 shows a screen capture of the Grafana dashboard developed for this workflow. At the top left, three selection inputs allow the user to select any set of devices, endpoints, or KPIs to be shown

in the plot. In the main area, the traffic traversing the devices/endpoints over time is shown, followed by a summary of statistics in the table below. This dashboard allows TeraFlowSDN adopters to analyze the current traffic flexibly and promptly.

4.3.3.6. Final KPI Measurements

Table 2 summarizes the relevant KPI measurements related to this section.

Table 2. L2/L3 Service Setup/Teardown KPIs

Name	Metric	Value (min/p80/max)	Comment
Service setup delay	Overhead L2VPN	100 ms / 150 ms / 1+ sec	Overhead contributed by the controller. Measured using an Emulated Topology.
	Overhead L3VPN	100 ms / 1 sec / 4+ sec	
Service teardown delay	Overhead L2VPN	90 ms / 140 ms / 1+ sec	
	Overhead L3VPN	100 ms / 1 sec / 4+ sec	
Data rate		100G	Data rate can be shown in Grafana. Available data rates depend on the topology and network equipment, so we are limited to the transponders available in whiteboxes (e.g., 100G).

4.3.4. Slice Grouping

This section's evaluation is split into two main sections; first, an experimental evaluation is carried out; second, the performance of the slice grouping technique is assessed by means of an ad-hoc simulator.

4.3.4.1. Experimental Assessment

The results of this proposed use case on slice grouping have been submitted and will be demonstrated at [OFC23a], including hierarchical control of the underlying network technologies.

Figure 43 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 43. Example of slice templates

Figure 44 provides an example of a slice request. The requested slice includes a service-id and 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 two metrics are considered in this work, but the network slice definition is flexible enough to support multiple SLO/SLE requirements.

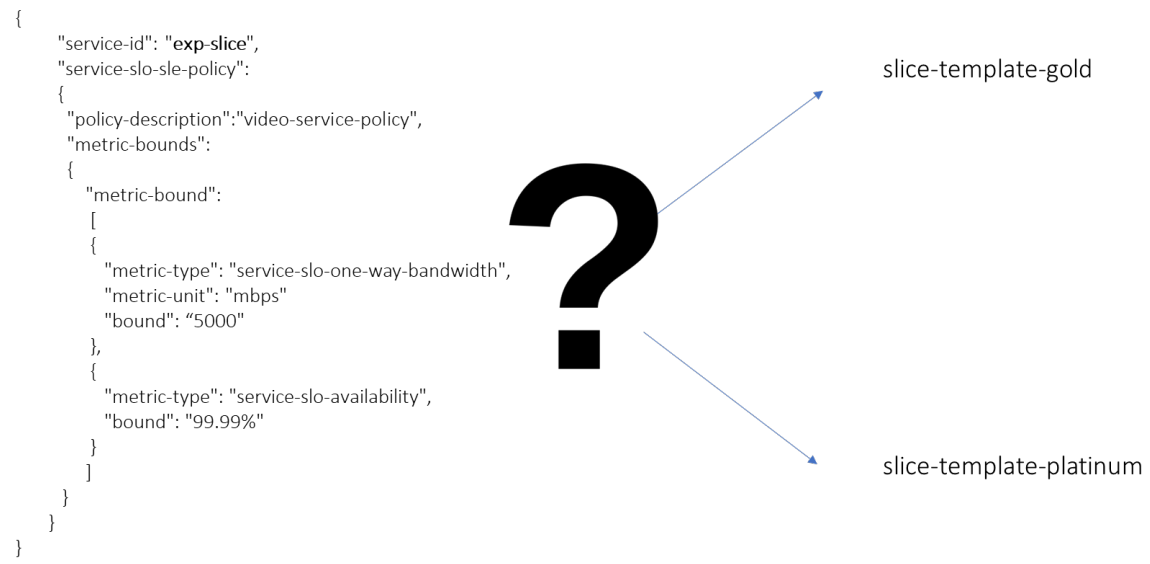


Figure 44. Applying slice grouping on new slice request depending on previously deployed slices

We use the K-Means clustering algorithm to support the slice grouping based on the requested SLO/SLE. This unsupervised machine learning algorithm groups data into a pre-determined (i.e., K) number of clusters. The user defines this number, and the K-Means algorithm groups the data into

that specific number of clusters. This is 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 several 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.

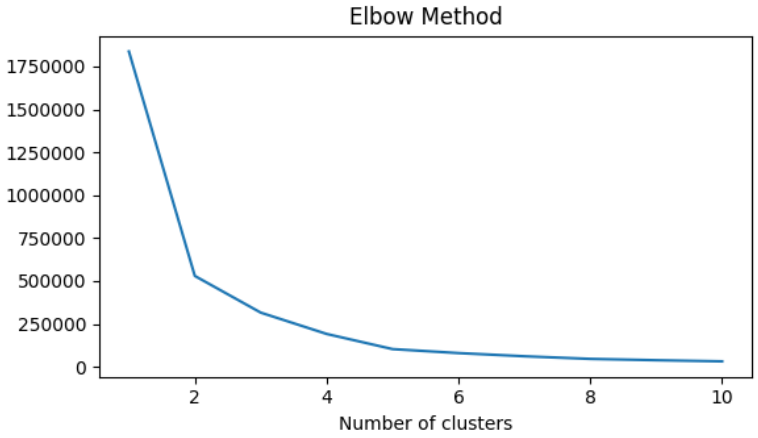


Figure 45. Elbow method applied to slice grouping

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).

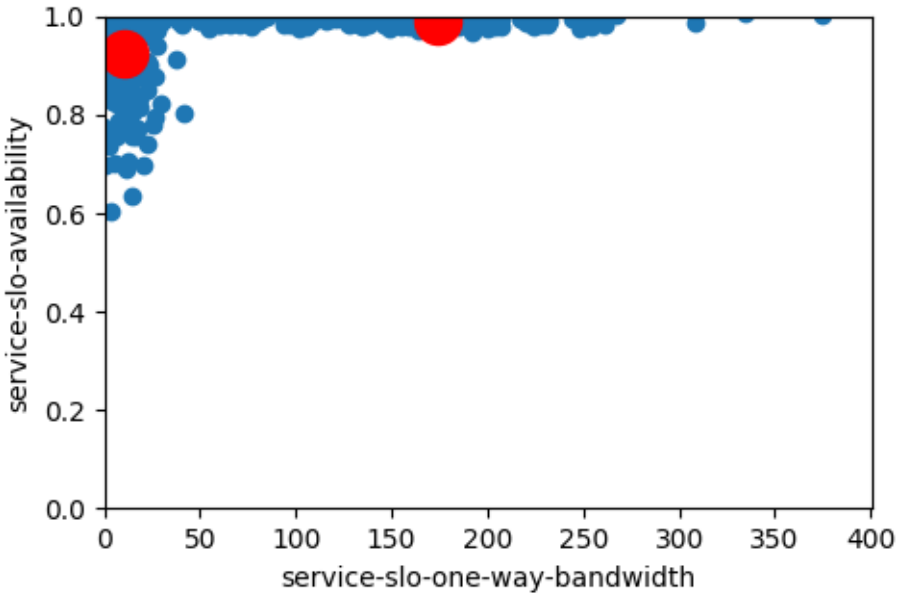


Figure 46. Allocated network slices and their slice groups

4.3.4.2. Slice Grouping Performance Evaluation

To assess the performance of the Slice Grouping technique, we developed an ad-hoc discrete event simulator implementing the same slice grouping technique used in the TeraFlowSDN controller. The simulator tracks each device's switching capacity and each port's capacity. Whenever a new request violates any of these capacity constraints, that request is blocked.

When activating the slice grouping technique in the simulator, the path computation is done exactly as for the classical approach without slice grouping. The procedure differs in 3 aspects. First, the procedure computes the associated slice group for the new requested slice, based on the k-means algorithm. Second, no configuration is performed during the provisioning phase if a traversed device is already configured for the computed slice group. Third, if a traversed endpoint is already configured for the computed slice group during the provisioning phase, no configuration is performed for that endpoint.

For the sake of simplifying the simulation and without loss of generality, we assumed all the requests issued to the simulator have no strict isolation constraints, meaning they can be safely grouped with other existing slice groups, provided they do not violate any capacity constraint.

The simulator reports statistics on: (i) the number of blocked connections and the overall blocking probability, (ii) the total number of rule installations carried out during the simulation, and (iii) the aggregation (avg and max) of the number of configuration rules installed instantaneously per device.

The simulator supports two transport network slices: L2 and L3 VPNs. That way, it enables us to compare the slice grouping technique for different slice types. Configuring L2 services and slices requires fewer configuration rules than configuring L3 ones. The reason for that is that in L2, we only need to create the network instance for the service and perform a configuration on the virtual circuits. However, for L3 VPNs, it is needed to configure a virtual routing function for each service with independent configurations for its routing tables, e.g., static and BGP routing policies. Besides, even in a device/endpoint is already configured, while it needs to support a new request, at least 1 configuration rule needs to be executed to increase the capacity of the associated endpoint/network instance; for this reason, we consider both the case of unconfigured and already configured devices and endpoints.

Table 3 summarizes the number of configuration rules to be installed in the devices according to the transport network slice kind and whether the device/endpoint is already configured or needs full configuration. We extracted this information from the TeraFlowSDN controller's Service component; in particular, the L2NM-OpenConfig and L3NM-OpenConfig Service Handlers that define the set of rules to configure in each network device involved in a specific connectivity service. For instance, configuring a L3VPN network instance in a device, and adding 3 endpoints to this network instance implies installing, in total, 12 + 2*3 = 18 configuration rules.

Table 3. Slice Grouping Simulator – Configuration rules per slice type

	L2 VPN	**L3 VPN**
Rules per Unconfigured Device Network Instance	2	12
Rules per Unconfigured Endpoint added to the Network Instance	2	2
Rules per Configured Device Network Instance	1	1
Rules per Configured Endpoint added to the Network Instance	1	1

We also configured the simulator with 4 slice groups (Bronze, Silver, Gold, and Platinum) with the parameters reported in Table 4. The simulator maps the received request to the closer group for each group based on availability and capacity parameters.

Table 4. Slice Grouping Simulator – Slice Groups Configured

Slice Group Name	Availability	Capacity
Bronze	10 %	10 Gbps
Silver	30 %	40 Gbps
Gold	70 %	50 Gbps
Platinum	99 %	100 Gbps

We configured the Telefonica Spain network (14-node, 44 links) depicted in Figure 47. Each location had a packet router supporting 200 Tb/s of switching capacity. Besides, each trunk endpoint connected to an adjacent packet router is composed of 5x 800 Gbps links aggregating their capacity, resulting in 4 Tb/s.

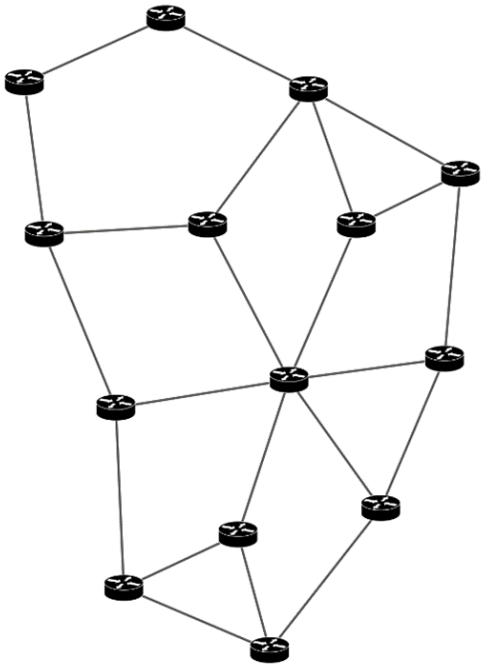


Figure 47. Telefonica Spain Network (14-node, 44-link)

We configured the simulator to run 56 scenarios with variable offered load (from 10 to 10k Erlang) and the slice request kinds varying between L2VPN and L3VPN. We fixed the holding time of the requests to 60 seconds and configured the generator to issue 5k requests for each scenario. For each request, the endpoints have been uniformly chosen. The availability constraint is chosen uniformly in the 0.01 – 99.99 % range, while the capacity for each request is chosen uniformly in the 1 – 100 Gbps range. Each individual scenario has been repeated 5 times with different seeds.

Figure 48 plots the blocking probability (in percentage) of the requests as a function of the offered load, while Figure 49 showcases the total number of rules configured during the overall simulation. In both figures, the solid lines depict the blocking probability when activating the slice grouping technique, while the dashed lines are used as reference values with the slice grouping technique deactivated. The green and orange plots correspond to the L2VPN and L3VPN experiments. No

significant blocking probability is appreciated until the offered load reaches 600 Erlang. When using the slice grouping technique, the blocking probability asymptotically reaches 0.7% while evaluating the maximum considered offered load of 10k Erlang. It is interesting to note that the slice grouping technique increases a bit the blocking probability but nothing significant with respect to the classical approach, e.g., the increase is in the range of 0.02-0.03%.

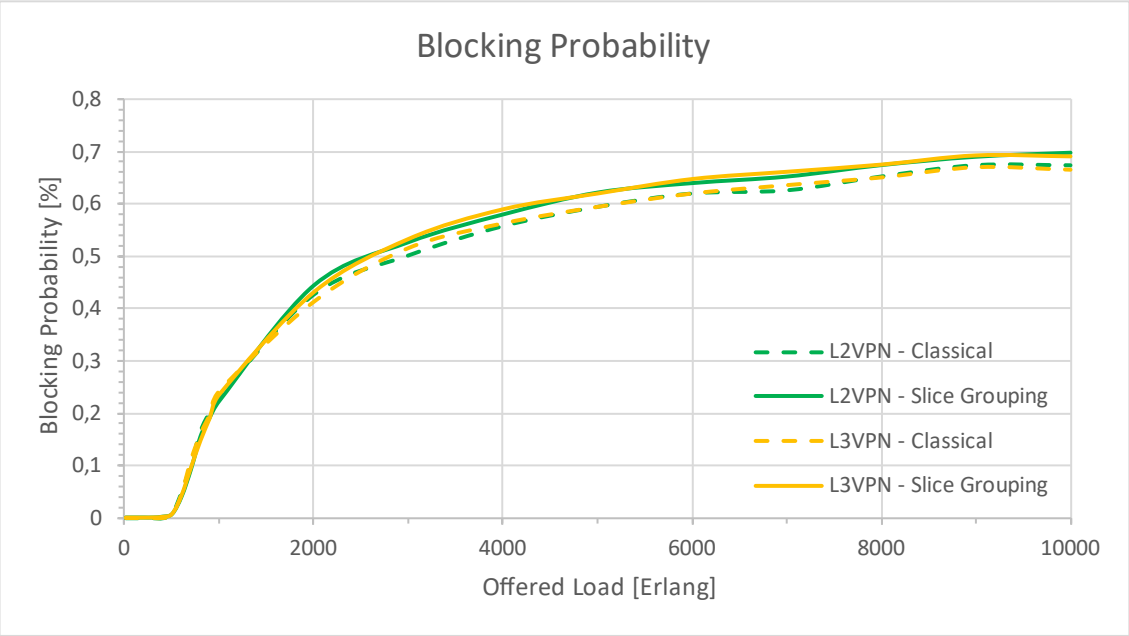


Figure 48. Blocking Probability

Regarding the total number of rules configured during the simulation, the number of rules configured using the classical approach is fixed around 80k rules for the L2VPN slice, and 240k for the L3VPN slice. The peak happening at 600 Erlang is associated with increased blocking probability. At that point, the simulator cannot set up new requests due to the lack of capacity resources, thus blocking them and reducing the number of rules installed in the devices.

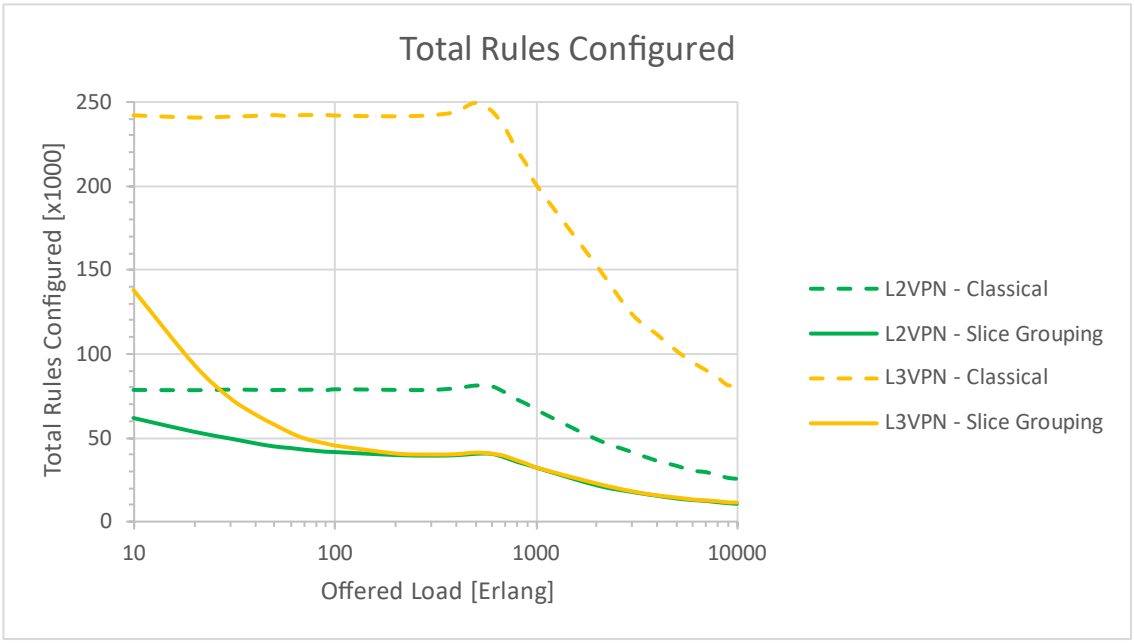


Figure 49. Total Rules Configured

When compared to the results obtained using the slice grouping technique, it is interesting to appreciate that, by reusing already installed configuration rules, the slice grouping technique saves an increasing number of rules while the load increases, asymptotically reaching the value of 40k rules at 600 Erlang, where the blocking probability starts to increase, and the number of rules installed starts to decrease, as it happened with the classical approach.

We illustrate the instant number of rules configured per device for L2VPN slices in Figure 50 and for L3VPN slices in Figure 51. In both cases, the classical approach, depicted in grey, is used as a reference, while the approach using the slice grouping technique is depicted in black. In both cases, we plot the instantaneous average number of configuration rules with a solid line, while the dashed line corresponds to the maximum value observed over all the repetitions for a specific scenario configuration. The blue plot depicts the reduction factor between the classical approach and the slice grouping approach; this plot is associated with the axis on the right side.

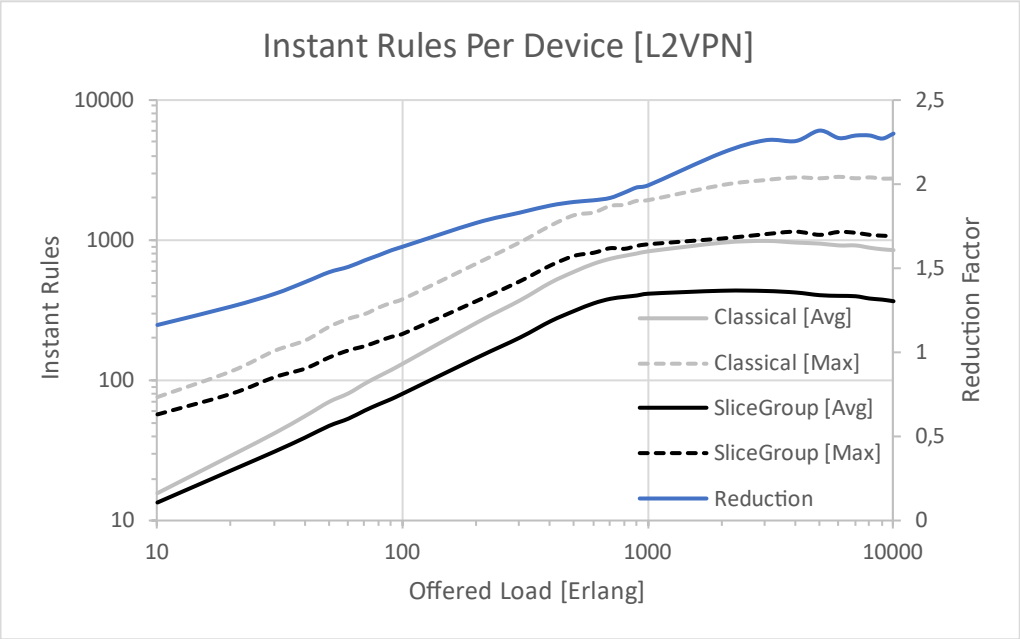


Figure 50. Instant Rules per Device - L2VPN

It is worth noting that thanks to the slice grouping technique, the number of rules configured per device is reduced. Studying the results in Figure 50, using the classical approach for L2VPNs, the average number of rules installed per device along the simulation reaches 990 rules, with a maximum value of 2.8k rules. In contrast, using the slice grouping technique, the average value reaches 440 rules, with a maximum value of 1.1k. Comparing the average number of rules installed one-by-one per offered load, we found that slice grouping enables reductions of up to 2.3x for the L2VPN slices. It is worth noting that the number of rules installed per device stabilizes after reaching the value of 600 Erlang of offered load, i.e., when the blocking probability starts to increase.

Even better results can be found by studying the results for L3VPNs in Figure 51. The average number of rules installed per device along the simulation using the classical approach reaches 2.9k rules, with a maximum value of 8.2k. In contrast, using the slice grouping technique, the average value reaches 480 rules, with a maximum value of 1.1k. Comparing the average number of rules installed one-by-one per offered load, we found that reductions of up to 6.4x can be achieved. As in the L2VPN case, the number of rules installed per device stabilizes when the increase of blocking probability becomes significant.

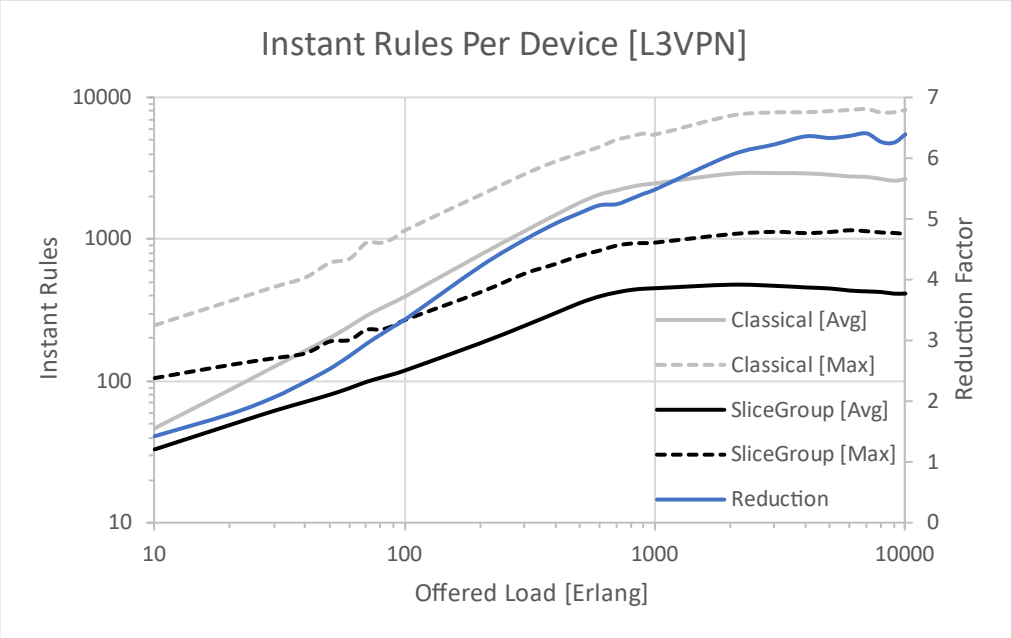


Figure 51. Instant Rules per Device - L3VPN

4.3.4.3. Final KPI Measurements

Table 5 summarizes the obtained results.

Name	Value	Comment
Resource efficiency reduction factor	L2VPN 1,9	At offered load of 600 Erlang (when blocking probability arises)
	L2VPN 2,32	At offered load of 5k Erlang (peak resource efficiency)
	L3VPN 5,22	At offered load of 600 Erlang (when blocking probability arises)
	L3VPN 6,4	At offered load of 7k Erlang (peak resource efficiency)

Table 5 Slice Grouping KPI measurements

4.3.5. Policy-driven Service Restoration with P4 devices

This section demonstrates a policy-driven service restoration scenario that involves P4 devices, an overlay service, and TeraFlowSDN's policy layer for maintaining the SLA of a deployed service.

4.3.5.1. Concept

This scenario spans three TeraFlowSDN controller layers, as shown in Figure 52. The device layer leverages the SBI component to interact with the underlying network devices through the P4Runtime API and the Monitoring component for collecting network telemetry. Managing a network at the level of individual devices requires network administrators to master highly complex and technology-specific device configurations. For this reason, this scenario introduces additional components atop the device layer, as shown in Figure 52.

This tiered segregation offers two powerful abstractions:

- A service layer that allows network administrators to describe end-to-end connectivity between any two endpoints through high-level intents.
- A management layer that allows network administrators to associate end-to-end connectivity services with run-time network policies.

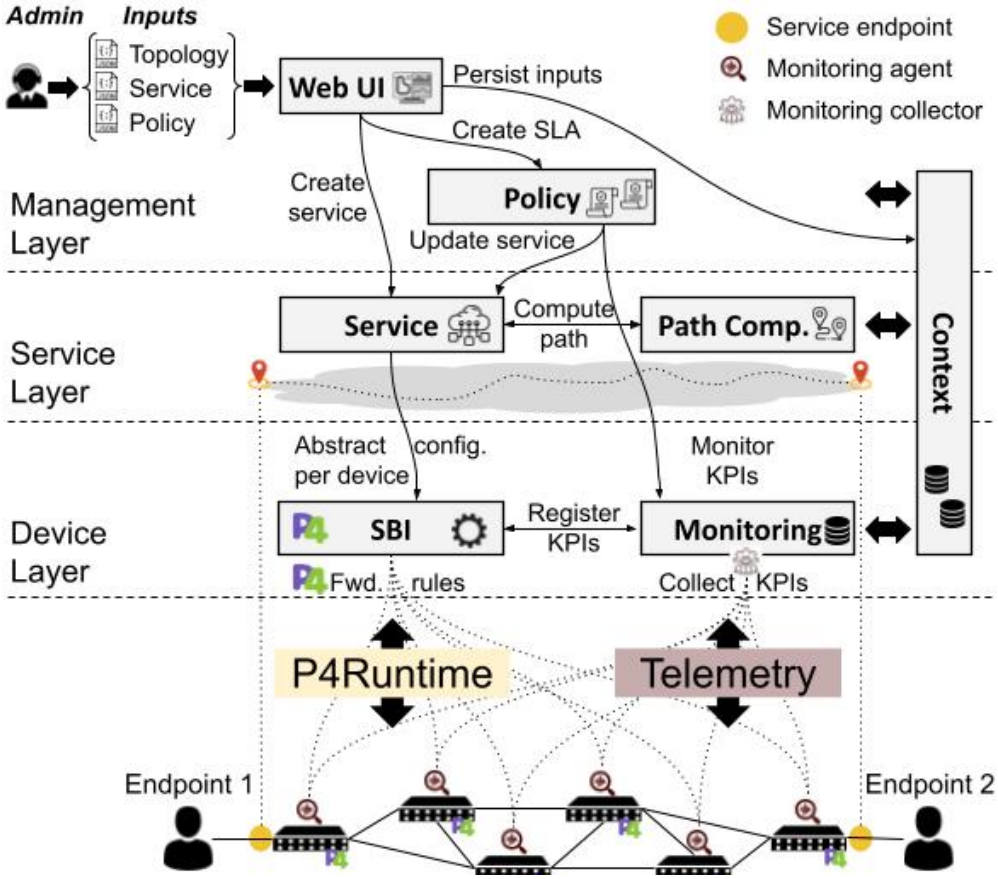


Figure 52. Policy-driven service restoration architecture within TeraFlowSDN.

Abstraction #1: **Abstract end-to-end services translated into P4 configuration.** Upon a service request by a network administrator (see Figure 52), the Service component receives a request to provision connectivity between two remote endpoints from the Web UI. First, the Service component requests a path between these endpoints - highlighted in yellow in Figure 52 - from the Path Computation component. Then, the Service component configures the underlying network devices along the path between the endpoints through the SBI. To keep the Service layer agnostic from technology-specific details, the Service leverages a minimal service definition (see Table 6) that allows users to express what they want to connect, while letting the underlying system decide how to realize the connection. The Service component translates this minimal service definition into an abstract device configuration model that, in turn, is automatically translated into P4 rules from the P4 device driver of the SBI component.

Table 6. Example end-to-end service definition for P4-based connectivity between two endpoints.

```
{
   "service_type":"P4",
   "service_endpoint_ids": [
      { // endpoint A
         "device_uuid": "SWA", // device ID
         "endpoint_uuid": "X"   // port number
      },
      { // endpoint B
         "device_uuid": "SWB", // device ID
         "endpoint_uuid": "Y"   // port number
      }
   ]
}
```

Abstraction #2: Real-time policies atop end-to-end services. Managing the run-time of an end-to-end service is of paramount importance for network administrators as modern systems become more and more complex. This scenario offers another powerful abstraction, which allows network administrators to associate monitoring metrics stored in the Monitoring database with conditions according to the event-condition-action policy model [Bou2019]. When these conditions are met, the Monitoring component raises an alarm that the Policy component consumes to trigger specific service or device-level actions. The example policy in Table 7 invokes path re-computation for a given service as an action to bind the end-to-end latency of a service below 4ms. Additional actions can be requested for other use cases, such as adding specific service constraints or configurations, in which case the network administrator shall specify an action configuration. This is how the Policy component offers event-driven SLAs for end-to-end connectivity services through P4 pipelines.

Table 7. Example network policy for bounding the end-to-end latency of a service below 4ms. The action simply triggers path re-computation, without additional configuration needed.

```
{
   "service_uuid": "d5261206-1047-00345",
   "policy_rule": {
      "priority": 0,
      "condition_list": [
         {
            "kpi_id": "E2E_LATENCY",
            "operator": "GREATER_THAN",
            "kpi_value": 4000 // in us
         }
      ],
      "action_list": [
         {
            "action": "RECOMPUTE_SVC_PATH",
            "action_config":[]
         }
      ]
   }
}
```

4.3.5.2. Testbed

We use a Mininet-based topology of P4 switches based on the bmv2 [BMV2023] software switch to verify this scenario, as shown in Figure 53. On top of this topology, we deploy the TeraFlowSDN controller using the seven components depicted in Figure 52. The network administrator inputs a list of devices and links in JSON format, which the controller parses and establishes connections with all eight P4 switches through the SBI component. Once device handshaking is completed, the topology is stored in the Context component, thus, the network administrator can proceed with service instantiation as per Table 6. In this demonstration example, the service endpoints are “SW1-port4” and “SW8-port4”, thus, we formulate the JSON accordingly.

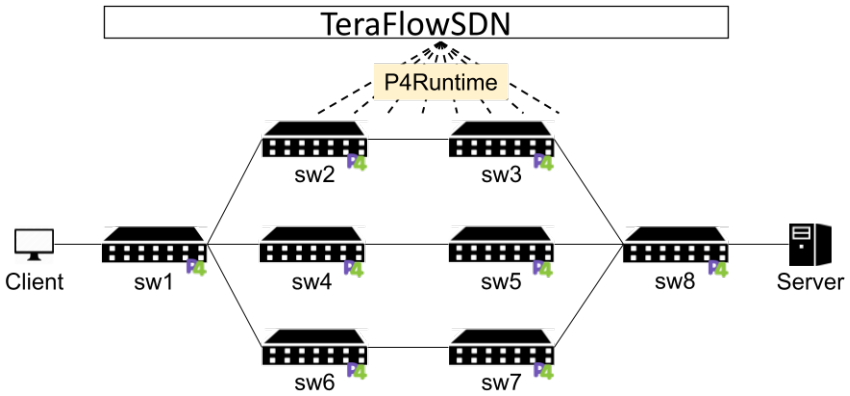


Figure 53. Policy-driven service restoration testbed.

4.3.5.3. Workflow

For this scenario to start, we assume two workflows as prerequisites. First, a topology provisioning workflow (see Figure 54) that ensures that all P4 devices are deployed, and the links between these

devices are established. Second, a service provisioning workflow (see Figure 55) ensures an end-to-end connectivity service is established between client and server through the various P4 switches.

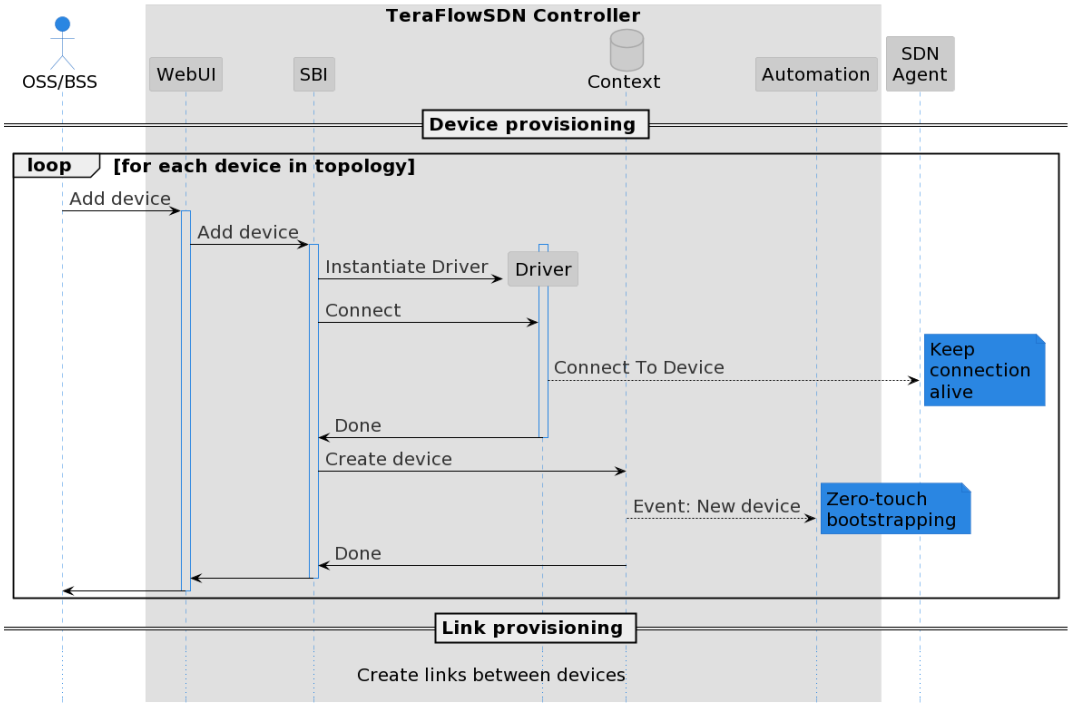


Figure 54. Device and link provisioning as a pre-requisite for policy-driven service restoration.

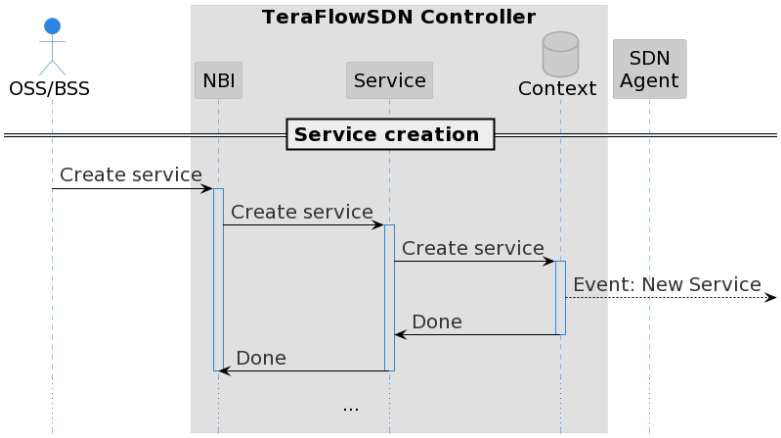


Figure 55. Service creation as a pre-requisite workflow for policy-driven service restoration.

Given that the workflows depicted in Figure 54 and Figure 55 are pre-established, we demonstrate a policy-based service restoration workflow comprising various steps, highlighted in blue in Figure 56. An input policy - similar to the example shown in Table 7 - bounds the end-to-end latency of the deployed service between client and server within a certain threshold. A probe is deployed to monitor the end-to-end latency and report it to the Monitoring component. Initially, we assume the service is established according to the red path shown in Figure 56.

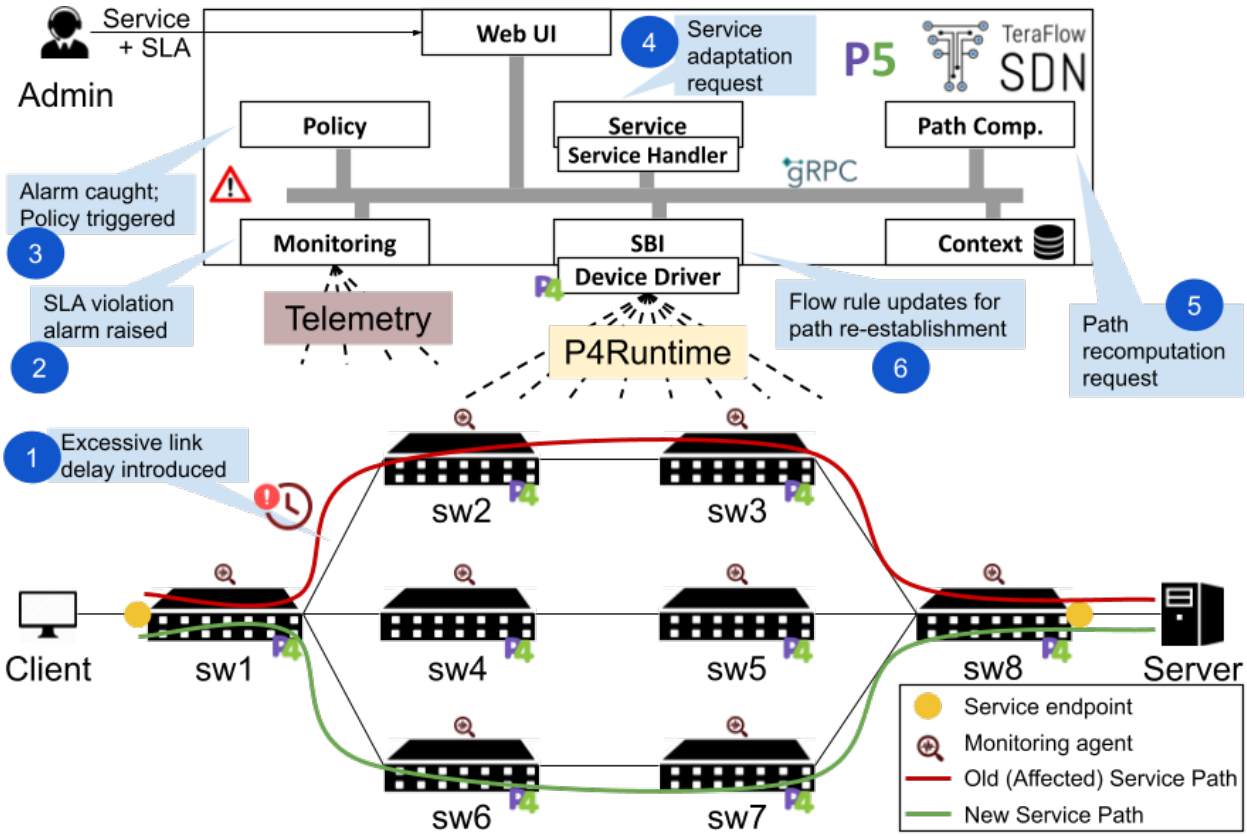


Figure 56. Policy-driven service restoration demonstration scenario.

To validate the policy, we explicitly introduce excessive link latency (using Linux traffic control) along the service path as shown by step 1 in Figure 56. The Monitoring component captures this state change in step 2 and raises the alarm for potential policy violation, which is caught by the Policy component in step 3. This event causes the execution of a policy action, which jointly involves a service update (step 4) through path re-computation (step 5). When the Path Computation component returns a new service path (e.g., the green path in Figure 56), Service compiles a list of device configuration commands for establishing the new path followed by another list of commands for decommissioning the old path. These commands are translated into actual P4 flow rules by the SBI, before being enforced to the data plane via the P4Runtime API (step 6). In the scenario in Figure 56, TeraFlowSDN can pick any path between client and server; thus, the red and green paths in Figure 56 are illustrative. The workflow diagram shown in Figure 57 captures the exchange of messages among the key TeraFlowSDN components participating in this scenario.

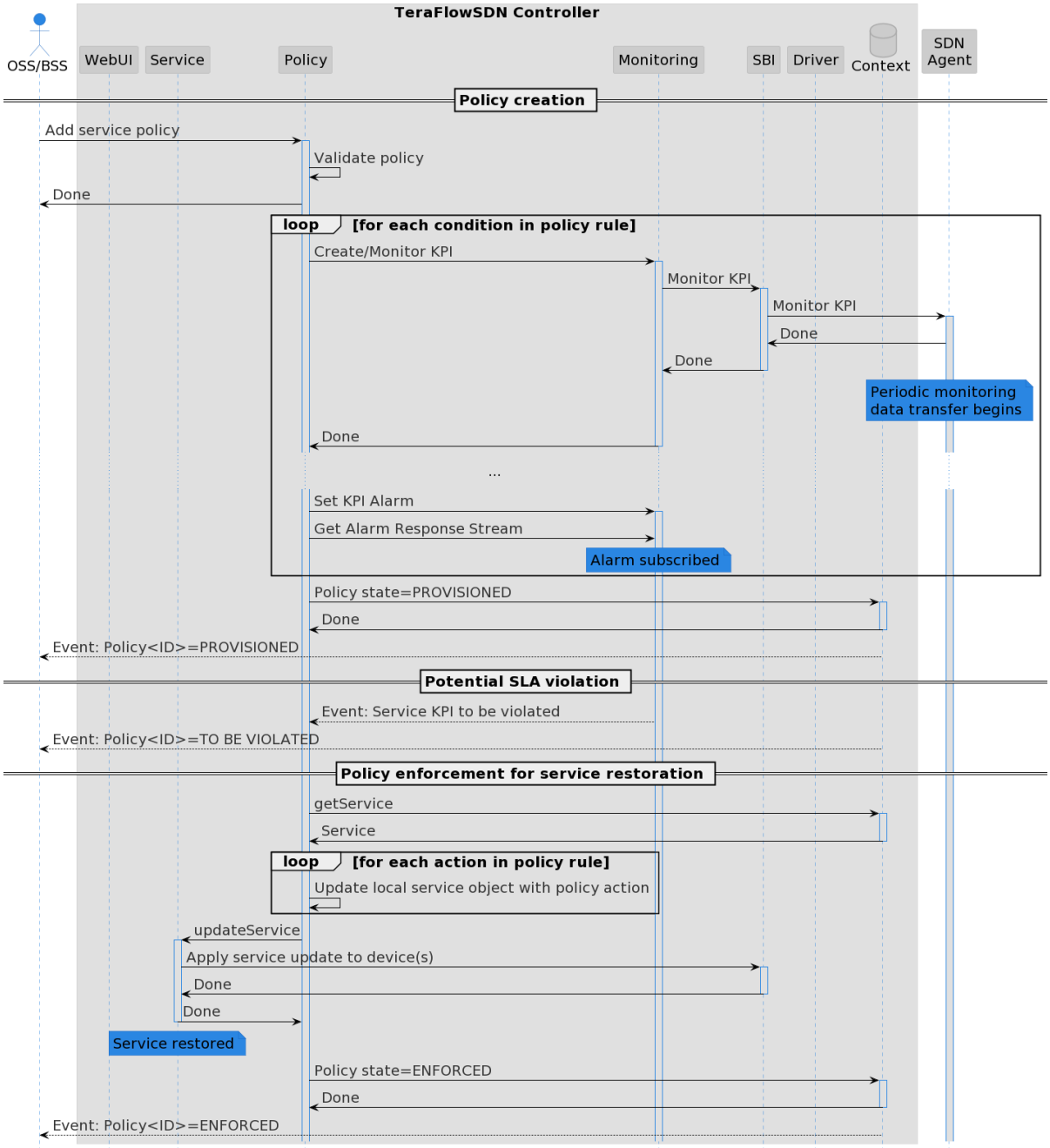


Figure 57. Policy-driven service restoration workflow.

4.3.5.4. KPI Measurements

An indicative view of the TeraFlowSDN dashboard for this scenario is visualized in Figure 58. This dashboard was used for the proof-of-concept demonstration of this TeraFlowSDN functionality [HPSR23]. This dashboard visualizes runtime end-to-end service latency data over time. A latency threshold of 10ms is drawn as a horizontal line, highlighted in orange. When latency exceeds this threshold, the policy is triggered, and the state (i.e., state ENFORCED in Figure 58) of this policy is visualized atop the graph in red.

Figure 58. Dashboard of the Policy-driven service restoration workflow.

Table 8 summarizes the KPIs that TeraFlowSDN preserves during the service restoration workflow presented in this section. This workflow focuses on the end-to-end latency of the connectivity service that is deployed atop the P4 devices. A policy is crafted to monitor the end-to-end latency at runtime; in the case that the latency exceeds a certain threshold, TeraFlowSDN ensures that the service is updated to keep the end-to-end latency within the desired window.

Table 8. Policy-based service restoration KPIs.

Name	**Value or Value Range**	**Comment**
End-to-end service latency	5ms (indicative)	The actual value depends on the topology and hardware adopted. For example, hardware switches are faster than software switches, while software switches perform better on high-end Commercial off-the-Shelf (COTS) hardware. Therefore, this value may vary.
Reaction time to ensure SLA	~4s	The reaction time can be further broken down into RPC calls (approx. 1s) and path recalculation time (approx. 3s). It should be noted that the path recalculation time is dependent on the topology. Larger topologies may require more time than smaller topologies to find new paths.

4.3.6. Energy-Efficient Path Computation

This section aims to validate and evaluate the performance of the Path Computation component when dynamically handling network connectivity services relying on a devised energy-aware routing (EAR) algorithm. The proposed algorithm is based on the traditional *K*-shortest-paths mechanism (K-SP), based on the Yen algorithm. Upon receiving a network connectivity service request (between a pair of network nodes with specific requirements in terms of bandwidth and maximum tolerated end-to-end latency) the implemented algorithm seeks for the spatial path (i.e., nodes and links) that fulfils the request needs while minimizing the overall network power consumption. The conducted evaluation benchmark the EAR algorithm with the K-SP algorithm, where the latter routes connectivity services to meet their requirements whilst minimizing the overall network resource utilization (i.e., link bandwidth). The performance comparison is carried out upon dynamic creation and deletion of network services with heterogeneous bandwidth (data rates) and latency demands. The performance indicators used for comparison are: the blocked bandwidth ratio, the average amount of consumed network energy, the average network throughput, and the energy efficiency (defined as the consumed network energy over the average network throughput).

4.3.6.1. Transport Network Energy Model

Before tackling the implemented EAR algorithm, the transport network topology, and the experimental scenario/assumptions to generate the network connectivity requests, it is worth briefly reviewing the adopted energy network model, which was thoroughly discussed in D4.2. This energy model has been traditionally used and applied within packet-switched transport networks. The consumed network energy is linked to two main contributors: i) devices (i.e., switches or routers) and ii) their ports when transmitting and receiving data packets. This is why, at any given time, the consumed network energy is tied to the amount of "running" network connectivity services and the amount of data traffic being transported. In this context, network devices are based on the well-used *store-and-forward* switch architecture (see D4.2). This architecture defines a network node as formed by a pool of line-cards containing a determined number of input and output ports. Each port has a Ternary Content Addressable Memory (TCAM) circuitry which allows storing and forwarding of data packets. Thereby, the contribution of the consumed energy of an active port is proportional to the volume of the data traffic being processed/stored/routed. The mathematical expressions modelling the relationship between energy consumed and transmitted data rate are reported in D4.2. On the other hand, a powered-up device (switch) also increases the consumed network energy by the so-called *environment* contribution, e.g., the fans used for cooling the device. Regardless of the data traffic being processed by the device/switch, there is an *idle* consumed energy because the node is powered up, ready to route/switch connectivity services.

As described in D4.2 Section 5, an additive expression can calculate the total network energy consumption. This expression sums up the contribution of all the active devices and the energy consumption contribution bound to all the device ports that transport data traffic. Therefore, the most straightforward approach to reduce network energy consumption is to serve arriving connectivity service requests through the lowest number of active devices and ports.

4.3.6.2. Workflow

Figure 59 depicts the workflow to process an incoming network connectivity service request, select the network resources to accommodate the request and eventually configure the involved network devices within the TeraFlowSDN controller's Components. Specifically, once the Service component receives a network connectivity service from a North-Bound Interface (NBI), it delegates to the Path Computation Component to find a feasible path (i.e., devices and links) which meets the service requirements (i.e., bandwidth and latency). The Path Computation component hosts a pool of algorithms to be executed targeting diverse objectives, e.g., EAR, simple shortest path computation, K-SP, etc. The algorithm to be executed can be notified in the request sent by the Service to the Path Computation component. Prior to triggering the algorithm execution, the Path Computation component retrieves the Context information, i.e., network topology (devices and links) and status of the resources (e.g., available link bandwidth). The result of the Path Computation component is then sent back to the Service component to handle the allocation of the selected resources via the SBI component, which properly programs the devices (and their ports/links) forming part of the computed path.

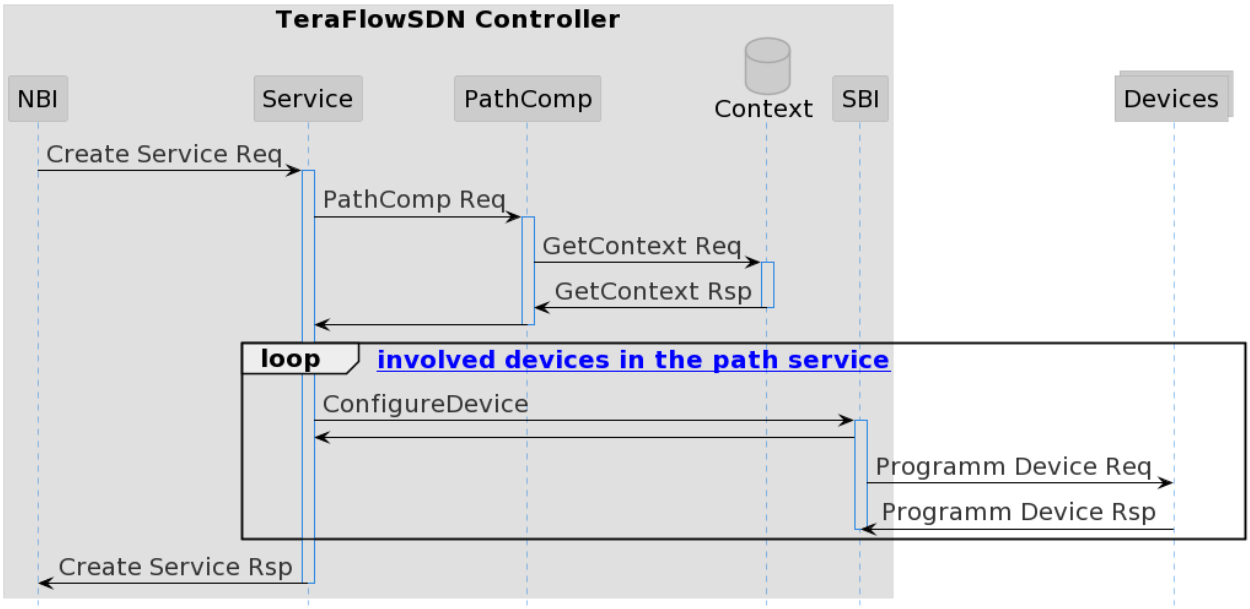


Figure 59: Path Computation serving Network Connectivity Services workflow.

Observe that the workflow is indistinctly applied for the experimental results discussed below regardless of the executed Path Computation algorithms (i.e., either EAR or K-SP).

4.3.6.3. Considered Path Computation Algorithms

The devised EAR algorithm is a heuristic based on a modified K-SP algorithm which pursues two prioritized objectives: O1) to fulfil the service requirements and O2) to reduce the overall network energy consumption. To this end, it explores K shortest paths to fulfil O1 and selects the one which leads to the lowest network energy consumption. To that end, every *k* shortest path is found using an additive link metric, which is attempted to be minimized. In this energy-efficiency context, this metric is referred to as *power_path,* and it includes 1) the *static* device *idle power* and 2) the energy consumed on each used port/link. For the latter, recall that for every candidate device port (link) the computed energy consumption is proportional to the aggregated data traffic volume from the existing network connection services routed through that port. Additionally, every port is featured by an *energy_consumption_parameter*. This is a vendor-defined characteristic which is expressed in nJoules/bit. Bearing this in mind, given a port transmitting/receiving a data rate (in Mb/s) of *dataRate*, its energy consumption can be analytically approached as: *energy_consumption_parameter* x *dataRate*.

As outlined above, the EAR algorithm aims to reduce network energy consumption. To achieve this goal, it is recommended to accommodate new incoming network connectivity services on top of the existing transport infrastructure made up of active devices and links rather than just powering up network elements which are in *sleep* mode (i.e., powered off since no active connections are routed through them). We are assuming that the TeraFlowSDN controller is enabled with a mechanism through which it sets network nodes to *sleep* mode. Once the TeraFlowSDN controller realizes that no network connectivity service is traversing a node, it could handle the programmability operations/commands to enforce *sleeping down* a selected node. We do insist that this mechanism is an assumption made for the EAR algorithm operation. In other words, this capability is not implemented in the TeraFlow SDN controller when writing this contribution but is planned to be done in upcoming releases.

With the above considerations and assumptions, the EAR algorithm follows a two-step computation:

1. Firstly, it attempts to route a connection request over the active subset of the network infrastructure, i.e., only using active devices and ports. To do that, it triggers the K-SP to minimize the power_path and select the *k* path attaining the lowest end-to-end power consumption. If two or more *k* paths have the same end-to-end *power_path*, the tie is broken, choosing the path offering the lowest end-to-delay. If the tie remains, the path having the largest available bandwidth on the most congested link is selected. Otherwise, a path is randomly decided.
2. Secondly, if the routing over the active network subset fails, all the network devices and ports (i.e., both powered up and off) are considered. Then, the same criteria as above are followed, i.e., out of the *k* computed feasible paths, the one with the lowest end-to-end path energy is chosen; If there is tie between two or more paths, as above, the delay and the available bandwidth policies are adopted.

For the sake of comparison, a regular K-SP algorithm is used where the computed *k* paths are sorted by i) the number of hops, ii) end-to-end delay, and iii) the available bandwidth. Note that with this algorithm, no end-to-end energy path minimization is targeted. Once the path is chosen, the resulting *power_path* it is computed for comparison purposes with the EAR approach.

4.3.6.4. Experimental Scenario: Topology, Connectivity Service Request, and Performance Metrics

Transport Network Topology

The transport network topology to conduct the performance evaluation is shown in Figure 60. This is an emulated transport network formed by 14 devices (packet switches), and 42 bidirectional links. Devices' ports are interconnected via bidirectional point-to-point links with a total transport capacity of 10 Gb/s. Different network connectivity services can be transported through the same link exploiting the statistical multiplexing capability of packet-switched networks if the maximum link transport capacity (i.e., 10 Gb/s) is not exceeded.

The link delay is labelled at each edge depicted in the network topology and expressed in ms. For the energy-related parameters, it is assumed that each "powered-up" device does consume an *idle* power of 90 W. On the other hand, all the ports are characterized by a homogeneous *energy_consumption_parameter* set to 0.6 x 10^-9 Joules/bit.

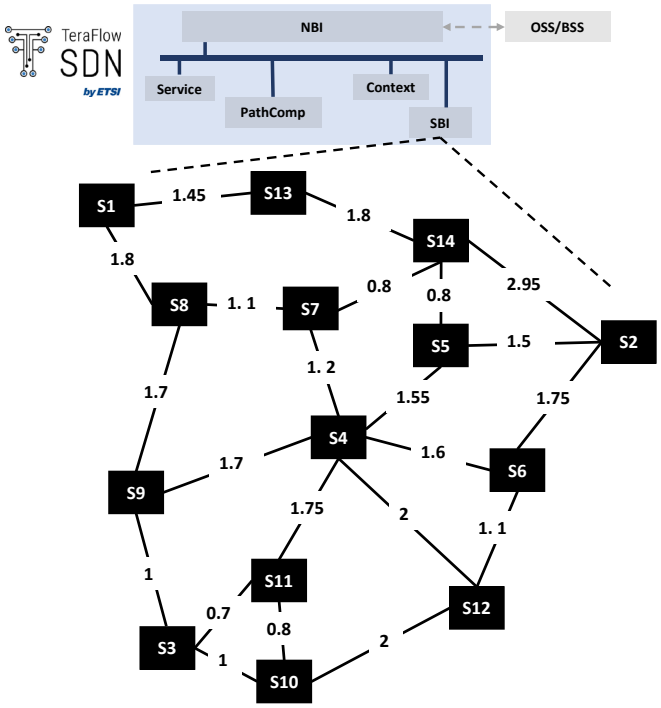


Figure 60: Emulated Transport Packet-Switched Network Scenario.

Dynamic Generation/Termination of Network Connectivity Service Requests

Network connectivity services are assumed to dynamically arrive and depart from the network being handled by the TeraFlowSDN controller component and workflows to conduct an exhaustive comparison of both EAR and K-SP. Creation and termination requests arrive to the TeraFlowSDN controller via the NBI Component (e.g., triggered by an external OSS/BSS). The arrival of the service requests follows a Poisson model whose mean inter-arrival time is set to 10 s. To achieve different traffic intensities, the duration of the service requests follows an exponential function whose mean Holding Time (HT) is varied. For each traffic intensity, 20k connection requests are generated.

All the requested network connectivity services are point-to-point unidirectional connections whose *endpoints* are uniformly chosen among the following nodes: S1, S2, S3, and S12. In other words, all requested connectivity services interconnect any pair of those nodes. The rest of the network nodes (i.e., S4, S4, S6, …) operate as *transit* nodes to enable the physical connectivity for any service. Note that S1, S2, S3 and S12 can act as a transit node if the connectivity service does not initiate and terminate on such a node. The requested bandwidth is uniformly selected from a discrete set [600, 1000, 2000] Mb/s. Finally, each request's latency requirement is randomly chosen from the set [8, 10, 12] ms.

Performance Metrics

The comparison of the two algorithms is done by relying on the following performance metrics:

- **The blocked bandwidth ratio (BBR)**: this metric indicates the amount of bandwidth (b/s) that cannot be accommodated over the total requested bandwidth for all the network connectivity services. The lowest the BBR, the better an algorithm performs in successfully serving network connectivity service requests.
- **Average Consumed Network Energy** (in kW): this metric accounts for the average amount of energy consumed by the network in operation (considering all the network devices and links' energy consumption). Since network connectivity services dynamically arrive and departure

from the network, the metric is iteratively computed whenever a network *event* happens. A network event refers to either a new connectivity service is set up (i.e., link bandwidth is allocated) or when an existing service terminates (i.e., resources are released). Thus, let's assume that a new connection service is established at time *t1*. Then, the whole network's energy consumption is computed considering all the active services. The result is referred to as *NetwPower_t1*. Next, another event takes place at time *t2*, where *t1 < t2*. Note that the network's energy consumption between *t1* and *t2* period is *NetwPower_t1*. However, at t2, *NetwPower_t2* needs to be calculated to account the resource variation. Consequently, to obtain the average network energy consumption, all the individual network resource variations are taken into consideration being weighted by their respective durations through a specified observation time.

- **Average Throughput** (in Gb/s): this metric determines the average amount of data traffic volume being transported in the network for all the connectivity services. Likewise, the average consumed network energy, the throughput metric is re-computed iteratively upon services are established and removed.

4.3.6.5. Numerical Results

The numerical results comparing the performance attained by both EAR and K-SP algorithms, executed at the TeraFlowSDN controller Path Computation Component, are depicted in Figure 61. It is worth mentioning that the obtained results exclusively focused on low-medium traffic intensity. In these traffic conditions, adopting a strategy to save energy consumption has larger flexibility (i.e., the number of feasible routing path candidates) to enhance the overall network energy-efficiency. However, as traffic intensity grows (i.e., HT is increased), the routing algorithms start operating with resources being more occupied. This reduces the candidate set of feasible paths for any incoming service request. Consequently, the differences between EAR and K-SP are less noticeable, since the primary objective of whatever algorithm is first to accommodate services fulfilling their requirements.

For both the EAR and K-SP strategies, K is set to 5. Figure 61.a shows the BBR for average HT ranging from 1000 to 2000. Indeed, K-SP routes/distributes traffic services more uniformly over the network. This, in turn, leads to favour the establishment of the upcoming service requests. On the other hand, EAT attempts to accommodate traffic services over the active network elements, which may congest some specific links and nodes. Consequently, adopting the K-SP algorithm makes it more likely to find paths with enough bandwidth to accommodate the service needs than using the EAR algorithm. The behavior of both approaches has a significant impact on the resulting BBR. As shown, K-SP achieves a lower BBR (enhanced service provisioning) when compared to the EAR algorithm.

As mentioned above, the EAR algorithm focuses on reducing the overall network energy consumption. Thereby, it is possible that for the received service requests the EAR algorithm may choose paths which traverse more devices and links than the K-SP algorithm. The selected EAR path devices and links tend to be already active (i.e., powered up) which do reduce the overall network energy. Conversely, the K-SP does not consider whether a device or link is already being used by existing services since its purpose is to reduce the overall resource consumption. Consequently, the EAR algorithm does reduce the energy consumption at the expenses of accommodating the services through larger number of hops, i.e., occupying more link bandwidth. This creates the well-known trade-off between resource utilization and energy-efficiency objectives. Observe that the enhancement of the K-SP in the BBR concerning the EAR strategy remains practically constant for all the HTs.

In Figure 61.b, for the same HT values, it is shown the average network energy consumed by both EAR and K-SP algorithms. As already discussed, EAR does reduce the network energy compared to K-SP solution. This performance difference is more relevant at lower HT values. The reason for this is that resources are less occupied; thus, it is more likely that active network elements (devices and links) can be re-used rather than powering up new ones. However, as HT grows, resources become more occupied, and fulfilling the service needs is more complex. Then, more devices and links need to be powered up to deploy the network services, increasing the overall network energy consumption. In other words, we observe that the improvement attained by the EAR concerning K-SP on the network energy consumption tends to disappear as HT grows.

Finally, note that the average throughput (in Gb/s) is practically the same for both EAR and K-SP strategies. Macroscopically, both solutions tend to transport the same amount of data traffic (in Gb/s) since the BBR difference is not very high. To conclude, for the considered HT values and evaluation scenario, EAR algorithm allows reducing the overall network energy consumption achieving similar aggregated throughput when compared to traditional K-SP approach. Only the BBR accomplished by EAR is slightly worsened to the one attained by the K-SP algorithm.

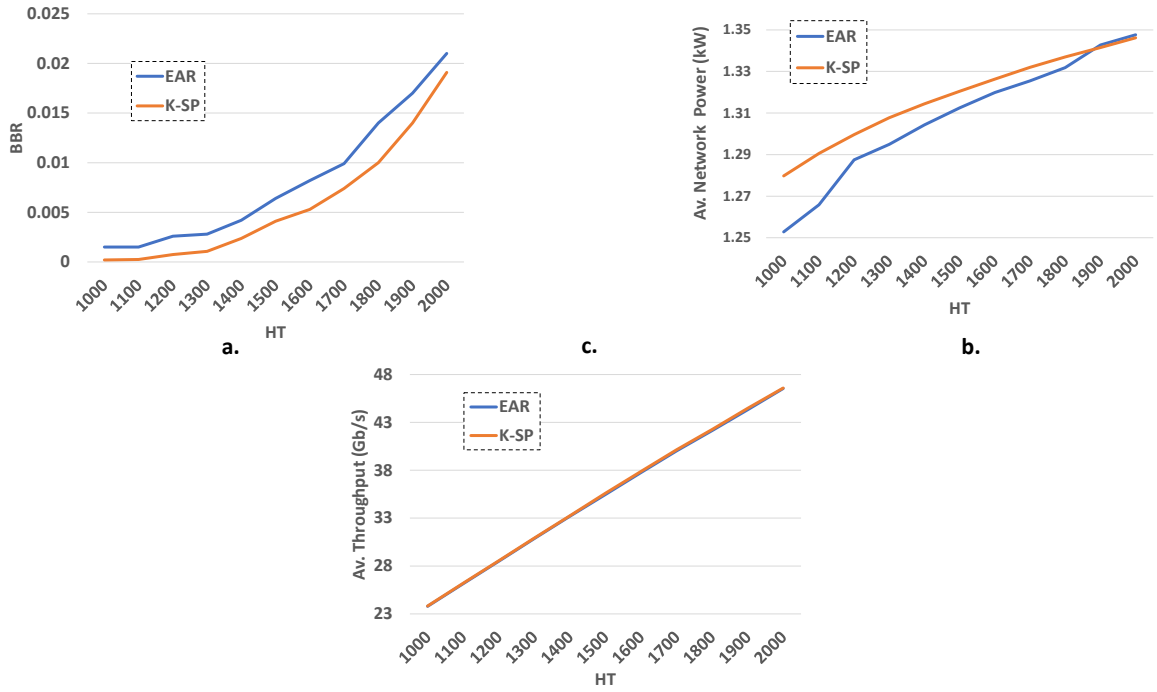


Figure 61: EAR and K-SP performance evaluation: a) BBR; b) av. Consumed Network Energy (in kW); c) av. throughput (in Gb/s)

Name	Value	Comment
Energy	< 30%	In a dynamic traffic environment where connection services arrive and departure, adopting routing algorithms to achieve reducing energy consumption depends notably on diverse factors: 1. the transport network (e.g., nodal degree, physical connectivity, link characteristics, etc.) 2. the service requirements in terms of bandwidth, bandwidth + latency, etc. 3. the network energy model (i.e., device and/or port energy consumption). For a very specific scenario, we have evaluated in the TeraFlow SDN controller Path Computation, the EAR algorithm to contribute to reducing the overall end-to-end energy consumption without notably degrading other performance metrics such as the average BBR or throughput. The devised EAR relies on a heuristic which favours the routing through active network elements rather than powered up devices and/or ports as much as possible. The conducted study paves the way to continuing working in the TeraFlow SDN controller to become a controller which adopts energy-efficiency objectives. In this regard, next steps are envisaged to tackle more advanced algorithms such as re-allocating the established services to enforce powering-down network elements and/or exploiting the benefits of suing machine learning trained models to attain better trade-offs between service provisioning and energy reduction.

4.4. Scenario conclusions

As a conclusion, we provide a summary of measured KPIs in Table 9, we analyze them, and provide further steps.

Table 9 Scenario 1 summary of measured KPI

KPI	Target	Validation results
Device on-boarding time	< 50ms	100-400 ms. The target was too optimistic, but it does not have a real impact since on-boarding is performed only during the initialization phase. The current on-boarding procedure implies multiple interactions with the underlying database. In the next releases, we plan to optimize those interactions further to reduce the on-boarding time.
Service setup delay	< 50ms	The measured overhead using emulated devices is 100ms. This deviation is detailed at the end of this section.

Service teardown delay	< 50ms	The measured overhead using emulated devices is 90ms. Similarly, as with service setup delay, in future releases, we plan to review the overall Service teardown workflow to improve the internal finite state machine, identify unneeded database interactions, and add support for parallel device deconfiguration.
Data rate	100G	Data rate is able to be shown in Grafana. Available data rates are dependent on the topology and network equipment, so we are limited to the transponders available in whiteboxes (e.g., 100G).
End-to-end service latency	5ms (indicative)	The actual value depends on the topology setup. For example, hardware switches are faster than software switches, while software switches perform better on high-end Commercial off-the-Shelf (COTS) hardware. Therefore, this value may vary. 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.
Reaction time to ensure SLA	~4s	The reaction time can be further analyzed on RPC calls (~1s) and path recalculation time (~3s). It should also be noted that the path recalculation time depends on the topology. Larger topologies may require more time than smaller topologies to find new paths.
Resource efficiency reduction factor	2	L2VPN is of 2,32 at offered load of 5k Erlang (peak resource efficiency). L3VPN is of 6,4 at offered load of 7k Erlang (peak resource efficiency).
Energy	< 30%	The devised EAR relies on a heuristic favouring the routing through active network elements rather than powered-up devices and/or ports as much as possible. The conducted study paves the way to continuing working in the TeraFlow SDN controller to become a controller which adopts energy-efficiency objectives. In this regard, next steps are envisaged to tackle more advanced algorithms such as re-allocating the established services to enforce powering-down network elements and/or exploiting the benefits of suing machine learning trained models to attain better trade-offs between service provisioning and energy reduction.

The deviation in Service setup delay from the target comes from multiple sources, notably, the path computation time and the overall service setup workflow.

Path Computation component:

The Path Computation backend was contributed “as it was” to the TeraFlowSDN controller. No effort was planned for (re)designing a Path Computation component, so we had to take the easy step of implementing a front-end component connecting the TeraFlowSDN components with the backend. As a result, the front-end needs to convert the requests received from the TeraFlowSDN components into valid requests for the backend, and vice-versa with the replies. Besides, the backend implemented a completely custom path computation engine.

In future releases, we plan to implement the following improvements related to the Path Computation component:

- Combine the front-end and back-end modules of the Path Computation component to reduce network latencies and remove the current data conversion needs.
- Migrate the current path computation engine to a performant and deeply tested Graph library (e.g., Boost Graph Library) to accelerate the computations and improve its extensibility.

Service component:

The Service component implemented a generic workflow for setting up connectivity services and enabled specialized behaviors per service type through Service Handlers. The current scheme requires several interactions with the underlying databases and sequential device configurations.

In future releases, we plan to implement the following improvements related to the Service component:

- Review the overall Service setup workflow to identify unnecessary database interactions and improve the internal finite state machine used to manage the connectivity services.
- Implement support for a new parallel device configuration engine.

5. Scenario 2: Inter-domain

This section introduces the second scenario explored within the TeraFlow project. The scenario addresses the inter-domain deployment of transport network slices. This scenario is operator-led, emphasising the advancement of transport networks through the collaborative orchestration of multiple domains. Key aspects considered in this scenario include scalability and traceability, aiming to ensure efficient and manageable operations across diverse domains.

First, we introduce the scenario. Second, we present its alignment with TeraFlowSDN architecture. Third, we present the performance evaluation. Finally, we provide a summary of scenario conclusions and future steps.

5.1. Scenario Introduction

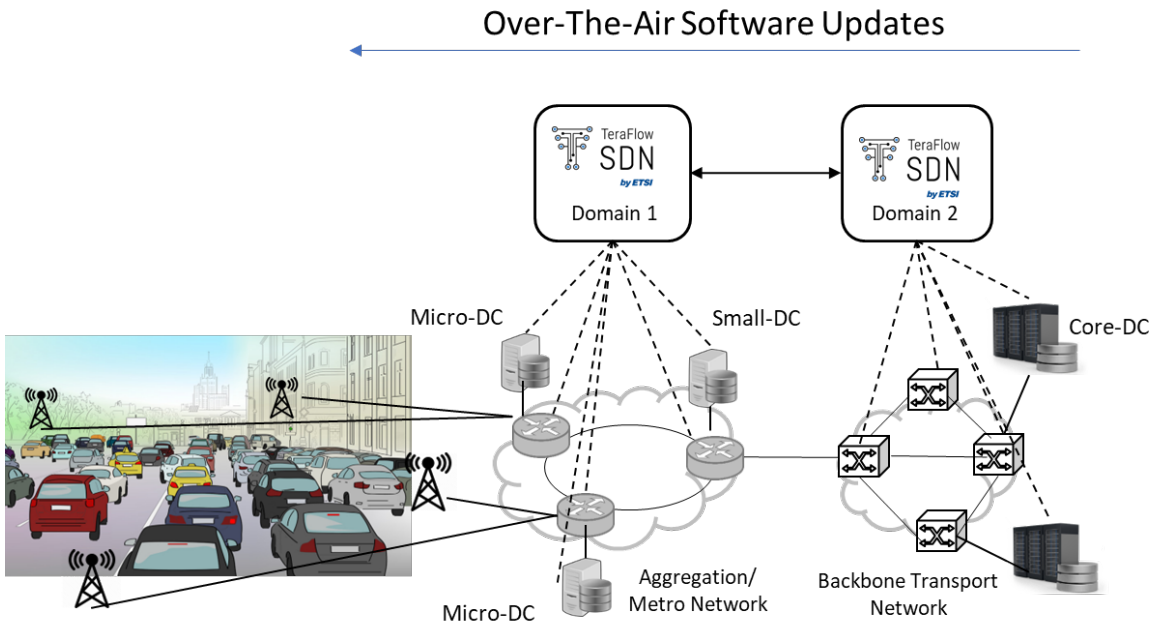


Figure 62 Scenario 2: Inter-domain

Several challenges must be addressed when deploying Cooperative, Connected, and Automated Mobility (CCAM) services over a distributed edge and cloud infrastructure. These challenges involve unified management of computing, storage, and networking resources, multi-domain networking, and inter-domain slicing between network operators while maintaining data confidentiality. The TeraFlowSDN Controller plays a crucial role in overcoming these challenges.

First, achieving unified resource management requires the TeraFlowSDN Controller, in collaboration with an NFV orchestrator like OSM, to deploy integrated services. This involves provisioning cloud and edge computing resources and establishing connectivity between them. Simultaneously, the optimization of cloud and network resources at the packet and optical layer, is carried out.

Second, the issue of multi-domain networking arises, where resources in each domain must be allocated and combined to create an end-to-end service. To address this, the TeraFlowSDN Controller deploys multiple per-domain slice instances and orchestrates their composition to form complete transport network slices spanning multiple domains.

Finally, when different domains belong to separate network operators, mechanisms for inter-domain slicing while maintaining the privacy of internal network details become essential. The TeraFlowSDN controller incorporates a Distributed Ledger Technology (DLT) component based on blockchain technologies. This ensures that data exchanged between per-domain TeraFlowSDN instances can be kept confidential, if required, while enabling inter-operator collaboration.

An illustrative representation of the inter-domain scenario is depicted in Figure 62. The scenario encompasses diverse packet and optical transport networks catering to metropolitan and core segments. These networks facilitate the connectivity among distributed cloud and edge computing infrastructures. In the context of the CCAM services, deployment options are available at various locations, including edge nodes (such as cell sites, street cabinets, and lampposts) where micro-DCs can be established. Small-DCs in central offices also offer low to moderate computational capacity and low response times. At the same time, core-DCs within the core network provide high-computational capacity and moderate response times.

To ensure effective management and control, the transport and cloud infrastructures are partitioned into distinct domains, each governed by an instance of the TeraFlowSDN Controller. While addressing scenarios involving uplink-heavy and latency-sensitive requirements, emphasis is placed on Over-the-Air (OTA) software updates. These updates involve the wireless transmission of software enhancements from car companies to vehicles. Given the dynamic nature of these updates, an inter-domain scenario is introduced to enable the provisioning of moving connectivity services based on the positioning of network elements. The interaction between the Transport Network Slice and its endpoints with neighboring access and service edge SDN control domains becomes a focal point for testing, experimentation, and exploration within this inter-domain setting.

5.2. Alignment with TeraFlowSDN architecture

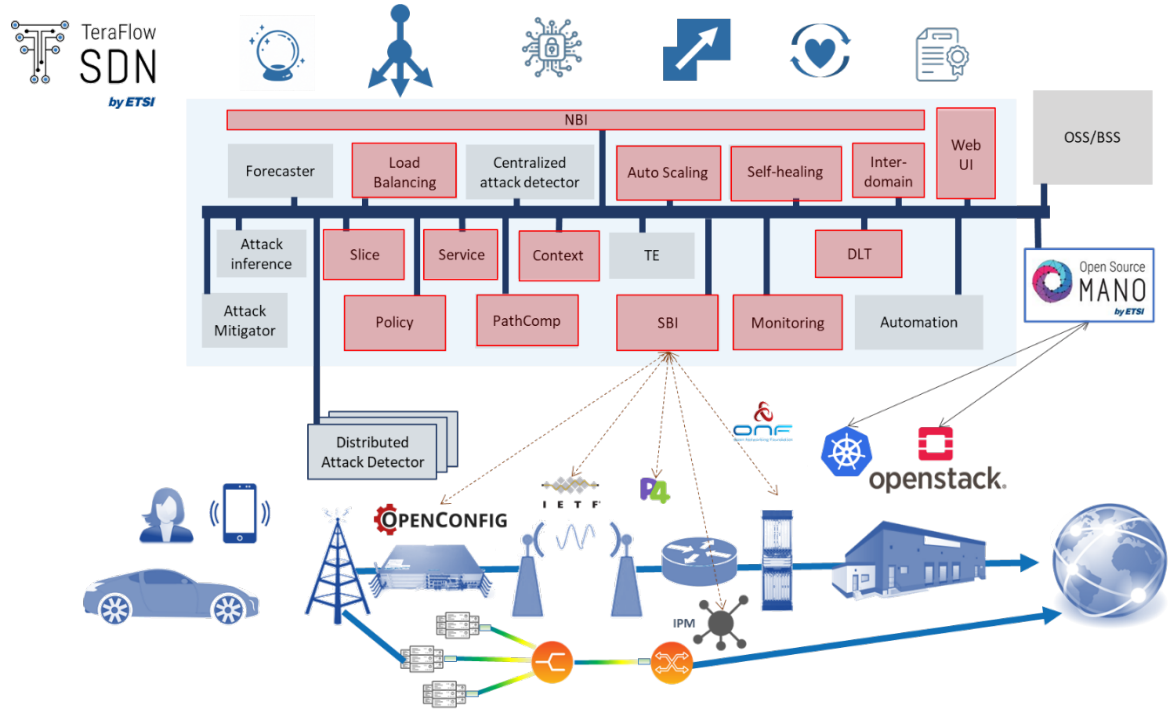


Figure 63. Scenario 2 TeraFlow instantiation in a single domain

Figure 63 shows the single domain instantiation (configuration and TeraFlowSDN templates) of the TeraFlowSDN controller. Interdomain connectivity will be provided either with DLT or the inter-domain components between multiple instances of the TeraFlowSDN controller.

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 Annex I 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

5.3. Performance Evaluation

This section presents the performance evaluation performed for the different workflows composing scenario 2. The workflows have been introduced and detailed in D5.2, Section 6.5. In this deliverable, we focus on the performance evaluation of the workflows. First, we introduce the testbed setup used by the partners to evaluate the performance of the workflows. Then, we present the performance evaluation of each workflow.

5.3.1. Testbed Setup

The setting 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.

We will use the TAPI-enabled OLS controller and the underlying optical transport network infrastructure to validate this scenario. Moreover, CTTC has two whiteboxes cell-site gateways (CSGW)

[EDG22] with IP Infusion OcNOS available and controlled using TeraFlowSDN (Figure 64 Interconnected CSWGs at CTTC Testbed).

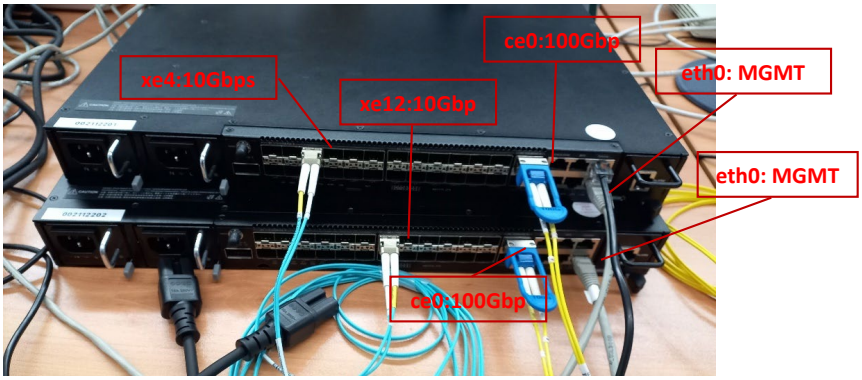


Figure 64 Interconnected CSWGs at CTTC Testbed

- **NEC** contributes to the blockchain infrastructure providing the means to interconnect different instances of the TeraFlowSDN for the different domains. More details are provided in D4.2 Section 4.1.
- **Telenor** Telenor's testbed includes 1x HPE Proliant DL360 Gen10 server and 2x Edge-Core CSR320 (AS7316-26XB) whitebox switches, which are interconnected by the FS S5860-20SQ switch through 10G links, as shown in Figure 65. Telenor's testbed. The server features 2x Intel(R) Xeon(R) Gold 6238R CPUs with 256 GB of RAM. The server runs Ubuntu 20.04.6 LTS Server OS and MicroK8s v1.24.13. The TeraFlowSDN controller runs on top of MicroK8s. The two Edge-Core whiteboxes are directly connected back-to-back through a 40G link.

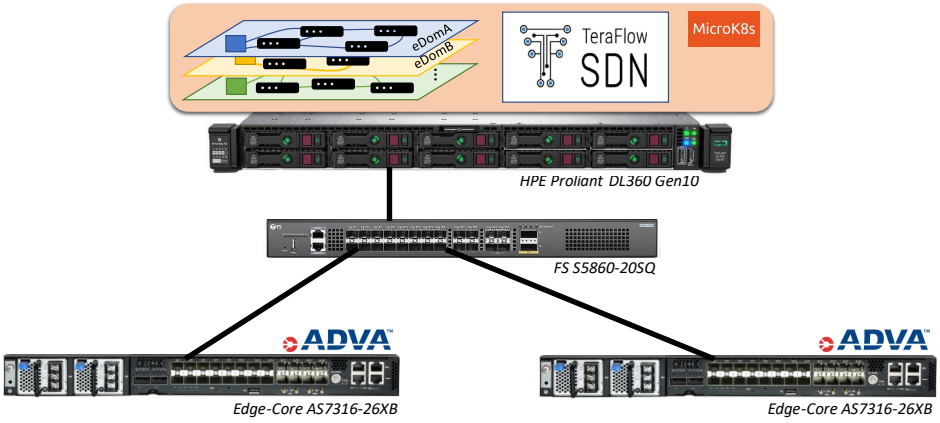


Figure 65. Telenor's testbed

- **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.

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

This section details the provisioning of inter-domain transport network slices with SLA. To accurately measure the time it takes to set up the inter-domain transport network slice using TeraFlowSDN, we used emulated devices to exclude the time taken to configure network equipment. This approach ensures precise accounting of TeraFlowSDN's contribution.

The experiment is based on the components and workflows reported in [OECC22] that have been upgraded to TeraFlowSDN Release 2.1 and improved with inter-domain SLA propagation between domains. The overall experiment has been carried out on the distributed testbed interconnecting CTTC and Telenor Norway premises.

We configured a TeraFlowSDN instance as domain D1 (*tfs-dom1*) on the CTTC side, representing a telecom infrastructure managed by one network operator. We then configured a second TeraFlowSDN instance as domain D2 (*tfs-dom2*) on Telenor side in charge of managing a transport network.

Figure 66 and Figure 67 depict, respectively, the topologies of domains D1 and D2. On one side, the telecom infrastructure in domain D1 is composed of 2 DCs (DC1 and DC2), and a Core network composed of 5 emulated packet routers that are managed locally by the network operator in charge of domain D1. On the other hand, domain D2 comprises 3 emulated packet routers forming a transport network managed by a remote network operator. Note that border endpoints in Domain D2 are in routers R2 and R3 that are connected, respectively, to R4 in D1 and DC2. DC1 is directly connected to the Core network through router R4; however, to reach the core network, DC2 needed to connect through domain D2.

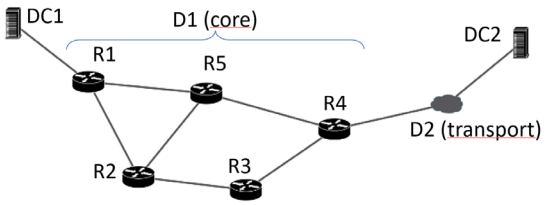


Figure 66. Domain 1 – Network Topology

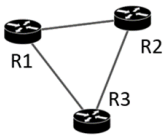


Figure 67. Domain 2 – Network Topology

The descriptor used to request the inter-domain DC-to-DC transport network slice is listed in Table 10. It provides the details for creating a transport network slice between DC1 and DC2, constraining the end-to-end latency and the desired capacity, and defining the Maximum Transfer Unit (MTU) and VLAN tags requested by the customer. Note that no slice type is specified; the TeraFlowSDN controller infers that a L2VPN needs to be created, given no IP addresses are specified in the request.

Table 10. Inter-domain slice descriptor

```
{"slices": [{
   "slice_id": {
      "context_id": {"context_uuid": {"uuid": "admin"}},
      "slice_uuid": {"uuid": "idc-l2-slice"}
   },
   "slice_endpoint_ids": [
      {"device_id": {"device_uuid": {"uuid": "DC1"}}, "endpoint_uuid": {"uuid": "int"}},
      {"device_id": {"device_uuid": {"uuid": "DC2"}}, "endpoint_uuid": {"uuid": "int"}}
   ],
   "slice_constraints": [
      {"sla_capacity": {"capacity_gbps": 10.0}},
      {"sla_latency": {"e2e_latency_ms": 15.2}}
   ],
   "slice_config": {"config_rules": [
      {"action": 1, "custom": {
         "resource_key": "/settings", "resource_value": {"mtu": 1512, "vlan_id": 300}
```

```
        }}
    ]}
}]}
```

We triggered the inter-domain slice setup by submitting the inter-domain slice descriptor through the WebUI of *tfs-dom1* controller. Figure 68 illustrates the 3 slices (1 main slice + 2 sub-slices) created in *tfs-dom1*. The WebUI forwards the request to the Slice component of *tfs-dom1* controller. The Slice component computes a simple shortest path and realizes it needs to traverse a remote network domain. Thus an inter-domain slice needs to be created. As a result, the Slice component delegates to the Inter-domain component the creation of the slice. The inter-domain component first creates an end-to-end inter-domain slice (slice *idc-l2-slice* in Figure 68). It performs an end-to-end path computation to identify the sequence of network devices in domain D1 and the remote domain that needs to be traversed. The details of slice *idc-l2-slice* are reported in Figure 69.

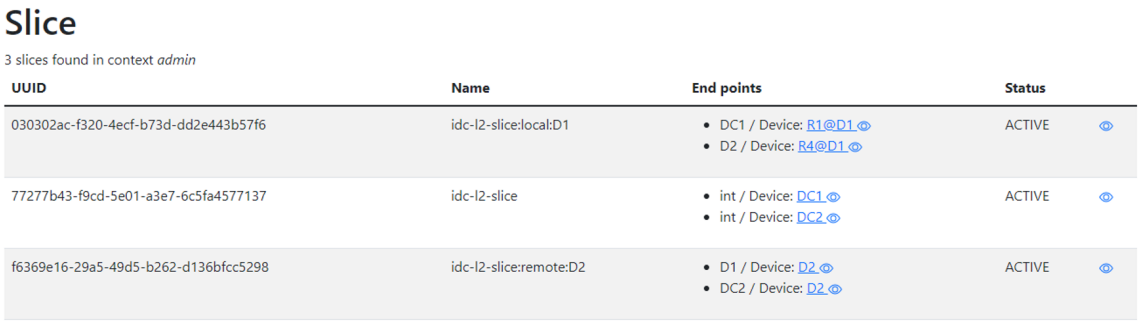


Figure 68. Slices in tfs-dom1

For the local network devices, it creates a local slice for domain D1 (*idc-l2-slice:local:D1* in Figure 68). The local slice inherits the SLA constraints and the configuration rules defined in the end-to-end slice. The endpoints for the local slice are inferred from the border endpoints in domain D1 that result from the end-to-end path computation. The setup of this local slice is then delegated to the Slice component that triggers the creation of the appropriate connectivity services and the configuration of the underlying local network devices using the appropriate and available control protocols.

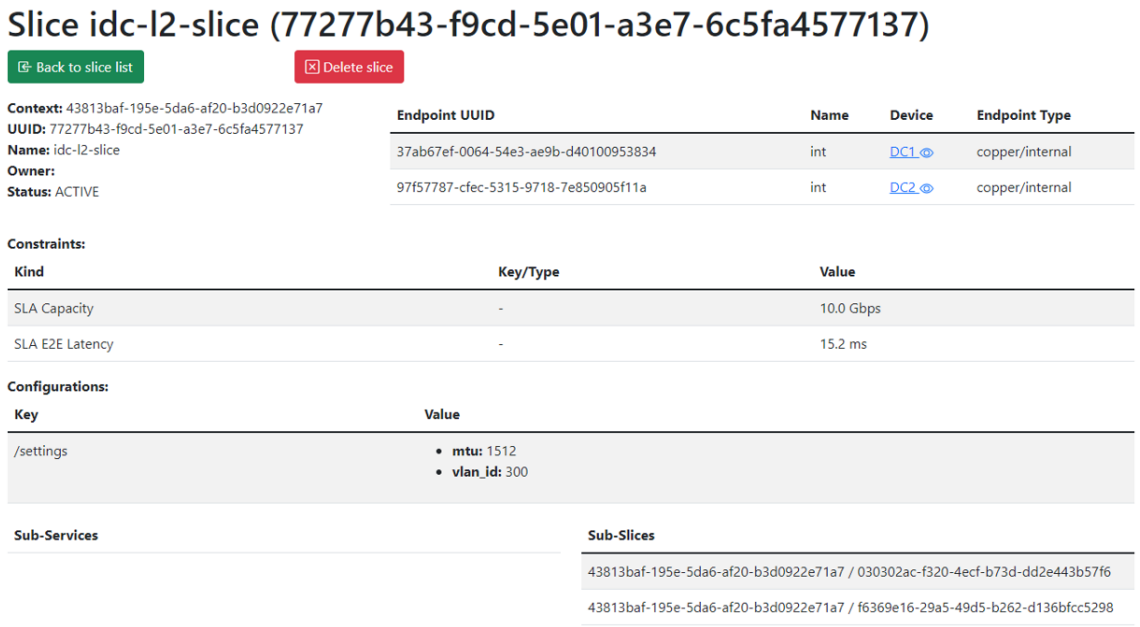


Figure 69. Detail of Inter-domain slice created in tfs-dom1

A screenshot of local slice *idc-l2-slice:local:D1* is not provided because it is almost equal to *idc-l2-slice*. The only difference is that the local slice has no sub-slices, but it has a sub-service. The sub-service created in *tfs-dom1* is illustrated in Figure 70, detailing the path through the network equipment in D1.

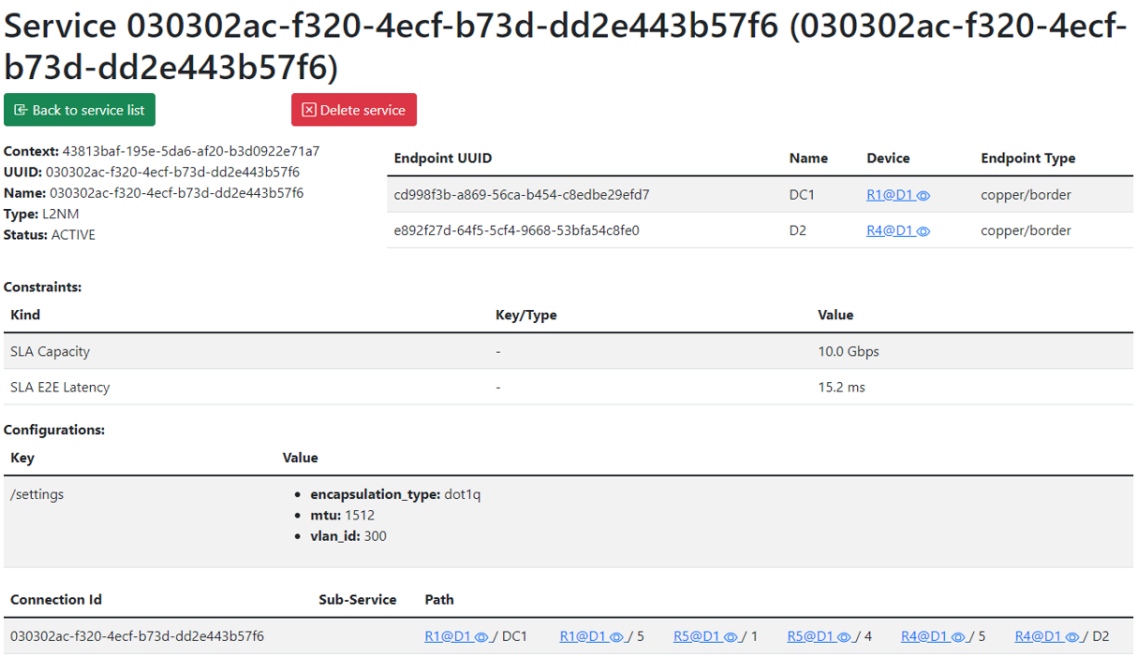


Figure 70. Detail of the service created in tfs-dom1

For each remote network domain to be traversed, the Inter-domain component creates a remote slice (*idc-l2-slice:remote:D2* in Figure 68). Again, the remote slices inherit the SLA constraints and the configuration rules defined in the end-to-end slice. Besides, the endpoints for each remote slice are inferred from the border endpoints in the corresponding domain that result from the end-to-end path computation. Inter-domain component in *tfs-dom1* interacts with the Inter-domain components in the remote TeraFlowSDN controllers (e.g., only *tfs-dom2* in this experiment) and delegates to them the creation of the remote slice(s).

When the Inter-domain component in *tfs-dom2* receives the request, it identifies it is a local slice requested by a remote domain, and delegates to the Slice component the creation of the slice (*idc-l2-slice:remote:D2* in Figure 71, details in Figure 72), the appropriate connectivity services, and the configuration of the underlying network devices using the appropriate and available control protocols.

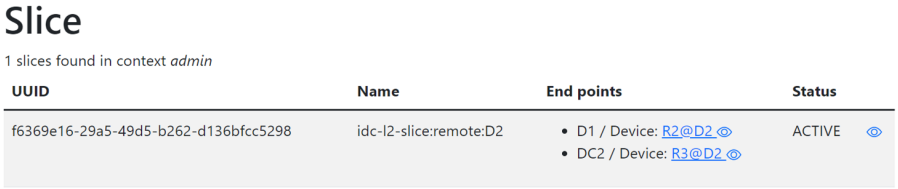


Figure 71. Slices in tfs-dom2

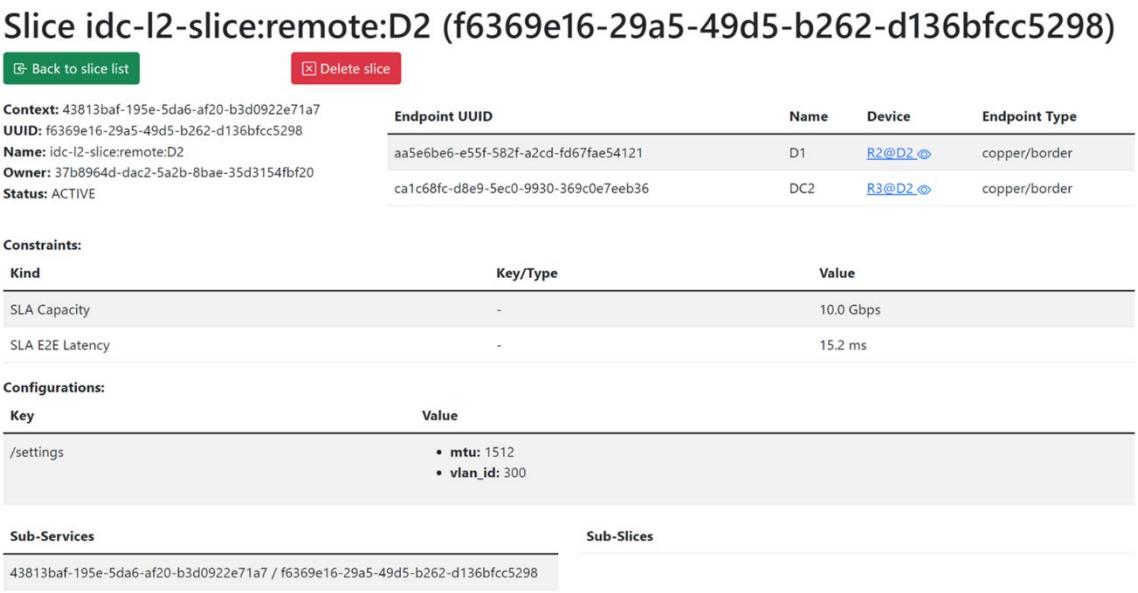


Figure 72. Detail of the slice created in tfs-dom2

Note that the Service component needs to perform a path computation local to domain D2 given that the *tfs-dom1* controller only knows the border endpoints of domain D2, but not the internal details on the topology of domain D2. The resulting sub-service created in *tfs-dom2* and the selected path through D2 are illustrated in Figure 73.

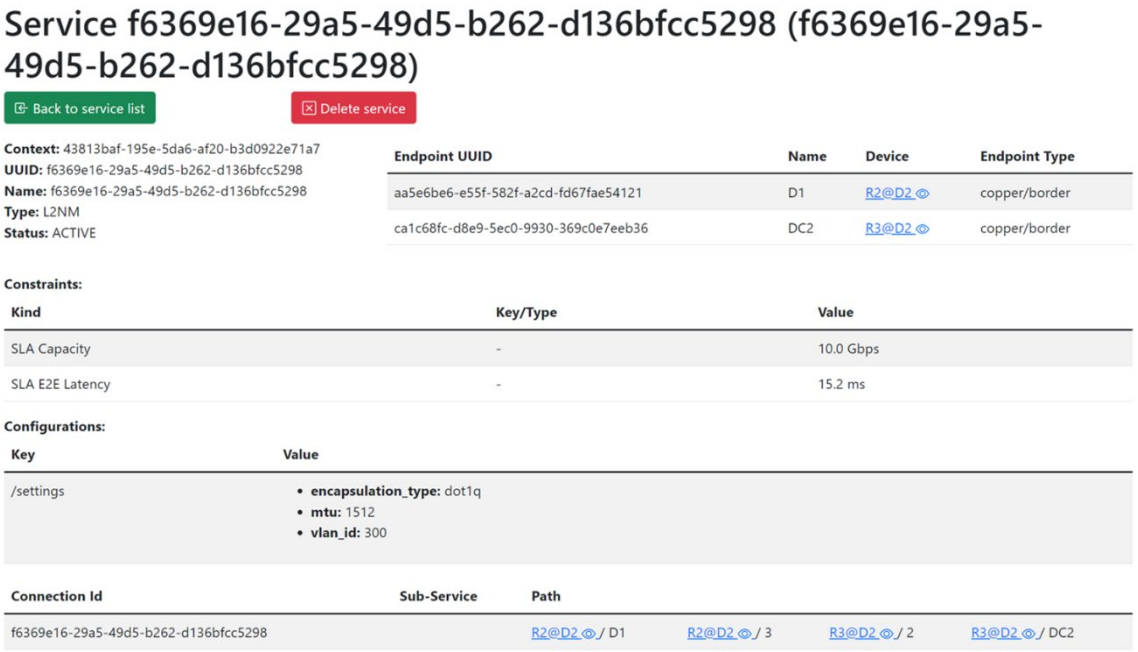


Figure 73. Detail of the service created in tfs-dom2

When the Inter-domain component in *tfs-dom2* completes the creation of the slice, it replies with the status of the slice to the Inter-domain component in *tfs-dom1*, (i.e., status=*ACTIVE*). The Inter-domain component in *tfs-dom1* assigns the created local and remote slices as sub-slices of the main end-to-end inter-domain slice requested by the user, as depicted in the bottom part of Figure 69. After all the local and remote sub-slices are created and activated, the Inter-domain component activates the main end-to-end inter-domain slice. It reports the result to the requesting user through the WebUI, or the requesting OSS/BSS through the NBI component.

In this experiment, we accounted for the setup time of an inter-domain slice in our distributed testbed. To be fair, we accounted the Round Trip Time (RTT) through the VPN between *tfs-dom1* at the CTTC premises (Spain) and *tfs-dom2* at the Telenor premises (Norway). The results are detailed in Table 11.

Table 11. CTTC-TFS to TNOR-TFS RTT through the VPN connection

Min	Avg	Max	StDev
63.20 ms	63.49 ms	63.90 ms	0.18 ms

To complement the experimental validation, in Figure 74 we provide the Wireshark capture detailing the *tfs-dom1* inter-domain – to – *tfs-dom2* inter-domain component interactions and the execution times (in seconds) for the operations. Note that 3 operations are illustrated: *i*) initial mutual authentication phase between domains (done during the onboarding of the network topologies in each TFS controller), *ii*) the Lookup Slice request used to check if the slice already exists in the remote domain, and *iii*) the slice creation in the remote domain.

Time	Source	Destination	Length	Info
REF	tfs-dom1	tfs-dom2	165	HEADERS[3]: POST /interdomain.InterdomainService/Authenticate, DATA[3] (GRPC)
0.067	tfs-dom2	tfs-dom1	87	DATA[3] (GRPC) (PROTOBUF) context.AuthenticationResult
REF	tfs-dom1	tfs-dom2	441	HEADERS[5]: POST /interdomain.InterdomainService/LookUpSlice, DATA[5] (GRPC)
0.098	tfs-dom2	tfs-dom1	74	DATA[5] (GRPC)
REF	tfs-dom1	tfs-dom2	451	HEADERS[7]: POST /interdomain.InterdomainService/CreateSliceAndAddToCatalog,
1.420	tfs-dom2	tfs-dom1	756	DATA[7] (GRPC) (PROTOBUF) context.Slice

Figure 74. Wireshark Capture inter-doman slice

This Wireshark capture includes the execution time per operation perceived in the *tfs-dom1* controller. It is worth noting that these execution times include the VPN RTT as reported in Table 11, thus, in Table 12 we provide the values including and extracting the VPN RTT contribution. For this calculation, we assumed an average RTT of 63.5 ms, given the standard deviation measured is very small (180 us).

As might be expected, the mutual authentication is almost immediate as it only requires validating credentials on the *tfs-dom2* controller. The slice lookup to validate existence of the slice requires a simple consultation in the Context database, thus taking around 35 ms at the *tfs-dom2* controller. Finally, the slice creation consumes a few seconds given that the following steps must be taken: path needs computation (including consultation of network topology in the Context database), the creation of the Slice and the associated Service, and the configuration of the emulated network devices.

Table 12. Inter-domain slice provisioning - Execution times with and without accounting VPN round-trip-time

Operation	Execution Time	
	Accounting VPN RTT	TFS contribution (wo/VPN RTT)
Mutual authentication	67 ms	3.5 ms
Check if slice exists in remote domain	98 ms	34.5 ms
Create slice in remote domain	1.42 sec	1.36 sec

Classical procedures for creating inter-domain slices require the two network operators to contact each other, agree on the parameters, configure and test the slices manually, and confirm the slice activation. This entire procedure can easily consume many hours or even days. In contrast, this solution enables the creation of an inter-domain slice in roughly 2 seconds, thus entailing a significant reduction in OPEX for the network operators.

Name	Value	Comment
Service setup delay	1.36 sec	Without counting VPN RTT between TFS instances.
OPEX Reduction (time)	1800x	Assuming extremely optimistic classical approach of 1 hour for setting up the inter-domain slice in both domains. Reduction factor = 3600 sec / 2 sec.

5.3.3. Distributed Ledger Technologies

This section has been organized into three sub-sections. The first one addresses the assessment of the trust and privacy of the data stored in the Blockchain. The second sub-section addresses the performance of creating, retrieving, updating, and deleting records on the HyperLedger Fabric BlockChain through the TeraFlowSDN DLT Gateway. The third sub-section compares the inter-domain slice provisioning time using DLT against the case without DLT.

5.3.3.1. DLT Trust and Privacy

In order to maximize privacy and ensure proper access control, all DLT component communications are encrypted using TLS and authenticated through certificates and gRPC. This includes both the intra-DLT component communication between the nodes as well as communication from/to the DLT clients.

To improve privacy, we further investigated Hyperledger Fabric's Idemix, an anonymous credential system for authentication. Idemix allows users to authenticate in a privacy-preserving manner, making the verifier only know that the user possesses a valid certificate. Subsequent authentications are unlinkable.

Name	Value	Comment
Trust/privacy	100% secured connection	All connection related to the DLT component are secured and authenticated.
DLT transaction delay	10s	Average latency is between 2.2 and 3.3 seconds, Figure 75.

5.3.3.2. DLT Gateway and Blockchain Performance

To assess the performance of the DLT Gateway and the Blockchain network regarding latency, we prepared a small performance assessment tool leveraging the DLT Gateway interface used by the DLT Connector component to interact with the HyperLedger Fabric Blockchain. This ad-hoc tool uses randomly generated entities (devices, links, services, and slices) to create, retrieve, update, and delete operations over them in the Blockchain. Note that, for the sake of validating the performance, the data model structure and the data size of records is relevant.

The performance assessment starts by initializing random entities to have data records available on the Blockchain. Otherwise, operations such as get, update, or delete are ignored due to the lack of records to manipulate. The tool keeps track of the entity data records in a local cache. Whenever an entity is requested and/or modified locally, the corresponding operation against the Blockchain is executed, and its response time is monitored.

Each modification in the Blockchain records triggers the distribution of a modification event to the peers connected to the Blockchain, providing the type of record modified, the unique identifier of the record, and the operation carried out, among others. The assessment tool subscribes to these events, it correlates each action performed on the Blockchain with the collected events. It computes the delay between the instant of time when the modification request was made, and the record modification event was received. This shows the synchronization time between peers connected to the Blockchain.

We configured the tool to initialize 20 devices, 20 links, 20 services, and 20 slices. Then, the tool sequentially executes 1000 operations, uniformly distributed between creation, retrieval, update, and delete, over randomly chosen devices, links, services, and slices. For each operation and record type, the size of the target entity in bytes, the number of endpoints, the number of constraints (for services and slices), the number of configuration rules (for devices, services, and slices), the number of sub-services (for slices), and the number of sub-slices (for slices) is recorded. Besides, it is also recorded the execution time of the store/retrieve operations in the blockchain, as well as the delay between a change being made in a record and a remote peer receiving the corresponding asynchronous notification event.

Given that all the data records are treated the same way in the Blockchain, no differentiation is made between record types. Instead, we report results based on the operation performed on the records and the actual size of the record payload in bytes. To produce different record sizes, we randomly filled in the device, service, and slice records with randomly generated numbers of endpoints, configuration rules, and constraints. For the links, TeraFlowSDN only supports unidirectional point-to-point links, so the size is constant (2 endpoints).

Figure 75 illustrates, for each type of operation, the execution time as a function of the record size. As expected, the Get operation takes significantly less time, given there is no need to reach a consensus between the Blockchain nodes, and any node can inspect its replica of the overall data and return it. The execution time increases linearly with the record size but not significantly. The Create and Update operations take almost the same amount of time. This is because creating or updating a record, in the backend, means just checking if the record exists or not, which has a negligible cost. The delete operation takes less time than the Create/Update operations. The reason is that deleting does not require reaching a consensus between the peers to allocate a new record; instead, it only requires notifying all the other peers that an existing record needs to be marked as removed.

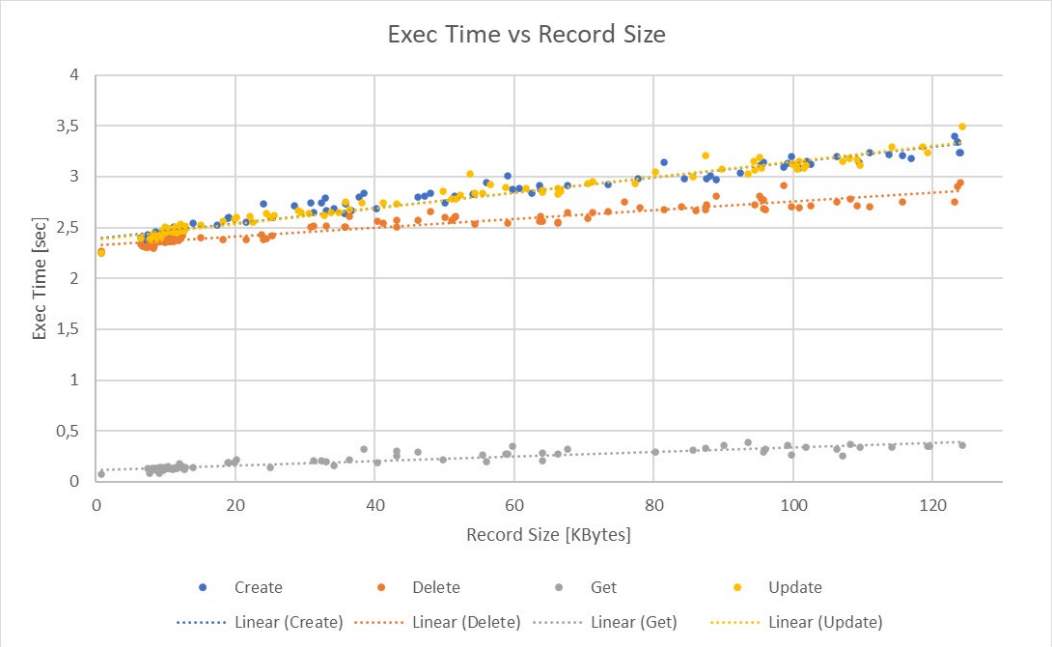


Figure 75. DLT Execution Time vs Record Size

Figure 76, depicts the delay between the instant when a create, update, or delete operation is triggered at one TeraFlowSDN instance, and a remote instance receives the modification event. Get events are not metered since no notification is circulated when a record is retrieved. Note that, for the sake of having realistic measurements, the performance assessment tool contains both the DLT gateway client and the collection of the events. Otherwise, deviations in machine clocks could result in erroneous results. Also, we measured the Round-Trip Time between the machine running the performance tool (at CTTC premises) and the one running the HyperLedger Fabric Blockchain (at NEC premises). The RTT measurement is reported in Table 13.

Table 13. DLT Event Delay - Setup Round Trip Time

Min [ms]	**Avg [ms]**	**Max [ms]**
58.109	62.829	67.093

Also, in this case, the delays for the create and update operations overlap and are slightly higher than the ones relative to the delete operation. The delay increases with the record size. This is because the event is distributed when the record is modified, and we observed in execution times in Figure 75 that they increase proportionally with the record size.

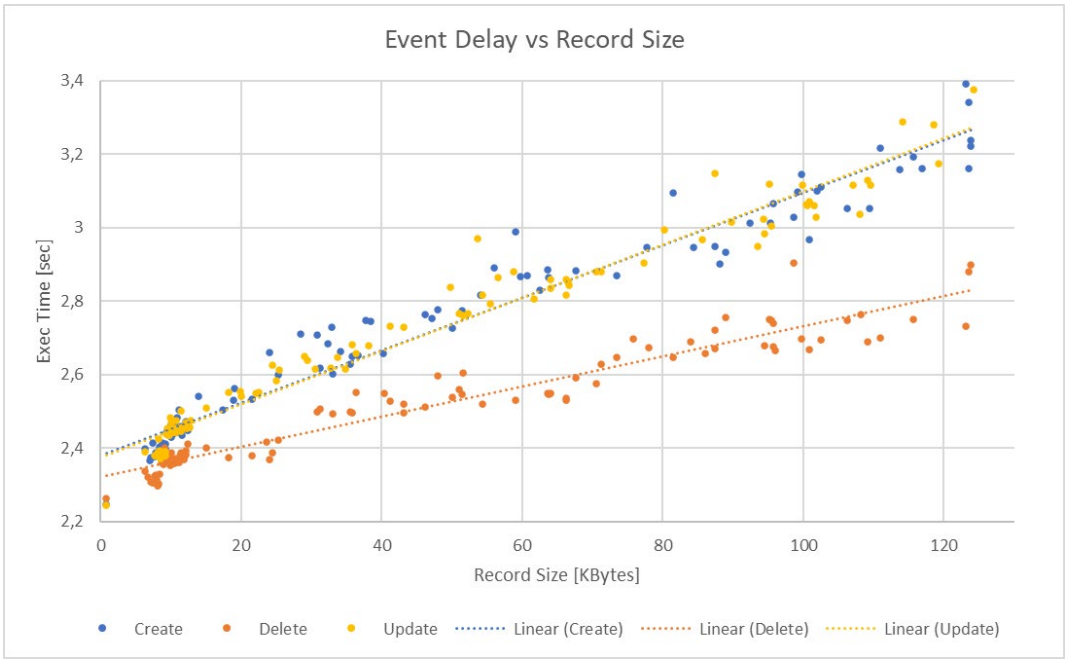


Figure 76. DLT Event Reception Delay vs Record Size

We can conclude that using the DLT for inter-domain operations, even when it provides enhanced trust and privacy, implies significant extra delays due to the internal consensus algorithms supporting the Blockchain operation. Thus, the information stored in the Blockchain needs to be as limited as possible to minimize the impact on the performance, both in execution times and in event distribution delays.

5.3.3.3. Inter-domain Provisioning through DLT

This workflow has been demonstrated in [NFV22] and validated and evaluated in D5.2 Section 6.

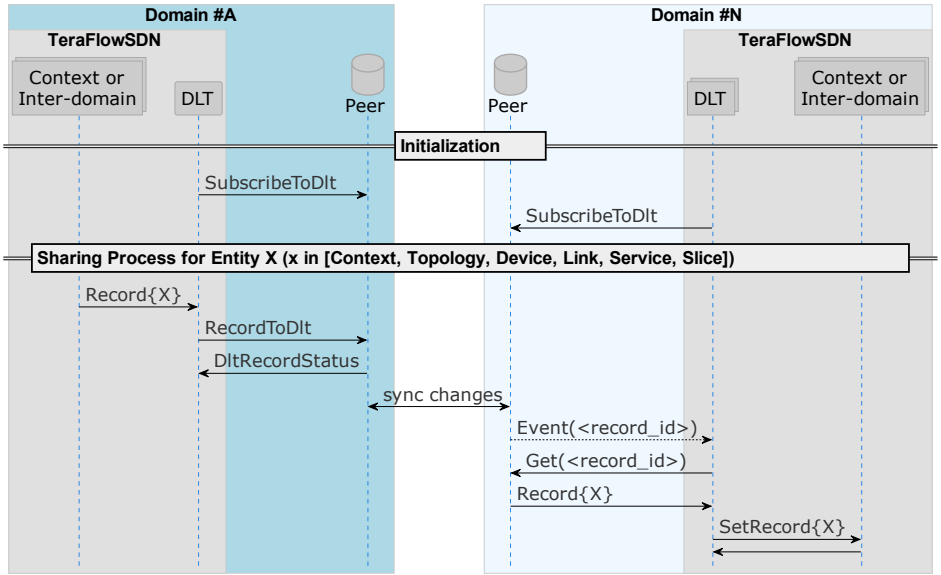


Figure 77 Scenario 2 workflow: Sequence diagram for DLT use

Figure 77 details the workflow for establishing an inter-domain transport network slice. Figure 78 details a Wireshark capture with the externally-visible messages involved in the validation experiment 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 Figure 78, an arbitrary TeraFlowSDN component requests 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). It forwards the request to the DLT Gateway that triggers the upload into the Blockchain hosted by NEC in Germany (messages 2056-2885). Once done, the DLT Gateway replies to the DLT Connector, which, in turn, 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] (GR
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 78. Transport Network topology for DLT evaluation

Figure 79 shows the Cumulative Distribution Function (CDF) of the DLT latency for the 100 generated requests (curves are drawn for Device, Link, Service and Slice). We observe that the delay takes around 10 seconds for the different curves. The main contribution of this delay is the cost of uploading the record into the blockchain due to the consensus and ordering constraints that need to be fulfilled.

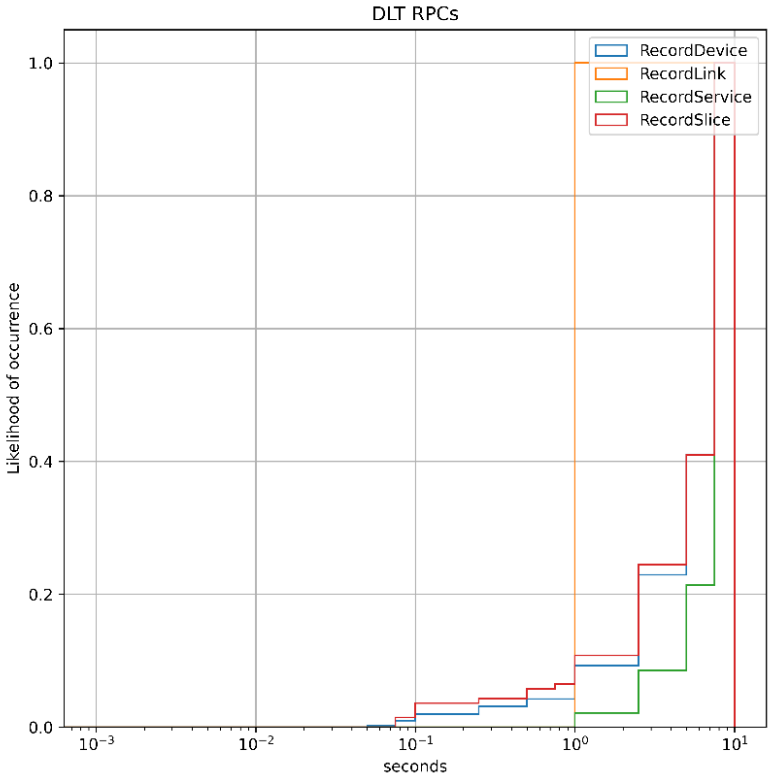


Figure 79. CDF for the DLT Delay

Figure 80 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.

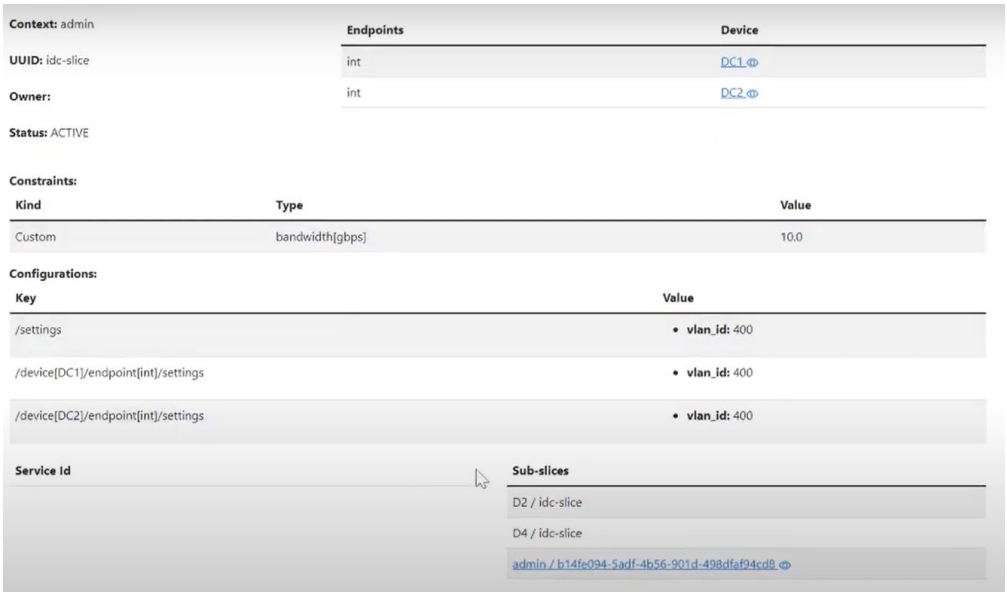


Figure 80. Inter-domain Transport Network Slice that includes sub-slices

Figure 81 provides the details of the local (from the initial domain perspective) requested sub-slice.

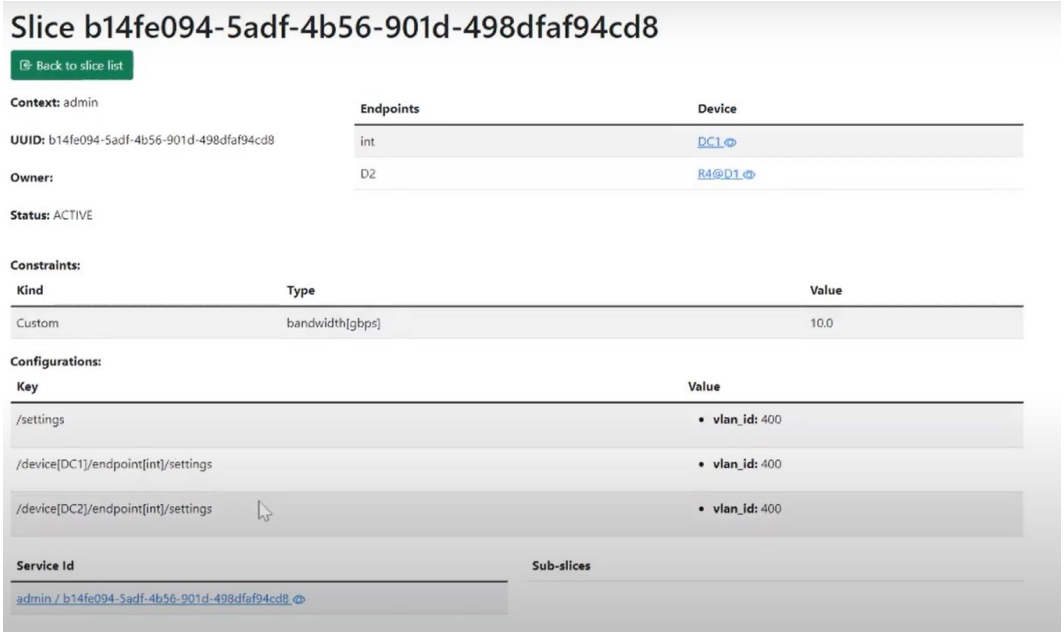


Figure 81. Sub-slice information details

5.3.4. Service/Slice Request Scalability

The performance assessment of TeraFlowSDN's scalability has been carried out at CTTC's testbed (described in section 4.3). The controller deployment included the Context, SBI (former Device), Service, Slice, Path Computation, WebUI and Load Generator components. The scalability feature for the scalable core components (Context, Service, Slice, and Path Computation) has been activated by configuring the Kubernetes Pod Auto-Scaler (HPA) for each one of them. Note that the Load Generator component issues requests directly to the Service and Slice component depending on the settings configured, thus, the NBI (former Compute) component is not required. Indeed, the strategy followed by the Load Generator is the same used by the NBI, thus, the measurements are equivalent.

The HPAs periodically monitor, every 30 seconds by default, the performance of the related components and check configurable metrics and thresholds to trigger the scale-up and scale-down operations. For this test, we configured a maximum of 100 pods (i.e., replicas) per component, and a threshold of 80% over the average CPU utilization metric. That means when the average CPU utilization of the existing replicas for a specific scalable component surpasses the threshold, Kubernetes will automatically deploy additional replicas to cope with the component's load following the standard HPA replication mechanism.

To perform the assessment, we onboarded the Telefonica Spanish network (14 nodes, 44 unidirectional links) depicted in Figure 82. Each node was equipped with 50 client endpoints to be used as request endpoints. We used emulated routers to assess the performance of the TeraFlowSDN controller. The reason for that is to measure how the internal components behave in the worst case for the TeraFlowSDN controller, i.e., the configuration of the underlying network devices does not consume time; thus the TeraFlowSDN controller components do not have spare time to wait and

perform other operations. In practice, we are performing a stress performance assessment where requests must be attended to as fast as possible.

The load generator has been configured to produce 1000 requests randomly mixing types L2 service, L2 slice, L3 service, and L3 slice following a uniform distribution, and choosing the endpoints for the requests using a random uniform distribution. To emulate a load generation coming from multiple tenants, we assumed each tenant, on average, generates 5 Erlangs of load to the system, each corresponding to one configuration request among the aforementioned request types.

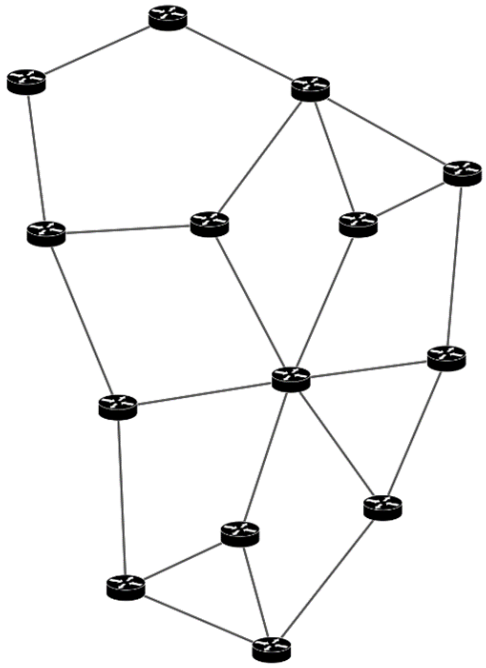


Figure 82 Telefonica Spain Network (14 nodes, 44 unidirectional links)

In our load generator, we can tune the value of offered load and the number of parallel workers generating that load. Given that, we configured the number of workers to be the number of tenants in each experiment. Besides, the value of the total offered load in the experiment is computed as the product between the desired number of tenants and the average offered load per tenant. We assumed a mean hold time of 10 seconds. We computed the mean inter-arrival time for each experiment according to this value of the holding time and the variable total offered load. Both the hold time and the interarrival time follow an exponential distribution.

Additionally, the load generator uses a uniform distribution to choose the capacity (0.1 Gbps and 100 Gbps), the maximum end-to-end latency (between 5.0 ms and 100.0 ms), and availability (between 0.1% and 99.9999%).

Table 14 enumerates the set of scalability experiments carried out. The table also includes the duration of the experiment. As can be seen, when the offered load and the number of tenants increase, the TeraFlowSDN automatically scales the pods per component to meet the requirements. Figure 83 illustrates the peak number of pods (replicas per component) the Kubernetes HPA configured for the scalable components.

Even when the total load offered increases, the duration of the experiments decreases while the number of pods deployed increases. The duration of the experiments reaches a minimum value at

experiment #5 (20 tenants and 100 Erlang of offered load) and later it starts slightly increasing. The interesting point is that even when relationship between offered load and experiment duration does not increase, the system keeps serving the requests. We computed the number of Erlangs processed per second and the value is monotonically increasing, showcasing the capacity of the TeraFlowSDN to scale beyond 100 concurrent tenants interacting with the TeraFlow controller.

Table 14. Scalability Experiment Configurations

Exp #	Settings		Duration	Erlangs processed per Second
	Tenants	Off. Load		
1	1	5	39 min 21 sec 650 msec	0,002
2	5	25	09 min 33 sec 650 msec	0,044
3	10	50	06 min 02 sec 487 msec	0,138
4	20	100	04 min 42 sec 198 msec	0,354
5	40	200	04 min 58 sec 573 msec	0,668
6	60	300	04 min 52 sec 663 msec	1,022
7	80	400	04 min 52 sec 666 msec	1,362
8	100	500	05 min 23 sec 440 msec	1,546
9	120	600	06 min 04 sec 313 msec	1,647

Analyzing the number of pods deployed per scalable component (Figure 83) depending on the offered load (in Erlangs) we see that the number of pods reaches a maximum at experiment #6; beyond that point, the Kubernetes HPAs decide not to scale the Context component, and the number of replicas of the Service, Slice and Path Computation components increase, but not significantly. The reason for that could be a bottleneck in the underlying database supporting the Context component's data. Note that all the components, in one way or another, depend on the Context component to store and retrieve data. To analyze that behavior, we first analyze the response time of the TeraFlowSDN controller, and then we inspect the performance reports of the CockroachDB database used by the Context component.

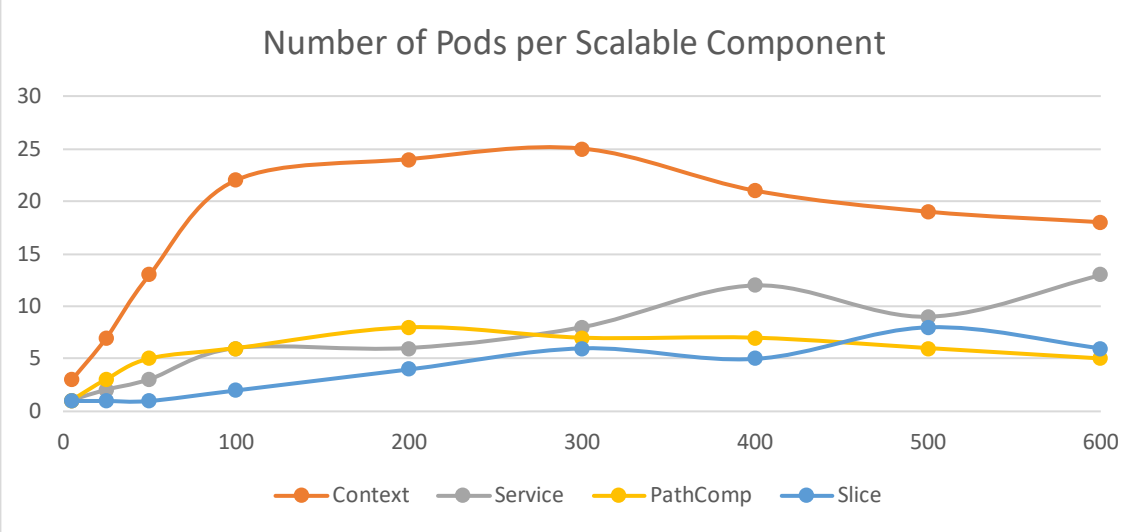


Figure 83. Scalability - Number of Pods per Component

Next, we analyze the response time of the TeraFlowSDN controller under stress. The measurements are taken at the Load Generator. The set of figures between Figure 84 and Figure 87 depict the

response time for the setup operation and for each type of request configured; in particular, for L2 services (Figure 84), L2 slices (Figure 85), L3 services (Figure 86), and L3 slices (Figure 87).

As expected, setting up services takes less time than setting up slices, and L2 services/slices take less time than L3 counterparts. The rationale is that a service backs each slice, thus, the response time for setting up a slice also includes the time to set up a service. Regarding the extra time required to set up L3 services/slices, L3 services require configuring a larger number of rules in the network devices than their L2 counterparts, thus consuming additional processing time. For L2 services in general, 40% of them take less than 1 second, and for low offered loads (<50 Erlang), while around 90% of executions are completed in less than 2 seconds. When the load increases, the response time increases proportionally, reaching up to a few tens of seconds at the maximum offered load considered in the experiment. Similar behavior can be observed for L3 services. Regarding L2/L3 slices, the difference is the increase in managing in the database of the slice entity, which is represented as an addition of hundreds of milliseconds in the overall response time. However, even at high loads, the controller can complete the requests within 10 seconds in around 90% of the cases.

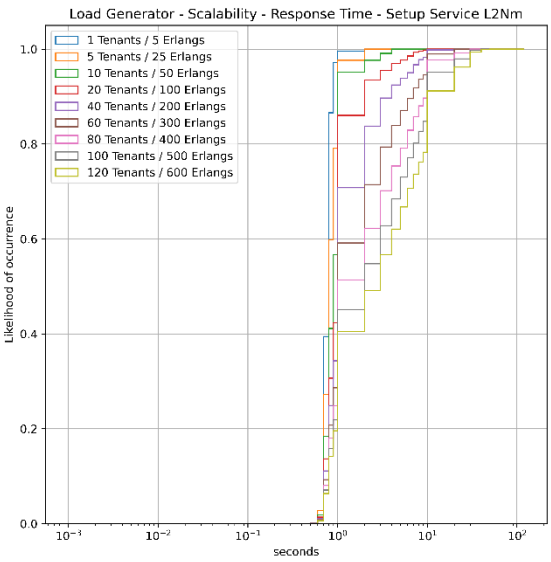


Figure 84. Scalability – Setup L2 Service – Response Time

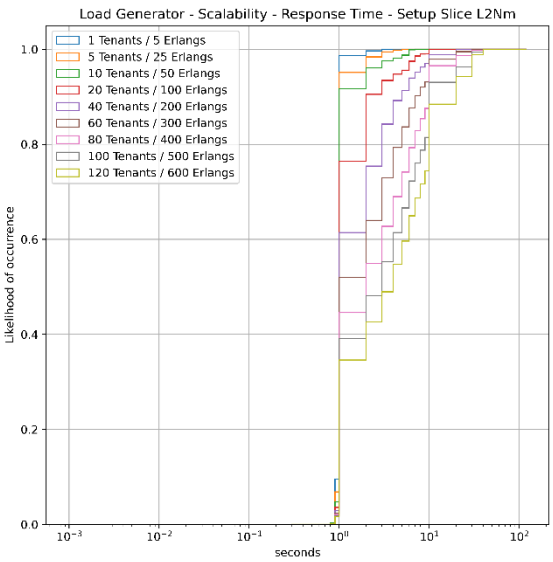


Figure 85. Scalability – Setup L2 Slice – Response Time

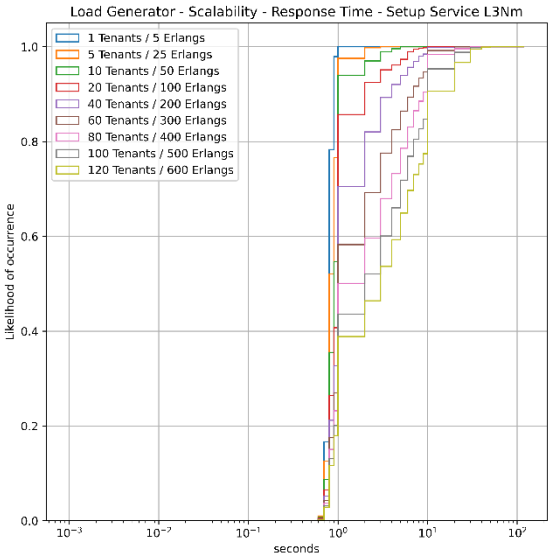


Figure 86. Scalability – Setup L3 Service – Response Time

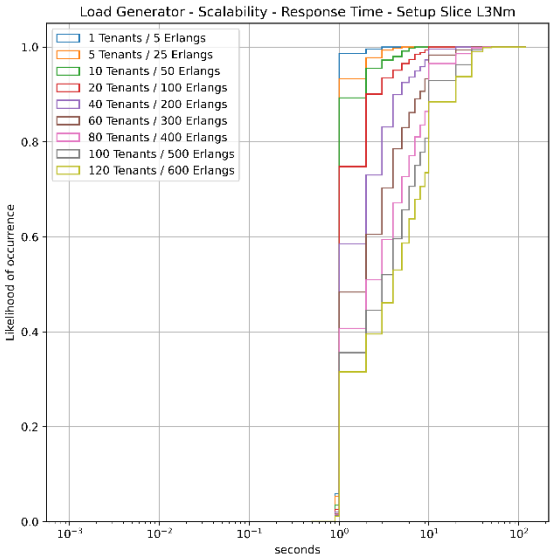


Figure 87. Scalability – Setup L3 Slice – Response Time

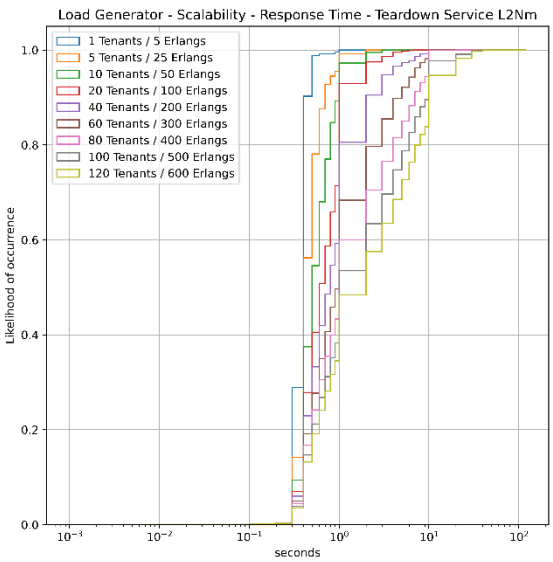


Figure 88. Scalability – Teardown L2 Service – Response Time

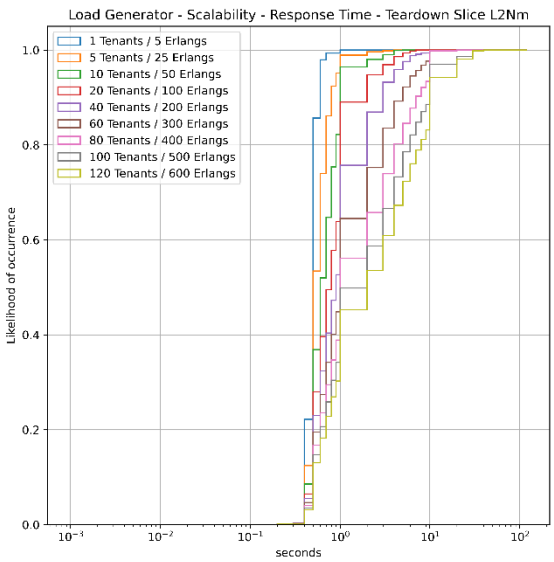


Figure 89. Scalability – Teardown L2 Slice – Response Time

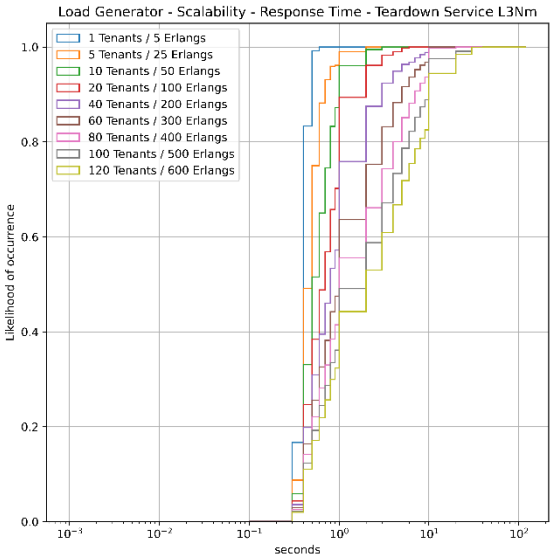


Figure 90. Scalability – Teardown L3 Service – Response Time

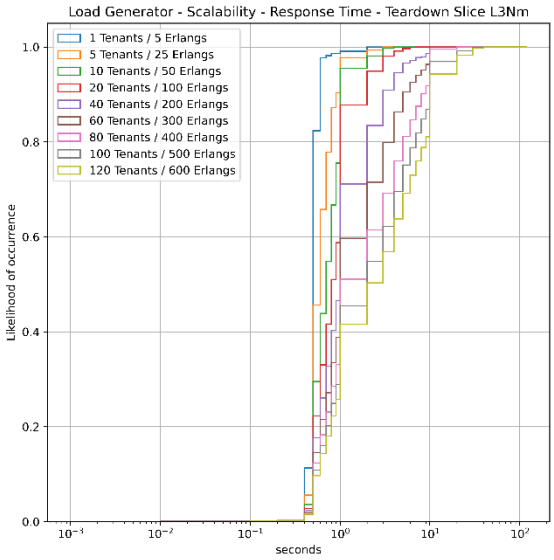


Figure 91. Scalability – Teardown L3 Slice – Response Time

Now we analyze the teardown response time of the TeraFlowSDN controller under the same stress conditions as for the setup. Again, the measurements are taken at the Load Generator. The set of figures between Figure 88 and Figure 91 depicts the response time for the teardown operation and for each type of request configured, i.e., L2 services (Figure 88), L2 slices (Figure 89), L3 services (Figure 90), and L3 slices (Figure 91).

The teardown response time is similar to the ones for the setup. Tearing down services takes less time than tearing down slices, and L2 services/slices take less time than L3 counterparts. The rationale is the same as used to explain this for the setup operation. Note that the teardown operation is generally faster than the setup, given that no path computation is required. Removing entities in the database is faster than storing them, given that fewer constraints must be enforced.

For L2 services in general, 50% of requests take less than 1 second, and for low offered loads (<50 Erlang), about 95% of executions are completed in less than 2 seconds. When the load increases, the response time increases proportionally, reaching up to a few tens of seconds at maximum load used for this study. A similar behavior can be observed for L3 services. Regarding the L2/L3 slices, essentially, the increase in time is due to managing the slice entity in the database. It corresponds to an extra hundreds of milliseconds to the overall response time. However, even at high loads, the controller can complete the requests within 10 seconds in about 95% of the cases.

As a final step of the scalability performance assessment, we analyzed the performance of distributed database the Context component uses to manage and persist the TeraFlowSDN controller information. CockroachDB is a distributed and scalable database using the NewSQL data model. The latter implements four key properties of relational database transactions on top of distributed and scalable databases. These key properties are:

- *Atomicity*: all statements in a transaction are executed as a whole or not executed, but no partial transaction is executed.
- *Consistency*: each transaction moves the database content from a consistent state to another consistent state preserving all the invariants and constraints of the database.
- *Isolation*: even when transactions are executed concurrently, multiple reads/writes on a single table leaves the data in the same state it would be if transactions were executed sequentially.
- *Durability*: once a transaction is committed, even in the case of a system failure, the information will be preserved and kept consistent.

Note that even when such databases bring enhanced scalability, they might suffer from additional overheads, given that they need to keep consistency across the distributed nodes forming the database cluster.

To analyze the performance of the CockroachDB database, we recorded directly from the CockroachDB cluster dashboard the performance metrics during the execution of the experiments. We captured four metrics:

- The latency between the nodes forming the cluster.
- The number of SQL statements (queries) executed per second.
- The 99th percentile service latency for SQL statements.
- The SQL statement contention.

The latency between the nodes forming the cluster is illustrated in Figure 92. It is a key performance metric since all transactions need synchronization and consensus between the cluster nodes. For this reason, this value directly affects all the database operations. Reasonable values with a mean value of 0.74 ms and a low standard deviation of a few tenths of milliseconds are observed. Besides, no connectivity interruptions are identified.

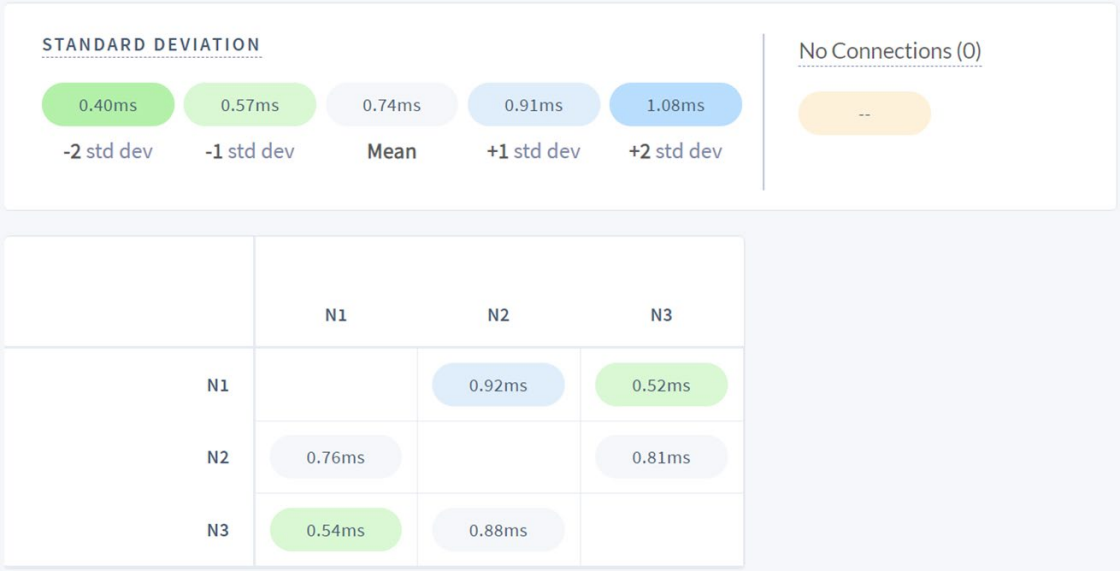


Figure 92. Scalability – CockroachDB – Latency between the nodes forming the cluster.

The other three metrics are plotted in a stacked and time-aligned manner in Figure 93, to help identify each experiment. The vertical dashed lines represent the (approximate) beginning of each experiment. The first plot illustrates a moving average of the number of SELECT (dark blue), INSERT (red), UPDATE (yellow), and DELETE (cyan) statements successfully executed per second across all cluster nodes. The second plot depicts the service latency, i.e., over the last minute, the nodes executed 99% of SQL statements within this time (each color corresponds to one node in the cluster). This time only includes SELECT, INSERT, UPDATE and DELETE statements and does not include network latency between the node and client. The third plot showcases a moving average of the number of SQL statements executed per second that experienced contention across all nodes.

As the plots demonstrate, until experiment #4, the system copes well with the load. Starting from experiment #5, the number of queries per second increases, the 99th percentile service latency, and the average number of queries suffering from contention. To some extent, such contention and service latency limit but do not prevent the system's scalability.

As an outcome, future releases of the TeraFlowSDN controller might address the improvement of the database schema to reduce the contention levels and optimize the database queries. This activity is outside the scope of the current project.

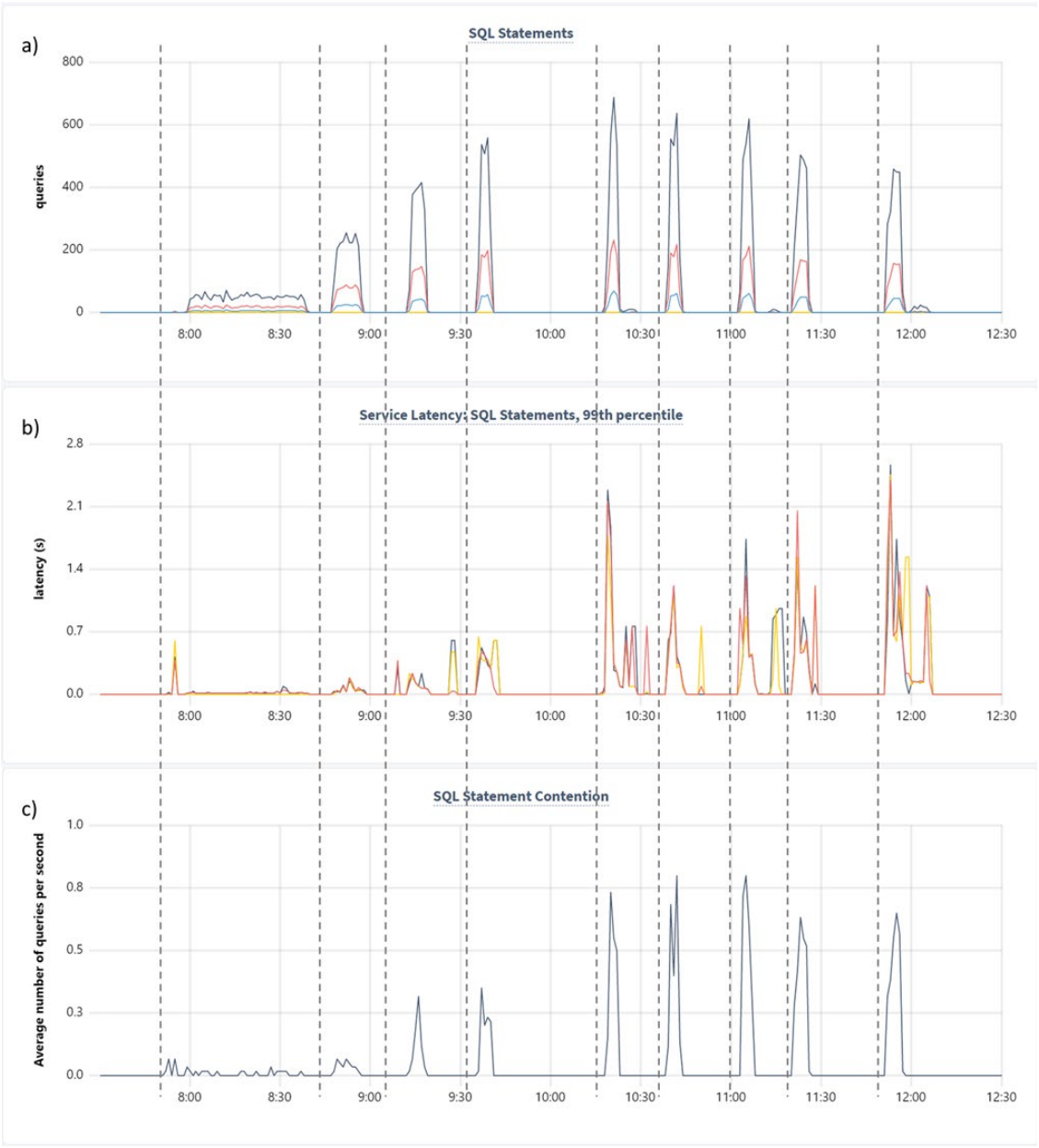


Figure 93. Scalability – CockroachDB – (a) SQL Statements, (b) 99th Percentile SQL Statement Latency, and (c) SQL Statement Contention.

Final KPI measurements:

Name	Value	Measurement	Comment
Multi-tenancy	>100 tenants	TeraFlowSDN can support more >100 tenants with reasonable service latencies.	The response time might increase at high loads due to database latency and SQL query contention. Future releases might study how to enhance the database schema.

5.3.5. Location-aware Service Updates

This section discusses the deployment of End-to-End packet-optical connectivity services over an edge-cloud continuum, considering the importance of location awareness. It proposes an architecture, data models, and placement algorithms for provisioning and updating these services in the ADRENALINE Testbed using the ETSI TeraFlowSDN controller.

Several challenges must be overcome when looking at the deployment of Cooperative, Connected and Automated Mobility (CCAM) services over a distributed edge and cloud infrastructure [CTTC22Par]. Firstly, we need unified computing, storage, and networking resources management. In this respect, the End-to-End SDN Controller (also known as SDN orchestrator, e.g., TeraFlowSDN), together with an NFV orchestrator (e.g., OpenSource MANO), will be able to deploy integrated services (i.e., to provision cloud/edge computing resources and connectivity between them) and optimize the cloud and network resources (i.e., packet/optical) concurrently. Secondly, we must address multi-domain networking, where resources must be assigned in each technological domain and combined for an end-to-end service.

Future services are deployed close to the end user, which requires an edge-cloud continuum. This concept refers to a distributed computing infrastructure spanning the network's edge (where end devices, e.g., sensors, smartphones, and IoT, reside) to the cloud (where data centers and servers reside). It encompasses a range of resources and services that can be utilized to process and store data closer to the source, providing low-latency and high-bandwidth connectivity, while also leveraging the scalability and processing power of cloud computing. The introduction of edge-cloud continuum significantly impacts Vehicle-to-Everything (V2X) scenarios, where latency aspect plays a significant role.

V2X technology is revolutionizing the transportation industry by enabling vehicles to communicate with other vehicles, infrastructure, and pedestrians. Location is a critical component of many V2X services as it provides contextual information about the surrounding environment. One example is collision avoidance, where vehicles can exchange location information to avoid collisions.

This is particularly important in high-density traffic areas, where drivers may be unable to see approaching vehicles. Another example is over-the-air software updates, where the vehicle's location ensures that the update is delivered to the correct vehicle. This helps to avoid errors that can occur if the update is sent to the wrong vehicle. Overall, location is vital to many V2X services and is crucial in improving road safety, efficiency, and convenience.

The location also plays a significant role in hierarchical computing architectures. This emerging technology can transform computing by allowing the orchestration of end devices, the edge, and the cloud. A survey by Ren et al. [Ren19] compares different network computing paradigms and discusses different research issues such as computation offloading, caching, and location awareness.

Introducing location awareness in end-to-end connectivity services and network topologies is indispensable. This section considers the provisioning and updating of end-to-end connectivity service, considering that location-aware connectivity services might need service endpoint migration due to the dynamic nature of joint edge-cloud continuum and the provisioning of services to moving end users.

This section presents the proposed architecture for the provisioning and updating the aforementioned connectivity services, as well as details of the augmented TeraFlowSDN protocol buffers. Moreover, a location allocation algorithm is presented to provide endpoint selection based on location. Finally, we

have experimentally evaluated the proposed solution in the ADRENALINE Testbed using the ETSI TeraFlowSDN controller [OFC22].

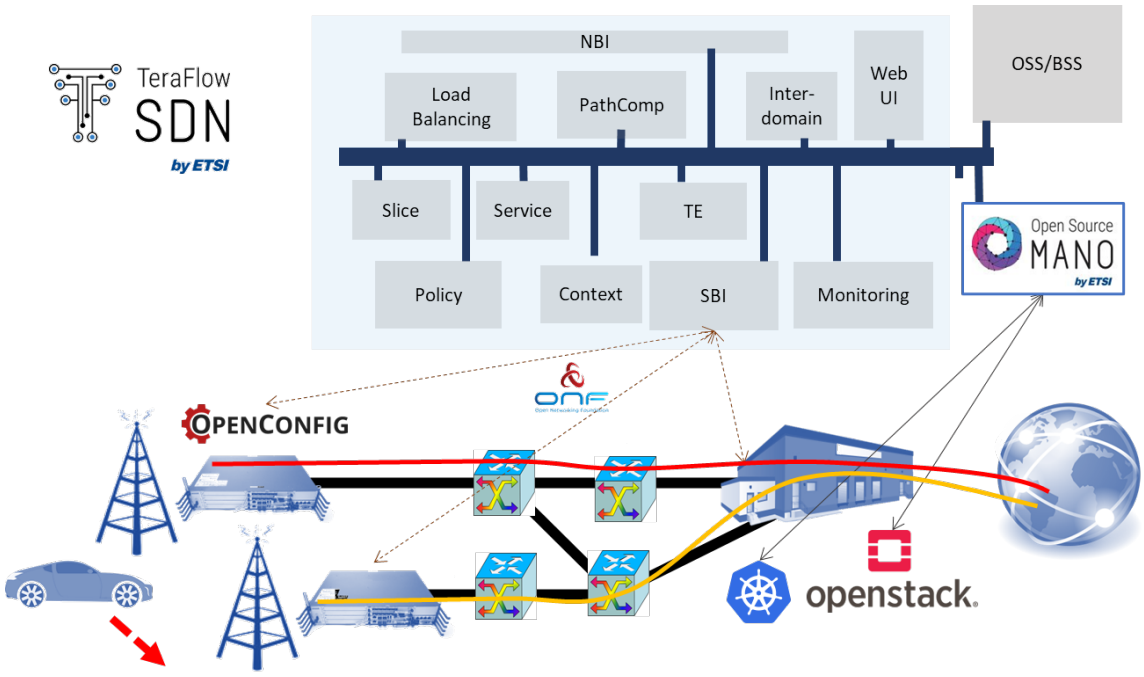


Figure 94 Architecture for E2E location-aware services, including SDN controller necessary components

The proposed architecture is depicted in Figure 94. On top, we have the internal cloud-native architecture of ETSI TeraFlowSDN, while below we can see the controlled multi-domain scenario, that considers E2E connectivity services. It divides the main components of the SDN controller into micro-services dedicated to specific functionalities of the SDN controller. TeraFlowSDN provides extended support for OpenConfig-based routers and interaction with optical SDN controllers through the ONF Transport API.

The scenario in Figure 94 shows a connected vehicle that might be consuming services from a specific edge computing node, that in turn is connected to a DC service. As the vehicle moves, the services might be migrated to another edge computing location. This triggers the update of the connectivity services for the E2E connection towards the DC services.

In a basic topology, we have included the GPS location for three different service endpoints connected to the edge nodes. The user node can then connect to the DC by means of the closest edge node, depending on its present location, which is introduced as a constraint to the connectivity service request/update.

The introduction of location awareness in transport networks (i.e., including packet and optical domains) requires the introduction of novel concepts to several SDN-based data models. The most affected one are the data models related to service endpoints, which refer to the network endpoints where connectivity services might be requested. Moreover, connectivity service also needs to include a new constraint based on location coordinates or specific region. The proposed modified data models for internal TeraFlowSDN usage are depicted in Figure 96.

The workflow of provisioning and updating a location-aware service in Figure 95 is described in two phases (service establishment and follow-me service). Service establishment starts with the request to create a new connectivity service between a specific endpoint (e.g., DC location) and the closest endpoint to a location (step 1). TeraFlowSDN controller NBI processes the request and forwards it with TeraFlowSDN internal data model to Service component (step 2). Location constraints and network topology endpoints are matched to best provision the necessary endpoints depending on the location. The E2E connection is created following the workflow described in [OFC22], and properly notified (steps 3-4).

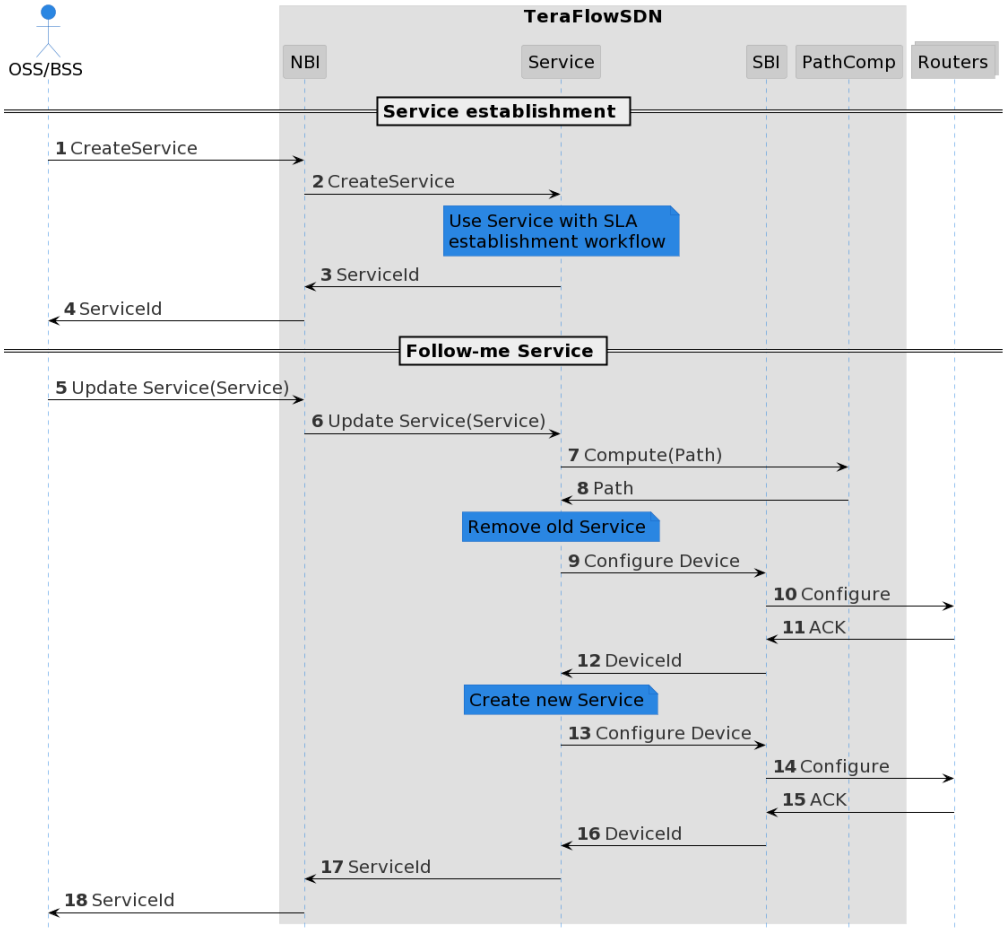


Figure 95 Sequence diagram of provisioning and update of a location-aware service

The follow-me service phase starts with an update of the provided service, including the new location (steps 5-6). A new endpoint is computed, and then a new path is calculated (steps 7-8). Then the service is updated following a break-before-make strategy. The old service is removed (steps 9-12), and the service with updated endpoints is provisioned (steps 13-16). Finally, a response is provided to the user (steps 17-18).

To provide location-aware services to TeraFlowSDN, three components have been modified. First, the Context module database has been extended to support the data models shown in Figure 95. The Device module has also been extended to retrieve the location data from the network devices. Finally, the algorithm to retrieve the closest access node was implemented in the Service module.

```
message GPS_Position {
  float latitude = 1;
  float longitude = 2;
}
message Location {
  oneof location {
    string region = 1;
    GPS_Position gps_position = 2;
  }
}
message EndPoint {
  ...
  Location endpoint_location = 5;
}
```

```
message Constraint_EndPointLocation {
  EndPointId endpoint_id = 1;
  Location location = 2;
}
message Constraint {
  oneof constraint {
    ...
    Constraint_EndPointLocation
endpoint_location = 3;
    ...
  }
}
message Service {
  ...
  repeated Constraint service_constraints = 5;
  ...
}
```

Figure 96 Internal SDN controller modified data models to support location-aware services.

The proposed algorithm aims to create a service between a data center (DC) and a user by identifying the nearest access edge node to the user's location. To achieve this, the algorithm computes the geographical distance between each of the access edge nodes and the user's location provided in the connectivity service request. Geographical distance can be computed using various methods. One commonly used approach is to calculate the distance between two points on the Earth's surface using the Haversine formula, which considers the radius of the Earth and the latitude and longitude of the two points to calculate their distance. By selecting the access edge node closest to the user, the algorithm can establish a connection between the user and the network with minimal latency and increased performance.

Figure 97 shows the Wireshark capture of creating a location-aware connectivity service and is updated with a new location. The 1st packet corresponds to creating the connectivity service, while the 2nd packet corresponds to its acknowledgment. The 3rd packet corresponds to the update and actual provisioning of the connectivity service, while the 4th packet corresponds to its acknowledgement.

Time	Info
REF	SETTINGS[0], HEADERS[1]: POST /service.ServiceService/CreateService, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
0.037609	HEADERS[1]: 200 OK, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), HEADERS[1], WINDOW_UPDATE[0]
0.041850	HEADERS[3]: POST /service.ServiceService/UpdateService, WINDOW_UPDATE[3], DATA[3] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
1.258795	HEADERS[3]: 200 OK, WINDOW_UPDATE[3], DATA[3] (GRPC) (PROTOBUF), HEADERS[3], WINDOW_UPDATE[0]
REF	HEADERS[3]: POST /service.ServiceService/UpdateService, WINDOW_UPDATE[3], DATA[3]
0.601359	HEADERS[15]: POST /context.ContextService/RemoveConnection, WINDOW_UPDATE[15], DATA[15] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
0.648959	HEADERS[17]: POST /context.ContextService/RemoveService, WINDOW_UPDATE[17], DATA[17] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
0.736901	HEADERS[21]: POST /context.ContextService/SetService, WINDOW_UPDATE[21], DATA[21] (GRPC) (PROTOBUF)
1.898406	HEADERS[13]: POST /context.ContextService/SetConnection, WINDOW_UPDATE[13], DATA[13] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
2.030466	HEADERS[3]: 200 OK, WINDOW_UPDATE[3], DATA[3] (GRPC) (PROTOBUF), HEADERS[3], WINDOW_UPDATE[0]

Figure 97 Wireshark capture of a location-aware service provision and update

After the user node has changed its position, we ask the TeraFlowSDN NBI to update the connectivity service in the 5th packet. The 6th packet corresponds to the removal of the connection that the connectivity service has originated, while the 7th packet is about the removal of the connectivity

service itself. The 8th packet corresponds to the setting up of the new connection while the final 9th packet is the ACK to the update of the service.

A hundred connectivity services have been requested via the TeraFlowSDN NBI to test the speed of the algorithm responsible for selecting the closest node. As shown in Figure 98, all requests were finished between 98 ms and 172 ms, fast enough to fulfil most use cases, with a mean of 0.137 ms and a standard deviation of 0.0185 ms.

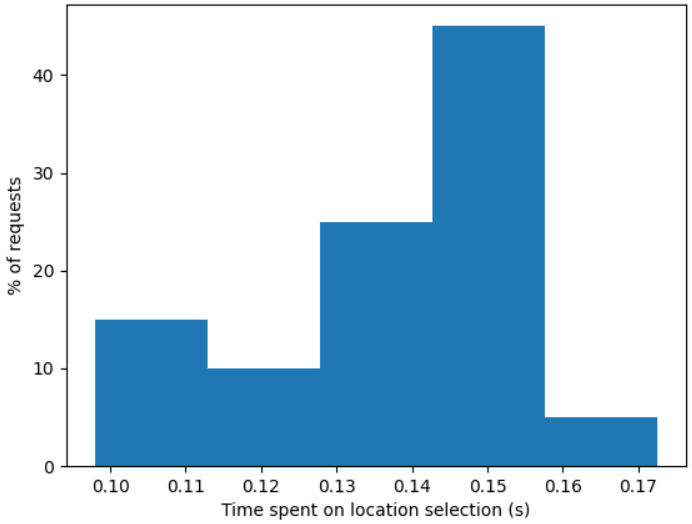


Figure 98. Histogram of the time spent on the location selection algorithm

Name	Value	Comment
Positioning	100%	We have validated location-awareness in end-to-end connectivity services and in network topologies. TeraFlowSDN has been extended with an augmented data model for topology and connectivity services to include GPS coordinates and Regions into service endpoints and connectivity service constraints. The proposed architecture considers the requested end-to-end connectivity service provisioning and update, considering that location-aware connectivity services might need service endpoint migration due to the dynamic nature of joint edge-cloud continuum.

5.3.6. Latency budgets as function of the application requirements

This section shall be included as part of research on T4.3 topics. It provides interesting results, and as D4.2 was delivered, we have considered to include in this part. Networked applications broadly range concerning their QoS requirements. While purely bandwidth-hungry applications like file downloads or video streaming are less sensitive to delays, interactive applications like video conferencing or telephony are much more sensitive to delays. This heterogeneity of applications is illustrated in Figure 99, where the Mean Opinion Score (MOS) - QoS in terms of available bandwidth and experienced delay is plotted against the resulting QoE for a total of four applications. In addition to a TCP-based

adaptive video streaming application and a UDP-based Voice over IP (VoIP) application, we perform what-if analyses involving applications with significantly stricter QoS requirements to mimic emerging and more demanding applications such as the ones involving haptic feedback [ZHA2018]. To this end, we use the QoS characteristics of the VoIP application, increase its bandwidth requirements and usage by a factor of 10, and increase its delay sensitivity by factors of 10 (“Delay Sensitive x10”) and 20 (“Delay Sensitive x20”), respectively.

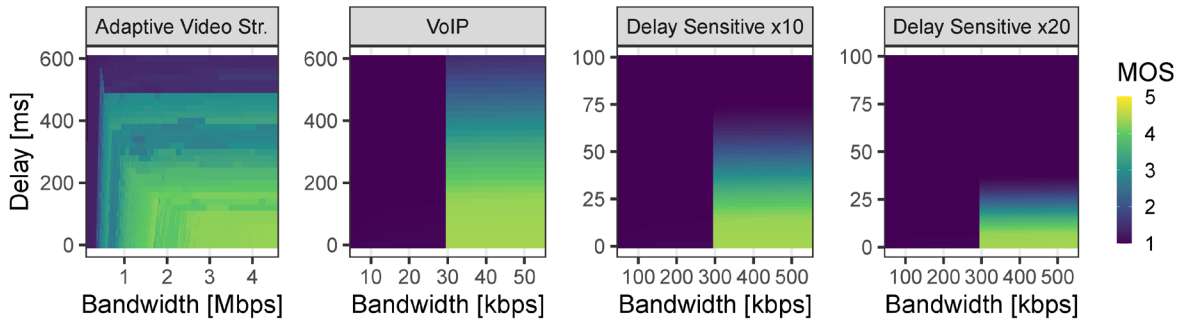


Figure 99. QoS-to-QoE relationship for exemplary applications. While the ITU-T P.1203 model is used for adaptive video streaming, the ITU-T G.107 e-model is employed in case of VoIP and extrapolated towards delay sensitive apps.

Several observations can be made. First, the relationship is clearly application-specific, both in terms of the sensitivity towards QoS changes as well as in terms of the absolute requirements. In terms of delay, the investigated adaptive video streaming applications is able to cope with delays up to 500ms. Higher delays impact the performance of the respective control loops, i.e., the adaptive video streaming control loop and the TCP control loop. For VoIP, delays in the area of 300 to 400 ms significantly affect the voice call's quality. Haptic applications, as emulated by the “Delay Sensitive x10” and “Delay Sensitive x20” require much lower delays, namely below 40 and 20 ms, respectively. Second, applications exhibit different degrees of elasticity: while the video streaming application smoothly transitions between video quality levels, the VoIP application under consideration is inelastic and abruptly degrades from a good or acceptable QoE level to the worst level as soon as its bandwidth requirement is not met. Third, diminishing returns occur, i.e., past a certain application-specific point, lowering an application's delay or increasing its bandwidth slows and eventually stops improving its QoE. Additional QoE impact factors such as jitter, packet loss, and subjective user-specific properties are deliberately omitted in this work to keep a clear focus on the general ideas of the proposed concepts.

In the following section, we discuss the possibility of using QoS parameters like delay and bandwidth and including them in a public Internet setting. Network operators are facing challenges in multiple directions. Resources are required dynamically to cope with increasingly varying demands, systems become less predictable due to the heterogeneity of services and applications, and control plane complexity will grow with multi-dimensional Service Level Agreements (SLAs). Furthermore, service delivery expectations are growing from the vertical sectors, the Online Application Providers (OAPs), and the consumer side towards a set of smart and specialized connectivity services (SCS) on-demand that are universally and equally provided.

5.3.6.1. Service Concepts

In this section, we discuss the requirements and main characteristics of service concepts and key principles around SCS to enable an evolution towards smart Public interconnected networks and

services (PINS). Furthermore, we identify key elements of the networking ecosystem that need attention to address the above challenges related to the long-term success of the service concepts. Our approach is inspired by "removing complexity and aiming towards simplicity".

Our discussion of service concepts revolves around two aspects relating to treating network traffic, traffic modes, traffic aggregates, and connectivity handling. In the following, we define and discuss these notions in detail.

Traffic Modes: due to the variety of applications in terms of their QoS demands and degrees of sensitivity and elasticity, we argue that diverse traffic modes which reflect this heterogeneity are required for efficient traffic handling. Relative differentiation between flows and absolute differentiation with strict performance guarantees can be performed depending on the fair amount of complexity, particularly control plane complexity. This way, QoS resources can be adjusted to reduce queueing delays for delay-sensitive traffic while identifying more delay-tolerant portions of the traffic that might be re-routed via longer paths. Evolving from the "best-effort" traffic mode of today's Internet, we propose and discuss four main traffic modes. These include three best-effort modes that differ relatively from each other and allow for more nuanced differentiation while retaining the benefits of best-effort handling and an assured quality mode that provides strict performance guarantees.

From today's perspective, the current best-effort mode could be labeled "Basic Quality (BQ)". We suggest an "Improved Quality (IQ)" mode relative to the BQ mode. The IQ mode improves the quality or performance relative to the BQ mode by mechanisms like Weighted Fair Queueing (WFQ) or routing via a shortest path. This may result in improvements along multiple dimensions of the connectivity, in principle, any combination of improved throughput, (queueing) delay, jitter, or packet-loss performance. Moreover, we foresee a "Background (BG)" traffic mode that provides connectivity with more relaxed quality properties than the BQ mode. This mode enables the NSP to provide valuable connectivity offerings while allowing a higher utilization of network resources. Examples of applications using the BG, BQ, and IQ modes could be the automatic download of an Operating System (OS) update, an on-demand video stream, and a live video stream with increasingly strict QoS requirements while not necessarily being mission-critical and requiring strict performance guarantees.

Since throughput is less of an issue in backbone networks, the primary difference among the three best-effort modes is along the delay performance, where IQ offers lower end-to-end latency than the BQ mode, and the BG mode supports more relaxed latency requirements. The BG, BQ, and IQ traffic modes can be considered multi-level best-effort.

The fourth suggested traffic mode, called "Assured Quality (AQ)" mode, offers strict performance guarantees. This mode is used if the client needs significantly higher or more stable network performance than available by the IQ mode. This will require more complex mechanisms than those of the IQ mode. To enable the AQ mode, the QoS, resource, and admission control mechanisms must be realized at a finer granularity than those for the IQ mode.

Traffic Aggregates and Scalable Connectivity Handling: we observe that provisioning connectivity resources for each individual traffic flow in an on-demand and end-to-end fashion is not feasible in terms of scalability, complexity, and timeliness. Hence, we suggest introducing multiple granularity levels of traffic aggregates that differ w.r.t. their size, lifetime, and mode of instantiation. In this section, we discuss a two-level example. At the coarse-grained level, we propose Managed Quality Paths (MQPs) that are high-capacity, long-lived, and pre-established paths between major interconnection points. In this context, multi-domain challenges related to security, information

exchange, fairness, and neutrality must be addressed. At the fine-grained level, we envision SCS Sessions that are expected to be highly dynamic, on-demand, and between endpoints. In the context of these sessions, only paths connecting the endpoints to suitable interconnection points need to be provisioned whereas the remainder can be car- ried by a suitable, already provisioned, and well-dimensioned MQP. These pre-established paths also help reduce the solution space's size and therefore allow for faster handling of connectivity requests. Depending on the specific requirements of an SCS, different traffic modes might be employed at both the MQP and SCS session-level. Figure 100 presents the main concepts and key infrastructure elements elaborated further in the following. We also show an exemplary SCS flow between the two highlighted endpoints to illustrate the core ideas.

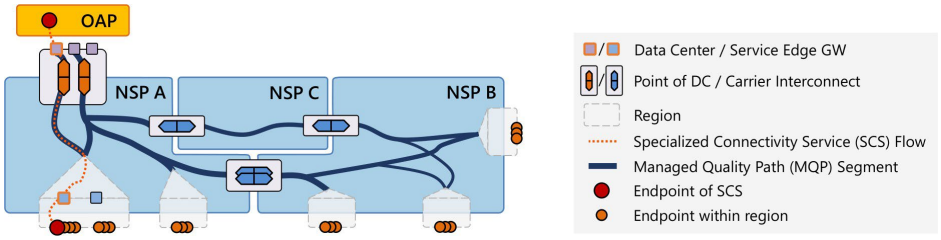


Figure 100. Multi-domain scenario: managed quality path infrastructure with exemplary specialized connectivity service

Peering and transit services in today's Internet connect very large, coarse-grained regions and have only rudimentary SLAs. Evolving from these services, we propose the MQP as a Point-of-Interconnect-to-Region (PoI2R) interconnection service, where the notion of "region" can be in a spatial/geographical or technological sense, e.g., a range of IP prefixes. The core idea of the MQP service is to enable dynamic traffic engineering, intelligent management, and configuration of coarse traffic aggregates and their services, and it may also support remote peering.

Since SCS sessions use MQP for PoI2R connectivity, only paths from end-users and application servers to endpoints of the corresponding MQP need to be provisioned. These endpoints are Data Center Gateway (DC GW) and Service Edge Gateway (SEG), respectively. To achieve QoS handling and charging support for SCS between end-user devices and OAP end-points in data centers, we expect the necessity for signalling and business relationships between OAPs, NSPs, and end-user devices. With the proposed two-level approach, we expect that setting or merely checking policies at the gateways is sufficient to cover SCS needs. Hence, signalling and QoS handling can be substantially simplified compared to mechanisms such as IntServ that require setting policies on each network element along the entire end-to-end path.

5.3.6.2. Main Solution Elements and Challenges

In this subsection, we identify key aspects of the networking ecosystem that require attention to pave the way towards smart PINS and enable the discussed service concepts.

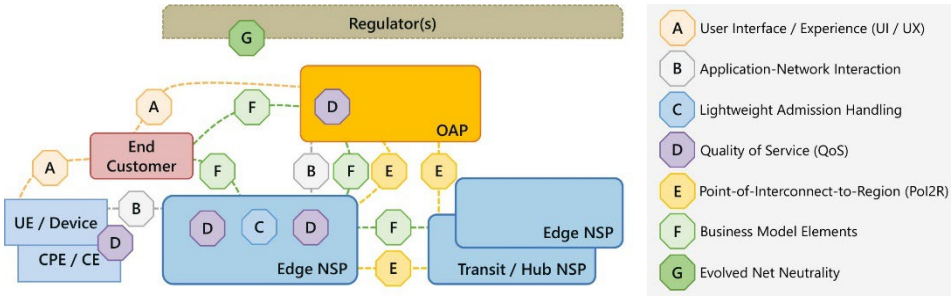


Figure 101 Overview of solution elements

We also highlight related research challenges labeled RC. Following the principles introduced above,

we cover seven key topic areas that all need to be addressed in the long term and in a holistic, coordinated, and interdependent way. A compact overview of these areas and corresponding solution elements is provided in Figure 101. While we discuss all seven areas from (A) to (G), we emphasise three that we consider critical already in the bootstrapping phase. These include (B) Application-Network Interaction (ANI), (C) Lightweight and Class-based Admission Handling (LAH), and (F) Business Model Elements (BME).

User Interaction / Interface (UI) / User Experience (UX): The customer shall control the service level selection. This can be achieved through precise control via UI and UX dialogues or implicitly based on relevant characteristics of the environment and end devices in use. We expect a common approach to service level expectations and indicators (RC 1) for this.

ANI: The goal of ANI is to allow expressing requests for SCS and corresponding NSP offerings. To this end, the NSP could provide SCS templates expressing possible QoS value ranges - e.g., target and lower bound - regarding supported throughput, latency, and packet loss. A solution should anticipate that the elasticity of applications can still be an important feature and support (re-)negotiation in case the desired quality level cannot be delivered (RC 2). Ideally, the SCS templates and the Application Programming Interfaces (APIs) for ANI should be standardized to ensure scalability, portability, and efficiency for the application developer. In order to guide and assist the application's creation of the SCS request, the ANI should also enable applications to query the network for available SCS profiles that can be delivered on-demand or at/for a specific time. This way, the application can make requests with a high likelihood of success instead of a best-effort approach. Similar application-initiated reservation strategies have been proposed for Software-Defined Networking (SDN)-based networks in the context of participatory networking [9]. Additionally, IETF efforts regarding Network Service Headers (NSH) and Application-Aware Networking (APN) provide building blocks toward achieving ANI.

LAH: According to the SCS session handling introduced above, we suggest a light-weight admission handling approach that is still sufficient to achieve the needed traffic handling targets. A fundamental objective is to ensure that the volume of admitted SCS sessions will not make the BQ mode and general quality level suffer beyond specific committed network performance levels. The concept of class-based admission handling is proposed and can be based on trust, logical network IDs, and per-class SCS treatment. Still, the NSP can monitor application traffic rate and behavior per OAP and perform class-based and per-OAP policing by various means in a scalable way. Numerous QoS and admission control mechanisms are available at different levels of granularity in the access parts of 4G and 5G. They can be aligned with the overall end-to-end approach.

QoS: QoS mechanisms are available at multiple locations within the User Equipment (UE)/device, edge NSP, and OAP domains. These mechanisms allow (re-)aligning the allocation of available resources with users' requirements. In particular, deterministic, time-sensitive, and high-precision networking advances can be leveraged to address delay requirements.

PoI2R: Several variants of the PoI2R / MQP interconnection service are needed to cover the NSP-OAP and NSP-NSP segments that form SCS. In the context of NSP-NSP segments, an NSP is referred to as transit NSP if it does not contain any endpoint of the SCS.

BME: Along with the ANI and PoI2R service variants and enablers, BME and charging principles should be defined. Here, we underline the need to support hybrid money flows from the customer to both the OAP and NSP and optionally support Initiating Party Network Pays (IPNP) for two-way connectivity across NSPs. Note that at the PoI2R/MQP level, the BME would be based only on traffic aggregates and should support settlement-free and transit-only BMs. Moreover, we anticipate that pricing models towards end-customers, in particular, will evolve to ensure a correspondence between the price paid and the resources used, hence incentivizing responsible resources use and "Green ICT".

Evolved Net Neutrality (eNN): Finally, we anticipate the need to evolve the NN Regulation to ensure that open and equal access to SCS can be supported at the global Internet scale. This will be critical to realize the vision of smart PINS and achieve the desired multi-level best-effort traffic modes. A key question in this context revolves around expressing and measuring the general performance level of the BQ mode to ensure the protection of the basic Internet access service (RC 3).

5.3.6.3. Evidence of Potential

In order to assess the potential of the proposed multi-level best-effort approach, we have studied a simulation that covers the main solution elements outlined in the previous section and compare its performance characteristics to those of a traditional best-effort approach. The simulation scenario is illustrated in Figure 102. It includes a heterogeneous mix of application servers and clients corresponding to four traffic modes: OS-initiated file downloads (DL) in the Background (BG), user-initiated video streaming (VoD) with Basic Quality (BQ), live video streaming (LVD) with Improved Quality (IQ), and a highly delay-sensitive application emulating the exchange of haptic feedback (HAP) that requires Assured Quality (AQ). The applications differ not only in terms of their requirements, but also w.r.t. their elasticity and transport protocols. For instance, video applications use adaptive streaming over HTTP and can adjust to given network conditions, whereas DL and HAP applications rely on their respective TCP and UDP connections.

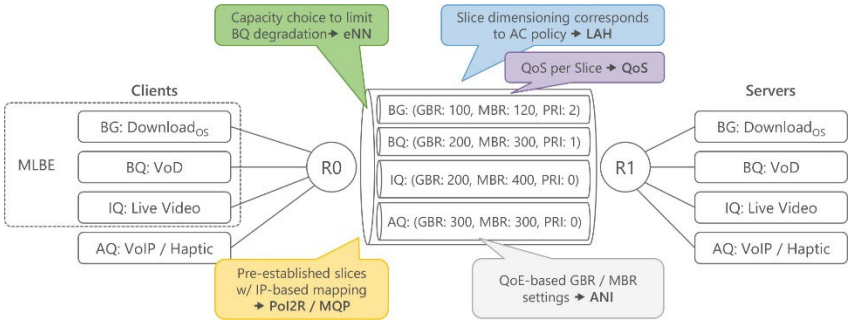


Figure 102 Simulation scenario for evaluating the proposed MLBE approach and coverage of corresponding key solution elements

The 1 Gbps link between servers and applications is sliced using a Hierarchical Token Bucket (HTB) scheduler, which enables us to include the following solution elements:

Per-slice Guaranteed and Maximum Bit Rate (GBR, MBR) settings and priorities allow for a QoS-based resource allocation. Since these settings are on coarse per-application and IP address range granularity, they can be provisioned in advance as per the PoI2R/MQP concepts.

Given the slice parameters, the static number of clients per application corresponds to an admission control policy that limits those numbers.

Slice parameters are chosen in a QoE-aware manner and are tailored to applications' requirements, hence covering aspects of ANI.

By carefully dimensioning the resources for the BQ mode, we ensure that the user-perceived application quality remains unchanged.

In particular, the GBR, MBR, and priority settings are chosen to represent the typical characteristics and requirements of the traffic modes: an isolated high-priority AQ mode as well as BG, BQ, and IQ modes with increasing levels of priority and potential for intra-slice capacity borrowing. For the following study, 110 active clients for each BG, BQ, and IQ modes and 825 clients of the AQ mode are

simulated. The latter number is higher to compensate for the lower capacity consumption per client.

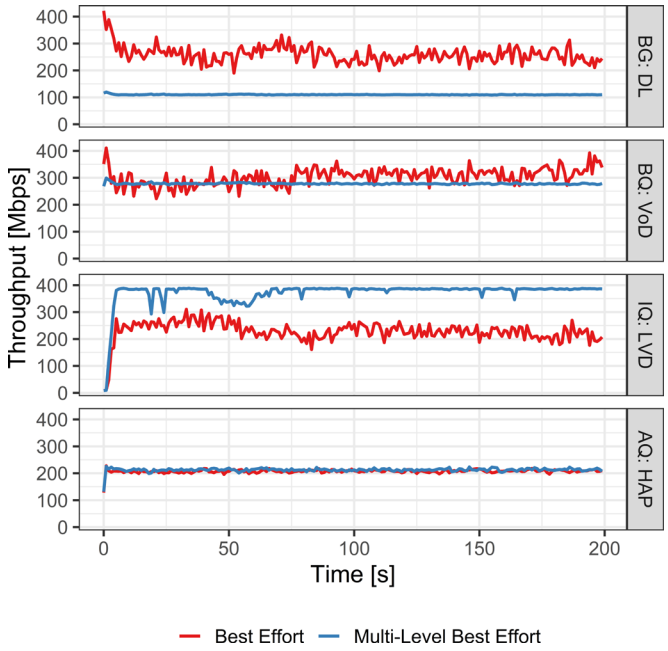


Figure 103 Time series of aggregated per-application throughput

The graphs in Figure 103 and Figure 104 show the aggregated per-application throughput and delay characteristics throughout the simulation. Each chart features two curves corresponding to the behavior in a Best-Effort (BE) regime without differentiation and the proposed MLBE approach. We make several observations on the throughput performance displayed in Figure 103. In there, highly fluctuating throughput values for the three TCP-based applications in the BE case are caused by the heterogeneous application mix and the clients' dynamic behavior, which prevents an equilibrium with a fully fair share. In contrast, the homogeneity of applications within each class in the MLBE case leads to significantly more stable throughput values constrained by the set GBR and MBR values. The UDP-based HAP application clients manage to send all their packets and achieve the same throughput in both the BE and MLBE conditions.

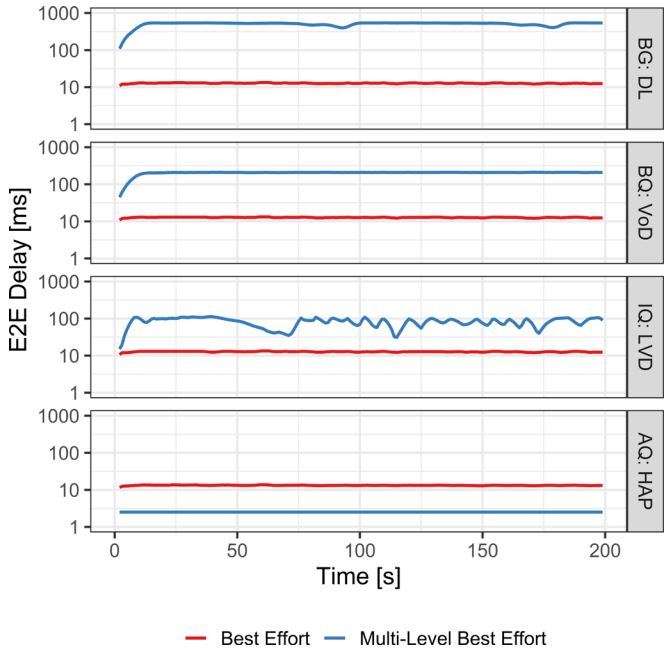


Figure 104 Time series of aggregated per-application delays

However, the delay performance reported at the bottom of Figure 104 highlights that using BE results in delays above 12 ms, outside the 3-10 ms range reported as a requirement for haptic applications in [ZHA18]. Without differentiation, all applications experience the same delay in the case of BE. When using MLBE, on the other hand, the delays of the HAP clients decrease to an average of 2.5 ms but come at the price of increased delays for all other applications. However, since these are more delay-tolerant, the user-perceived application quality is not negatively impacted in the BQ and IQ cases.

To quantify the QoE for video streaming applications on the 1-to-5 Mean Opinion Score (MOS) scale, we leverage the ITU-T P.1203 model. Our results show that while the MOS for the VoD application in the BQ mode remains unchanged at 4.18 in both the BE and MLBE cases, the IQ mode's MOS drastically improves from 2.83 to 4.38 when following the MLBE approach. This is mainly due to the increased link capacity allocated to the IQ traffic mode.

Note that although the delay experienced by the background download increases substantially and the lowered throughput causes longer download times, it maintains a stable bit rate throughout the experiment and is never starved. Since we assume this to be an OS-initiated background process, we do not expect this to have a noticeable effect on user experience. In summary, we have illustrated the potential of the combined MLBE and AQ approach to support emerging highly demanding applications and selectively improve existing applications' performance while limiting the impact on user-perceived application quality of background and basic traffic modes in a way that preserves net neutrality.

More details regarding the mechanisms and evaluation methodology can be found in [BOS2021, LON2022].

5.3.7. Path Computation within the Green Economy

This section shall be included as part of research on T4.3 topics. It provides interesting results, and as D4.2 was delivered, we have considered to include in this part. Next-generation telecommunications systems are envisioned to be sustainable and human-centric [Aho22]. Regarding customer

involvement, existing business models are currently limited to performance-focused Service Level Agreements (SLAs). It was pointed out in [Hos22] how it is natural for end-users to want the maximum Quality of Experience (QoE) possible, at the same time illustrating the potentially significant energy savings by passing to "good enough" QoE. Low (or lack of) awareness of end-users on digital services' energy usage implications/carbon footprint has been found in [Gna21], along with some indications of willingness to compromise. We have explored the said "willingness" and incentivization (for favouring energy efficiency at the cost of performance) to motivate environmental-friendly digital habits.

In more detail, the Decarbonization Level Agreements (DLAs) concept is introduced to include **green intents** in the service ordering process and specifically applied to path computation for inter-domain connectivity services. Supposing that, (transport) network devices have already started to embrace the state-of-the-art energy-aware features and mechanisms (e.g., Low Power Idle (LPI) [Bol11], Adaptive Rate (AR) [Les10] and vendor-specific mechanisms like [Ibm13]), **green states** are defined to represent the energy efficiency and performance trade-off in inter-domain links. The abstracted inter-domain topology can then evolve with information on each hop's available green states.

Figure 105 illustrates a sample scenario from the perspective of inter-domain connectivity services. The inter-domain links between **Domain A** and **Domain B/C** are supposed to support a *dynamic* energy-aware mechanism that reconfigures the network devices along the path according to the utilization dynamics, maximizing energy savings. At the time of a service order, the available green states on the two links differ mainly due to the observed utilization. On the other hand, it is also plausible that other inter-domain links support different energy-aware functionalities/mechanisms or not at all, as in the inter-domain link between **Domain B** and **Domain D**.

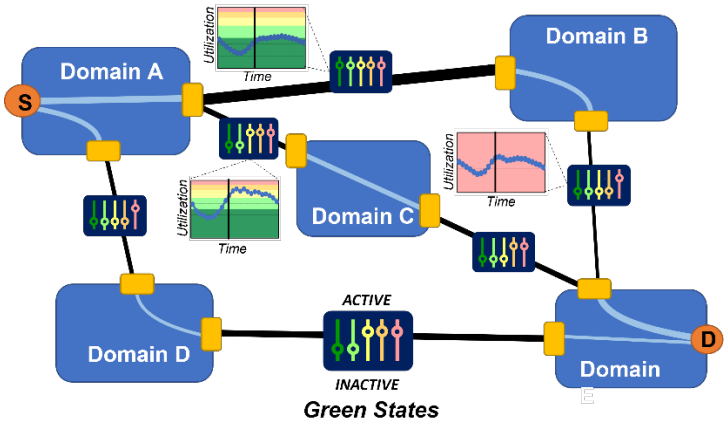


Figure 105. Differing green states on inter-domain links

Green path computation (gPC) policies are proposed with a reward system that allows greener states to correspond to higher rewards for lower performance. Considering that some green states might be unavailable (due to the trade-off mentioned above) for a given SLA requirement and link utilization at certain time intervals, introducing DLAs enables the customer to unlock greener states by allowing a certain level of performance degradation.

5.3.7.1. gPC for Inter-domain Connectivity Services

Let {**Gx**, **x**=1, 2, 3, 4} be the set of green states available on inter-domain links enabled by functionalities such as LPI and AR, among others. **G0** corresponds to the "business as usual" (BAU)

case, prioritising performance rather than energy efficiency. Considering the trade-off between energy efficiency and performance, the succeeding states **G1** through **G4** have increased energy efficiency at the cost of performance.

At a working state, mechanisms like AR can adapt the packet processing rates according to the traffic demands. On the other hand, mechanisms like LPI look into the burstiness of the traffic and can enter different sleeping states with increasing wake latencies and energy savings. Numerous works in the literature also try to combine these approaches towards driving energy-aware (network) device/link re-configurations according to traffic dynamics [Bol11], [Les10], [Ibm13].

Energy savings at the infrastructure layer result in reduced OPEX. This can then be escalated bottom-up through incentives. For instance, a vertical customer can earn Green Ambassador (GA) badges corresponding to increasing rewards as the allowable performance degradation increases. Within the green economy, green intents could come either as the desired GA level or for allowable performance degradation. Figure 106 compares the BAU, SLA-driven and DLA-driven inter-domain path computation. As previously mentioned, the BAU policy maximizes the performance, staying at the green state **G0**. The basic green intent is gPC_{SLA}, which corresponds to GA level 0 and maximizes energy saving while meeting the SLA. Going further, gPC_{DLA} maximizes energy saving for an allowable level of performance degradation.

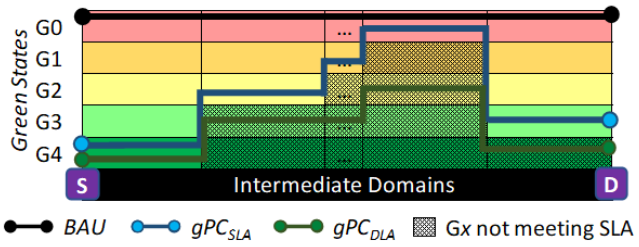


Figure 106. Green inter-domain path computation versus BAU

5.3.7.2. Power-Performance Modelling and Evaluation

The energy-aware inter-domain link is modeled as an M/M/1/SET queue, derived from the general model in [Bol20]. This considers the setup times, for instance, when waking up from a sleep state.

Considering renewal theory principles, the average power consumption of the link Φ is derived as the sum of the average consumption in the busy, wake-up and idle periods:

$$\Phi = \frac{\Phi_B(T_B + T_W) + \Phi_I T_I}{T_B + T_W + T_I}$$

Supposing that the former two consume Φ_B, while the latter consume Φ_I. For a given link speed μ and load λ, obtaining $T_I = 1/\lambda$ and $T_B = 1/(\mu - \lambda)$. The parameter γ is defined for evaluating AR, resulting in the effective link speed $\gamma\mu$. The wake-up time is supposed to be fixed as $T_W = \tau$.

On the other hand, the one-way delay $W$is derived as the sum of the average waiting time in the queue (W_q) and the propagation delay (W_p). The former is given as,

$$W_q = \frac{\rho}{\mu(1-\rho)} + \frac{2\tau + \lambda\tau^2}{2(1-\lambda\tau)}$$

where $\rho = \frac{\lambda}{\mu} < 1$ is the link utilization, while the latter is given as, $W_p = \Delta * 0.000005$, where Δ is the distance covered by the link. The reference value (i.e., 5 us per kilometer) for optical fiber in [PRD] is considered.

The power and performance models have been evaluated, and the power savings and delay increase are computed concerning the BAU case (i.e., links neither go to sleep during idle periods nor adopt lower rates during busy periods to save power). $\Phi_B = \mu * 10$ Watts/Gbps is derived from [Van12] for core routers. If LPI is disabled, the link does not go to sleep during idle periods, $\Phi_I = \Phi_i$, where $\Phi_i = 0.6 * \Phi_B$; otherwise, $\Phi_I = \Phi_s$, where $\Phi_s = 0.2 * \Phi_i$.

Figure 107a shows the impact of LPI and AR on power consumption. LPI generates high savings for scenarios with low link utilization, while AR improves the savings for scenarios with high link utilization. Figure 107b shows the impact of LPI and AR on delay. With the well-known trade-off between power and delay, it can be observed that the delay increase is higher when utilization is low. Moreover, increasing μ has direct effects on performance degradation and AR further worsens the degradation.

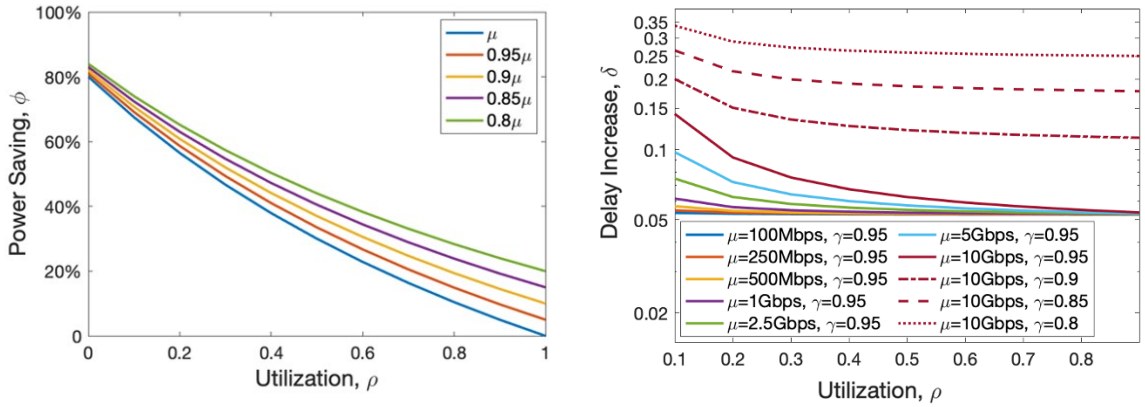


Figure 107. Impact of LPI and AR on: (a) power consumption, and (b) delay.

5.3.7.3. gPC Evaluation

The GEANT network topology dataset [ZOO] has been pre-processed such that missing data on link speed is set with 100Mbps (i.e., lower than the minimum value in the dataset, 155Mbps). In contrast, those with missing geographical locations (i.e., three nodes) are found to be on the edge of the graph, and hence, can be pruned. Figure 108 shows the graphical representation of the considered topology, with 37 nodes (corresponding to countries) and 58 links. Given the latitude and longitude data on the remaining nodes, the distances covered by the links are computed, which become the bases for W_p.

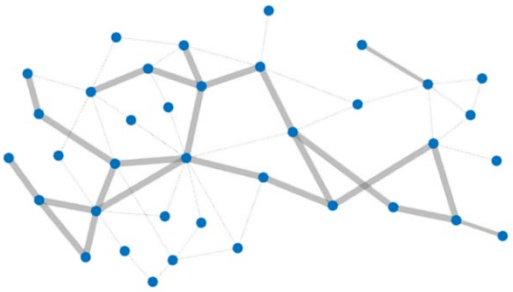


Figure 108. Graph representation of the GEANT-based topology.

On the other hand, the open telecommunications dataset from Telecom Italia [Bar15] is used as basis for the utilization variations on the inter-domain links. The data from the 100x100 grid has been aggregated into 100 10x10 grids to emulate traffic aggregation in domains. The aggregate values have been normalized, then considered as the link utilization dynamics for a given inter-domain hop in the path. For this evaluation, 58 out of the 100 utilization dynamics have been randomly selected to correspond to links of the topology. Moreover, 30 days' worth of data with hourly granularity have been used for the following results. The hourly load for each link is obtained as, $\lambda = \rho\mu$, while the wake-up time has been fixed to τ =30 us.

Four gPC policies are evaluated:

- **gPC$_{SLA}$** selects the greenest state among the ones available for the given SLA, with corresponding rewards, $R_{d0}[x]$, with d0 indicating that the SLA must be fulfilled;
- **gPC$_{DLA-d1}$** selects the greenest state among the ones available, also unlocking new states for an allowable performance degradation, d1, with corresponding rewards $R_{d1}[x]$;
- **gPC$_{DLA-d2}$** selects the greenest state among the ones available, also unlocking new states for an allowable performance degradation, d2, with corresponding rewards, $R_{d2}[x]$; and
- **gPC$_{DLA-d3}$** selects the greenest state among the ones available, also unlocking new states for an allowable performance degradation, d3, with corresponding rewards, $R_{d3}[x]$;

where d1 < d2 < d3 and $R_{d0}[x] < R_{d1}[x] < R_{d2}[x] < R_{d3}[x]$. The allowable performance degradation, d1=2%, d2=5% and d3=10% have been considered, with the following rewards equation:

$$R_y[x] = (y+1)(2y+x)$$

for using the available green state Gx, or unlocking it via GA level y.

Now, suppose a scenario where a customer orders an inter-domain connectivity service between any source **S** and destination **D** in each hour in the 30-day period, and the inter-domain links in the topology either have: (i) *Random* power saving modes implemented, uniformly distributed between BAU, LPI and LPI+AR; or (ii) *LPI+AR* implemented in all. The SLA requirement is generated from a uniform distribution with 1x, 2x and 3x the current delay for the path.

For simplicity, but without loss of generality, the green states, {**Gx**, **x**=1, 2, 3, 4} have been mapped to corresponding power saving intervals: (i) **G1**, 0-20%; (ii) **G2**, 21-40%; (iii) **G3**, 41-60%; and (iv) **G4**, >60%. Note that at the time of the service order, the links can then have different green states available for a given gPC policy.

Figure 109a shows the average power savings in the 30-day period when Random or LPI+AR power-saving modes are implemented across the topology concerning the BAU case. It can be observed that **gPC$_{SLA}$** improves the power savings for the inter-domain path by an average of around 12% for the Random modes, and around 21% for the LPI+AR. With **gPC$_{DLA-d1}$**, the savings improved to around 18% and around 31%, respectively. By unlocking greener states with allowable degradation levels, the savings increased by around 47% with respect to **gPC$_{SLA}$**. Similarly, Figure 109b shows the average delay increase for the different gPC policies, which are negligible in the order of 10^{-5} and 10^{-4}. In this evaluation, no changes on the power savings and delay increase have been observed when allowing more degradation, d2 and d3, since no new green states were unlocked.

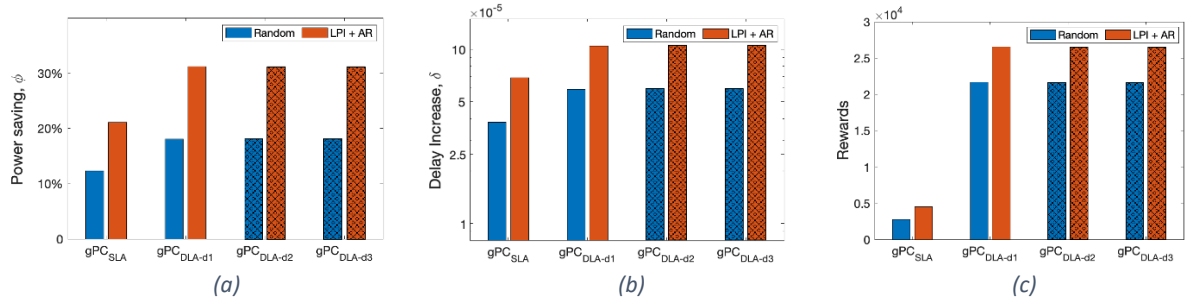


Figure 109. gPC policies comparson in terms of: (a) power saving, (b) delay increase, and (c) rewards.

Figure 109c illustrates the total rewards earned for the different gPC policies. Substantial rewards can be observed moving from **gPC$_{SLA}$** to **gPC$_{DLA-d1}$**, and for a negligible degradation. This could be a interesting way to drive customers towards earning GA badges and behavioural change.

5.3.8. Toolbox for scalability of Erlang microservices

TE component has been designed and developed in Erlang. In D3.2, we describe the component as-is, but in this section, we provide details on how the different tools in TeraFlowSDN have been used to provide scalability of Erlang microservices. These tools have also been reported as open-source software and detailed in D6.4.

TeraFlowSDN is equipped with a toolbox engineered for the scalability of Erlang-based microservices. The primary function of this toolbox is to facilitate the deployment, monitoring, and management of Erlang microservices, structured as Docker images, along with ensuring secure inter-service connectivity.

The toolbox comprises four primary components: *braid*, *braidnet*, *braidcert*, and *braidnode*. *Braid* serves as the client interface, designed to allow seamless interaction with the rest of the system. *Braidnet* is installed on each physical machine, acting as the controlling agent for the containers. *Braidcert* manages the Public Key Infrastructure (PKI), generating certificate and for the private keys of the Erlang nodes to connect to each other with TLS. *Braidnode*, on the other hand, functions inside each container as an interface between the orchestrator and between the microservices.

Unlike conventional tools, this unique toolbox extends beyond merely deploying, managing, and monitoring microservices. It strategically exploits Erlang's inherent distributed communication model to foster a highly interconnected, efficient, and dynamic microservices ecosystem.

Moreover, a distinct emphasis has been placed on security. This emphasis enables the toolbox to handle containers from numerous customers without compromising on privacy or security, thereby augmenting the resilience and versatility of the toolbox and making it a suitable choice for multi-tenant cloud environments.

The subsequent section provides an in-depth examination of the architectural design of this toolset, detailing the guiding principles, integral technologies, and critical architectural decisions that constitute the backbone of the toolbox.

The following section focuses on the specificities of the toolbox's implementation. Here, we delve into how these architectural principles have been translated into tangible solutions.

Finally, we present a comprehensive analysis of the rigorous testing and validation procedures to ascertain the toolbox's robustness and effectiveness. This section elaborates on the methodologies

employed, the results obtained, and the toolbox's overall performance, thus affirming the toolset's reliability and scalability under real-world conditions.

5.3.8.1. Architecture

Figure 110 illustrates a typical deployment of two Erlang microservices (Cluster A and B) over four host servers. It shows how the Erlang virtual machines are distributed on the different hosts, and how they communicate with each other's.

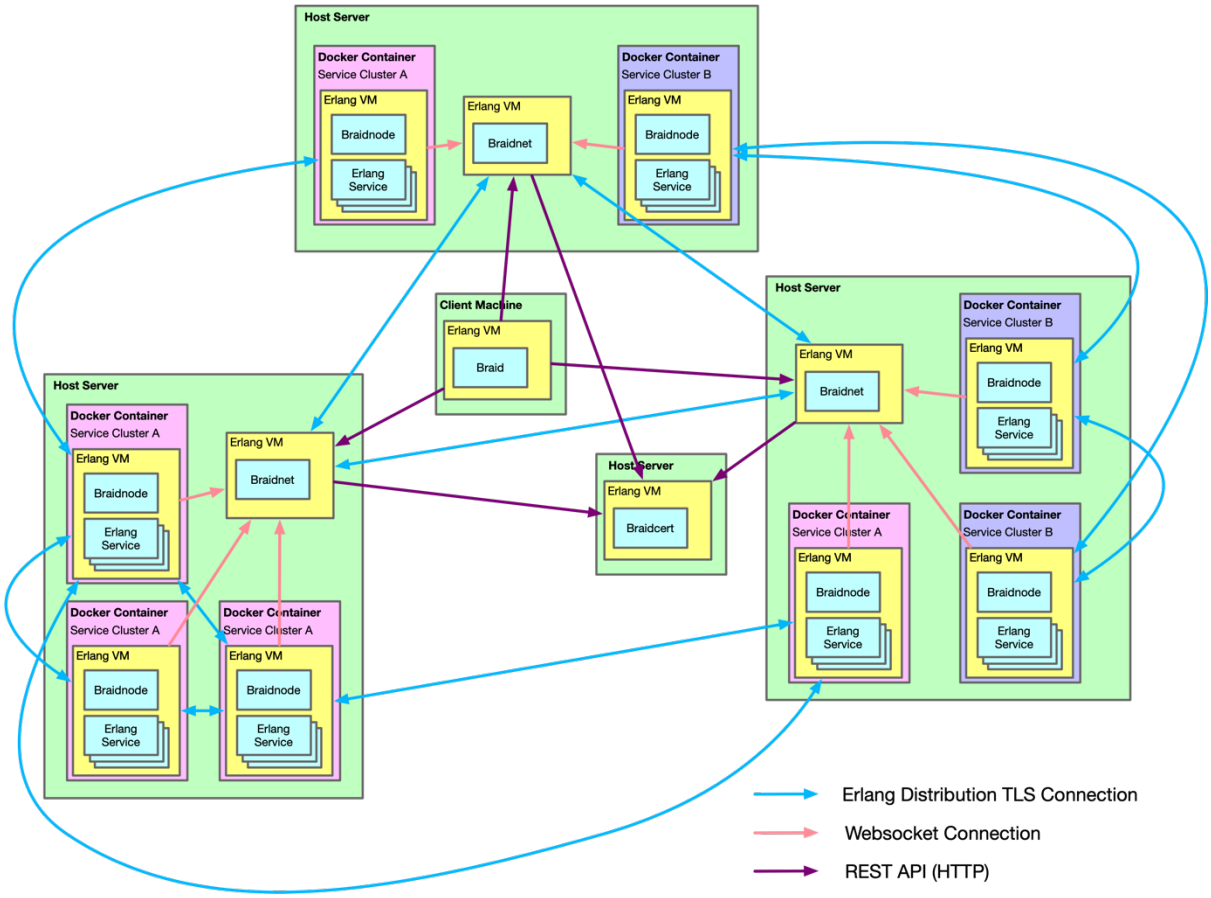


Figure 110. Example of deployment of multiple sets of services with braid toolkit.

Braid: The client Interface

The braid application serves as the client-side segment of this expansive toolbox, designed to streamline the management of diverse services. It operates as an intermediary between the system administrator and the services that form the backbone of the ecosystem, ensuring efficient deployment and seamless management.

At its core, *braid* interacts with services through the REST API provided by *braidnet*. This API serves as the communication channel between *braid* and the various services it is responsible for, handling all requests and responses. Using Representational State Transfer (REST) as the underlying protocol ensures that *braidnet* is both simple and scalable, accommodating an increasing number of services as the system expands. The statelessness of REST also enhances the reliability and efficiency of the *braid* system, as it eliminates the need to store and manage session information, thus simplifying the client-server interaction.

Braid's architectural design provides two primary modes of operation: as a client library application and as an executable script (escript) that runs as a command-line interface (CLI) tool.

When used as a client library, *braid* supports direct integration into other applications. This integration can enable these applications to programmatically utilize *braid*'s functionalities, allowing them to deploy, manage, and communicate with services within the system. This integration mode adds versatility and can greatly extend the toolbox's utility, making *braid* a powerful component within a larger system.

In contrast, when *braid* is exported as an escript and run as a CLI tool, it operates independently, directly interfacing with the system administrator. This allows the administrator to execute commands and operations directly, such as deploying new services, managing existing services, or querying the system's state. The CLI tool mode offers a straightforward and highly interactive way of managing services, which is especially beneficial in environments that require direct control and rapid feedback.

By providing these two modes of operation, *braid* demonstrates architectural flexibility that allows it to adapt to a wide array of use cases and deployment scenarios, such as the presented in [NFVSDN21], all while leveraging the power of the *braidnet* REST API to ensure efficient, scalable, and reliable operation.

Braidnet: The Orchestrator

Braidnet functions as the central orchestrator within the toolbox ecosystem. As a service running on each host, it manages local services, which are implemented as Docker containers running the Erlang-based *braidnode* application. These Docker images are signed by the client, generated with the help of the rebar3_docker plugin, adding an extra layer of trust and validation to the system. The system leverages Erlang's innate distributed nature, creating an environment characterized by both scalability and fault tolerance.

Braidnet takes advantage of the Cowboy framework [COW] to offer a REST API for braid clients. This API provides a streamlined interface for clients to interact with *braidnet*, enabling them to perform operations and access relevant information. By following REST principles, *braidnet* establishes clear endpoints and supports standard HTTP methods. The REST API, implemented with Cowboy, simplifies integration and facilitates seamless communication within the *braidnet* ecosystem.

To effectively control and monitor the *braidnode* instance, *braidnet* establishes a WebSocket connection with it. This real-time communication channel allows *braidnet* to handle the lifecycle of each Docker container, from its initialization to eventual termination. Through its role as an orchestrator, *braidnet* can start, stop, and monitor services, ensuring optimal system operations.

Braidnet also manages service discovery using its Erlang Port Mapper Daemon (EPMD) implementation, which keeps track of the nodes available in the network and the corresponding ports for communication. The discovery service provides a crucial functionality that ensures segregation between the services of different tenants within the ecosystem by maintaining separate namespaces.

This tenant segregation allows for a clear isolation of services belonging to different tenants, preventing unauthorized access and ensuring secure communication within each tenant's environment. *Braidnet* effectively keeps track of the nodes and their corresponding ports specific to each tenant, enabling the orchestration and interaction of services within their respective boundaries.

The *braidnet* architecture places a significant emphasis on ensuring secure communication between services. One of the main features it provides is the delegation of TLS certificate and private key handling to the *braidnet* orchestrator, as shown in Figure 111. This setup ensures that microservices can establish secure, authenticated connections with each other without storing any secrets within the Docker containers themselves, effectively minimizing the risk of secret leakage.

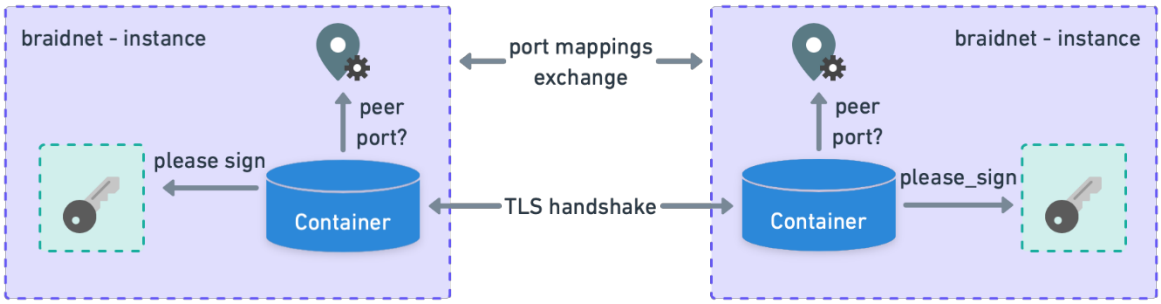


Figure 111. The delegation of security and discovery functions to braidnet.

Braidnode: The Service API

The *braidnode* component serves a critical role within the toolbox, furnishing microservices with a robust, secure, and authenticated pathway to communicate with other microservices. It achieves this through a sophisticated integration with the Transport Layer Security (TLS) stack and a series of delegated functions to the *braidnet* orchestrator.

One of the standout features of *braidnode* is its ability to hook directly into the TLS stack. By doing so, *braidnode* facilitates secure and private communications, essential in a microservices architecture where numerous services interact frequently. A key element of this setup is delegating the private key and certificate validation processes to *braidnet*.

Rather than storing and handling these security credentials within the microservice or the Docker container, this information is managed exclusively by *braidnet*. As a result, the microservices can establish secure, authenticated connections with each other without having to store sensitive information themselves, effectively minimizing the risk of secret leakage.

Braidnode leverages WebSocket, a real-time communication protocol, to maintain an open line of communication with *braidnet*. This persistent connection allows *braidnode* to relay logs and status updates continuously to the *braidnet* orchestrator, ensuring that the state of each microservices is continually monitored and managed.

Additionally, this WebSocket connection also enables *braidnet* to send commands directly to the *braidnode* instance, manipulating the lifecycle of the microservice it controls. Whether it's an instruction to start, stop, or restart the service, this direct line of command plays an integral part in maintaining a responsive and adaptable microservices ecosystem.

In a usual distributed Erlang system, node resolution would rely on the Erlang Port Mapper Daemon (EPMD). However, within our architecture, *braidnode* delegates this responsibility to *braidnet*.

By doing so, *braidnode* ensures that it can reliably discover other microservices, even in a highly dynamic and distributed environment. This strategy also contributes to *braidnet*'s consolidation as the central orchestrator, responsible for maintaining the overall system's organization and connectivity.

In summary, *braidnode* provides a robust framework for implementing microservices within the toolbox. By integrating with the TLS stack, leveraging WebSocket communication, and delegating critical functionalities to *braidnet*, *braidnode* framework ensures that microservices can operate securely and interact seamlessly and be managed effectively in a highly distributed environment.

Braidcert: Securing the Infrastructure

Braidcert is a dedicated component of the toolbox that shoulders the responsibility of Public Key Infrastructure (PKI) management. It serves as the foundational pillar for creating secure, authenticated connections across a network of microservices. As its functions involve high-level security, *braidcert* is implemented as a separate service, further strengthening its security measures and isolating it from potential threats in the wider orchestration layer.

Braidcert's primary function is to receive certificate signing requests from peers to generate certificates. These certificates serve as digital identities, allowing microservices to authenticate each other and securely exchange information over the network. In this process, the orchestrator generates the private key and sends a signing request to *Braidcert* in order to obtain the proper certificate. In a multi-tenant environment, this functionality plays a critical role by enabling access control at the microservice level, preventing unauthorized access, and maintaining tenant isolation.

Through its effective management of the PKI, *braidcert* not only ensures secure connections but also verifies the identities of the services communicating with each other. This process guarantees that the services involved in any communication are indeed the ones they claim to be, enhancing the overall security of the infrastructure.

Considering the significance of its role, *braidcert* is implemented as a separate service to ensure optimal security. By separating *braidcert* from other orchestrators, it is better shielded from potential threats that could compromise the system's security. This design choice aligns with the principle of least privilege, allowing each component to have only the necessary permissions and access, reducing the potential attack surface.

Furthermore, every *braidnet* instance must possess a certificate issued by the infrastructure's PKI. This requirement is an additional security measure that ensures only authorized entities can access the sensitive cryptographic material *braidcert* manages.

Braidcert provides a REST API, which is used by *braidnet* to securely retrieve bundles of certificates and generate new ones for authorized clients. This interface ensures an efficient, secure communication channel for obtaining the necessary credentials without exposing them to undue risks. This design minimizes the attack surface and ensures that *braidcert* remains the single, trusted source of all cryptographic materials.

In summary, *braidcert* is the fortress of the toolbox, securing the infrastructure by managing the PKI and facilitating the enforcement of authenticated and secure communications. Its architectural design, underpinned by strong security principles and efficient interaction with *braidnet*, ensures that *braidcert* upholds the trust and integrity in the communication between microservices, ultimately contributing to the overall reliability and security of the entire system.

5.3.8.2. Implementation Details

This section provides a detailed exploration of the braid toolbox internals. The discussion will focus on its specific configuration setup, microservices lifecycle management, service discovery protocol, and strategies for secure inter-service communication. Accompanying sequence diagrams provide a detailed visual narrative of these processes, enhancing understanding and highlighting the interaction between different system components.

Erlang

Erlang was chosen as the primary language for designing the orchestration toolbox based on several unique qualities that facilitate the development of robust, distributed, and concurrent systems. These advantages are particularly well-aligned with the requirements of microservice orchestration.

Erlang's built-in support for concurrency and distribution, combined with its unique 'let it crash' philosophy, provides an excellent framework for managing and orchestrating microservices. It allows for the handling of multiple microservices concurrently without a significant impact on system performance. If one service encounters an error, it crashes and restarts without impacting the other running services, thus ensuring system stability.

For the orchestration toolbox, the decision to use Erlang is further justified when the microservices themselves are written in Erlang and run in Docker containers. Using the same language for both orchestration and service implementation removes language barriers, simplifies debugging, and allows for greater code reuse. It enhances the development experience and encourages a cohesive approach to system design.

While the current architecture primarily leverages Erlang for both orchestration and the implementation of microservices, this does not preclude future adoption of other languages. Indeed, the microservices architecture by its very nature allows for each service to be implemented in the language best suited to its specific requirements.

In future iterations of the toolbox, it may be beneficial to implement certain microservices in languages other than Erlang if the need arises, perhaps due to the service's unique requirements, the need for certain language-specific libraries, or team expertise.

However, in such scenarios, careful consideration will need to be given to the communication and integration between the services, especially in terms of data serialization and deserialization. Despite this, the inherent flexibility of the microservices architecture would allow for this evolution without significant impact on the overall system design.

Braidnet Supervision Tree

Braidnet's supervision tree shown in Figure 112 embodies its dual responsibilities. On one side, there is Cowboy, which exposes the REST API to braid clients. This component is crucial in providing an interface for external clients to interact with the system and perform various operations.

On the other side, the supervision tree houses the EPMD server, which plays a pivotal role in distributed Erlang systems by mapping node names to TCP/IP ports. This component is essential for managing inter-node communication within the system.

The orchestrator worker, a process responsible for managing the lifecycle of microservices, also resides within this part of the tree. It communicates with the Docker containers running the microservices and performs actions like starting, stopping, or restarting them based on the commands it receives.

Finally, the pool of container managers oversees the management of the Docker containers hosting the microservices. Each manager is responsible for a subset of the containers, ensuring that the system can efficiently scale to manage many services.

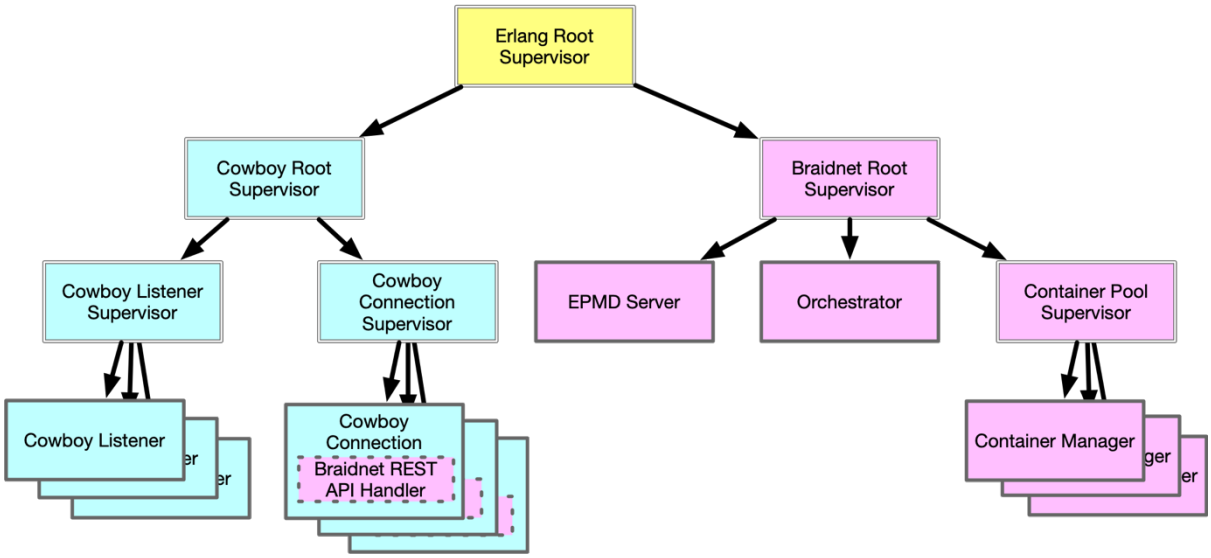


Figure 112. Braidnet supervision tree.

Braid Configuration

When configuring a microservice deployment via the *braid* toolbox, *braid* serves as the first-stage decision-making entity, selecting the appropriate *braidnet* instances where the microservices will be hosted. This deployment configuration, detailed in a map, is then provided to the chosen *braidnet* instances.

A glimpse into a typical configuration as a map (associative array) might appear like:

```
#{
    'braidnet-instance-id-0' =>
        #{
            alice => #{
                image => <<"stritzinger/braidnode:0.1.0">>,
                epmd_port => <<"43591">>,
                connections => ['bob@braidnet-instance-id-1']
            }
        },
    'braidnet-instance-id-1' =>
        #{
            bob => #{
                image => <<"stritzinger/braidnode:0.1.0">>,
                epmd_port => <<"43591">>,
                connections => ['alice@braidnet-instance-id-0']
            }
        }
}.
```

Braid's selection process ensures optimal resource allocation and load distribution across all *braidnet* instances. Upon receiving the configuration, each *braidnet* instance uses this information to manage the lifecycle of the microservices specified therein. This dynamic allocation process enables the system to balance the load and effectively utilize resources, thereby enhancing overall performance and scalability.

Lifecycle Management of Microservices

The sequence diagram Figure 113 depicts the process *braidnet* uses to manage the lifecycle of microservices. It demonstrates how *braidnet* interacts with the *braid* client and the *braidnode* service,

interprets the given configuration, and uses that information to initialize, monitor, and eventually terminate services.

Braidnet initiates a Docker container for each microservice on the appropriate server based on the service constraints specified in the configuration. After initializing the container, *braidnet* maintains a WebSocket connection to the microservice, enabling real-time monitoring and control over the service's lifecycle.

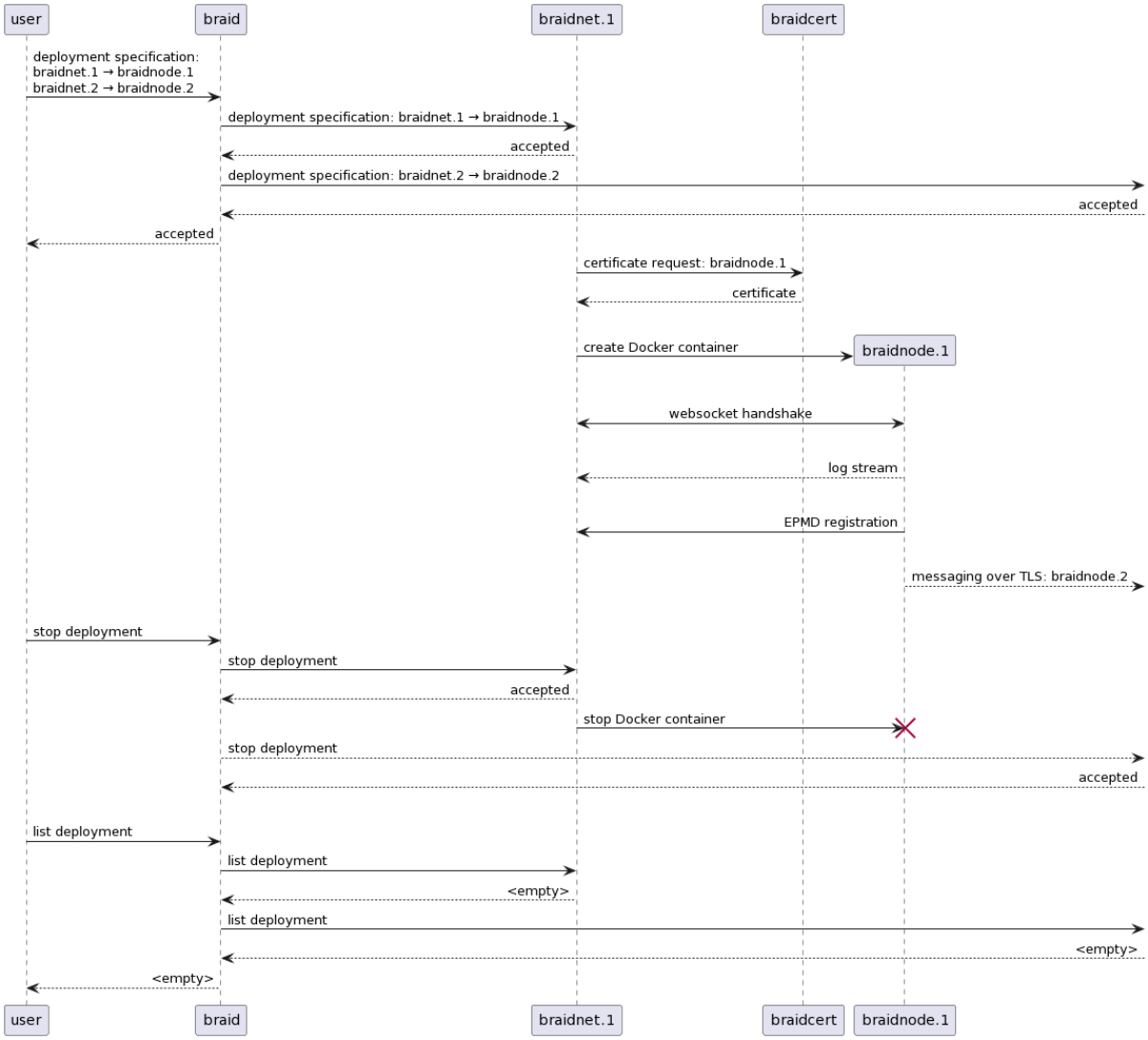


Figure 113. Sequence diagram of the microservices life cycle.

Service Discovery and Connection Details

The service discovery mechanism is illustrated in details as a sequence diagram in Figure 114. Instead of using the traditional Erlang EPMD within the microservice container, *braidnode* externalizes this functionality to *braidnet*.

Braidnet handles the process of discovery and connection establishment between services, using the information provided in the configuration. This approach streamlines updates and modifications to network configurations as changes are made in one central location and instantly propagated across the network.

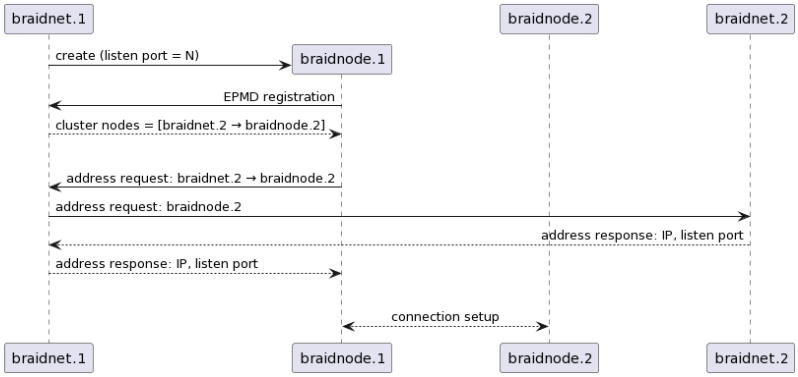


Figure 114. Sequence diagram of service discovery.

Securing Inter-Service Communication with TLS

The sequence diagram in Figure 115 illustrates how the *braid* toolbox ensures secure communication between microservices using TLS. *braidcert* is responsible for generating the certificates for each microservice.

Upon generating these cryptographic elements, *braidnet* retrieves them via the *braidcert* REST API. These credentials are then passed on to *braidnode*, which uses them to authenticate and secure connections with other microservices.

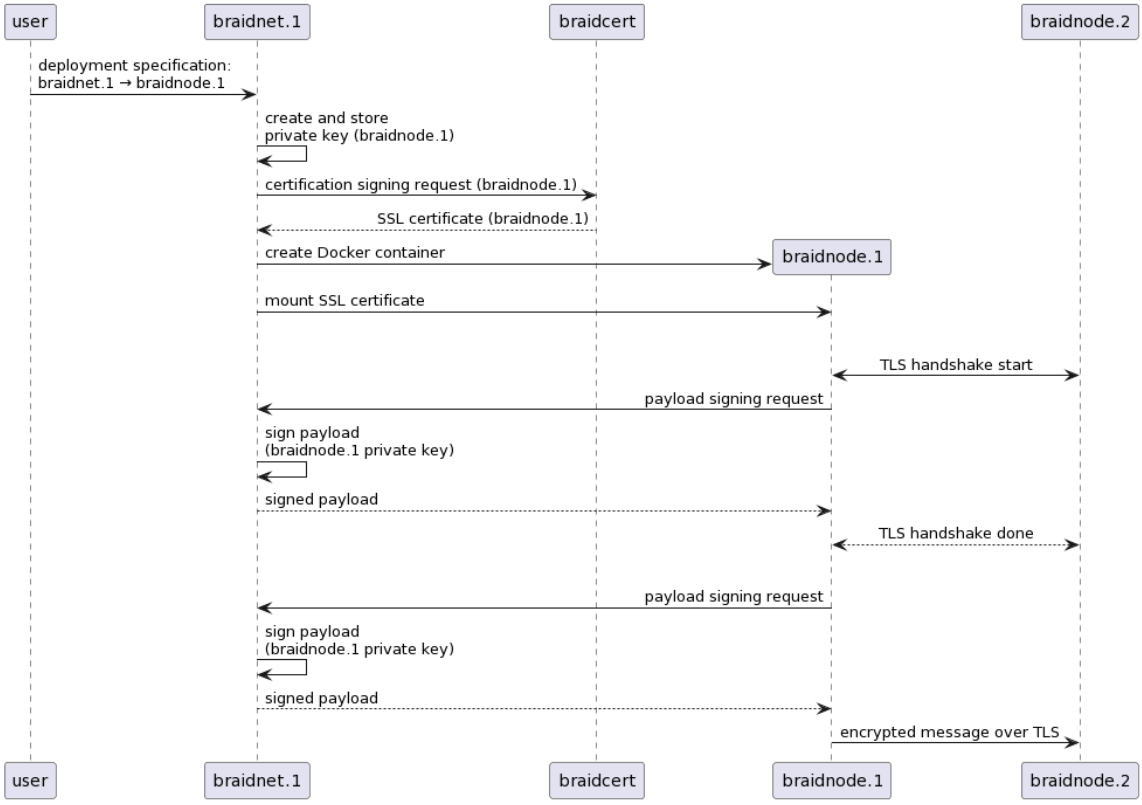


Figure 115. Sequence diagram of certificate management and secure connection.

5.3.8.3. Evaluation and Testing

Objectives

The objective of the validation procedure presented in this section is to assess the scalable nature of the *braid* toolbox. The procedure aims to demonstrate that the *braid* toolbox can reliably handle complex services running globally across multiple data centers. By conducting a series of tests and measurements, we seek to evaluate the performance and scalability of the toolbox in a real-life scenario.

Methodology

To ensure meaningful and accurate results, we have defined a comprehensive set of testing and validation measures. The tests focus on service startup time, messaging latency between services, and container recovery time. The tests performed involves starting multiple sets of services and examining their behavior in a manner that closely resembles real-life scenarios.

The set of services/containers are started one after the other to observe the impact on scalability. By carefully monitoring the concurrent services and their effect on performance, we can evaluate the scalability of the *braid* toolbox. Additionally, special attention is given to testing crash recovery and measuring the time required to resolve a crashing service.

Test Environment

For the testing environment, we have opted to use the fly.io platform. This platform allows us to deploy servers globally across multiple geographical regions, which closely mimics real-life scenarios. We have deployed 10 *braidnet* instances on 10 different geographical regions using the fly.io infrastructure. The servers used for testing are virtual machines with hardware specifications of 16 CPU cores and 32 GB of RAM each, as defined by the *performance-16x* virtual machine offering on *fly.io*. The containers are created using *rebar3_docker* and are based on a simple Erlang microservice.

Results

During our testing process, we collected three significant sets of measurements: startup latency, messaging latency, and crash recovery latency.

The startup time of a new set of services/containers is measured in relation to the number of concurrent sets already running. The results indicate the system's ability to handle different loads while maintaining acceptable startup times. The graph shown in Figure 116 visually represents the relationship between startup time and concurrent services, showcasing *braid*'s scalability.

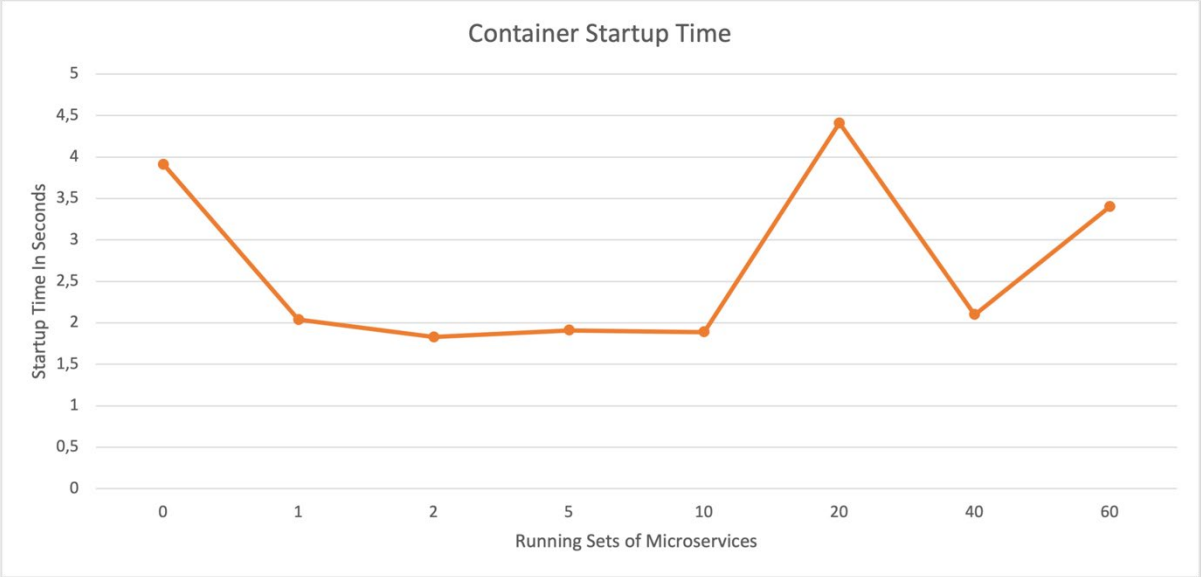


Figure 116. Container startup time in function of the number of concurrent sets of services.

The measurement of messaging latency between services is conducted as the number of concurrent sets of services increases. This measurement reflects how the system performs under increasing loads, particularly in terms of the latency of communication between services. The graph shown in Figure 117 illustrates the relationship between messaging latency and concurrent services, providing insights into *braid*'s scalability.

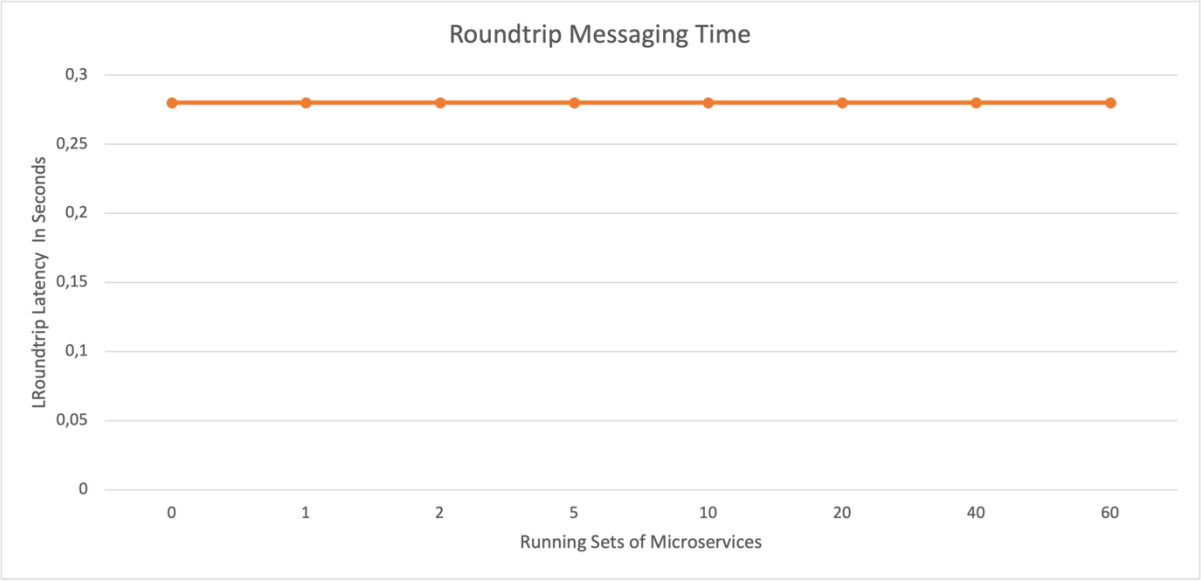


Figure 117. Message roundtrip time in function of the cumber of concurrent sets of services.

Container recovery time is evaluated by measuring the time required to resolve a crashing service as the number of concurrent sets of services containers increases. This measurement assesses the system's ability to recover from failures and maintain stability. The graph shown in Figure 118 showcases the relationship between container recovery time and concurrent services, demonstrating *braid*'s resilience.

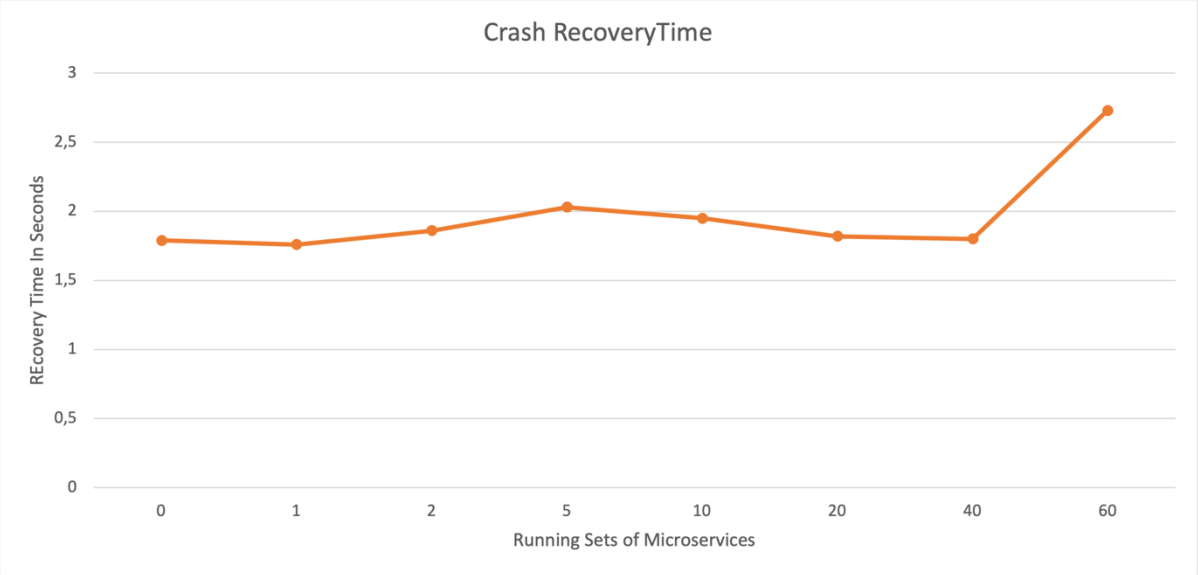


Figure 118. Container crash recovery time in function of the number of concurrent sets of services.

Conclusion

The result of the validation procedure show that the *braid* toolkit has the potential to ensure sufficient scalability. Despite room for improvement, the tests highlight *braid*'s ability to handle complex services running globally across multiple data centers. Erlang's inherent strengths are showcased in the results, reaffirming the choice of Erlang as the foundation for the *braid* toolbox. The validation report serves as evidence of *braid*'s performance and scalability, paving the way for further enhancements and optimizations.

5.4. Scenario conclusions

KPI	Target	Validation results
Multi-tenancy	> 100 tenants	TeraFlowSDN can support more >100 tenants with reasonable service latencies. At high loads, the response time might increase due to database latency and SQL query contention. Future releases might study how to enhancement the database schema.
Trust/privacy	100% secured connections	All connection related to the DLT component are secured and authenticated.
DLT transaction delay	10s	Average latency is between 2.2 and 3.3 seconds, c.f. Figure 75.
Positioning	100% vehicles	We have validated location-awareness in end-to-end connectivity services and in network topologies. TeraFlowSDN has been extended with an augmented data model for topology and connectivity services to include GPS coordinates and Regions into service endpoints, as well as connectivity service constraints. The proposed architecture considers the requested end-to-end connectivity service provisioning and update taking into consideration that location-aware connectivity services might need service endpoint

		migration due to the dynamic nature of joint edge-cloud continuum.
Social	< 20% cost	Green path computation (gPC) policies are proposed with a reward system, in which allowing greener states correspond to higher rewards for the lower performance. Considering that some green states might be unavailable (due to the aforementioned trade-off) for a given SLA requirement and link utilization at certain time intervals, the introduction of DLAs enables the customer to unlock greener states by allowing a certain level of performance degradation. By unlocking greener states with allowable degradation levels, the savings increased by around 47% with respect to **gPC$_{SLA}$**.

6. Scenario 3: Cybersecurity

This section introduces the third and last scenario explored within the TeraFlow project. This scenario addresses the cybersecurity aspects of transport network slices from two perspectives: packet-level communication or Layer 3 (IP / L3) and optical physical layer. The study emphasizes the important challenges in providing a holistic cybersecurity framework supported by SDN. Key aspects considered in this scenario include scalability, detection accuracy and reliability, and prompt response to detected attacks.

First, we introduce the scenario. Second, we present its alignment with the TeraFlowSDN architecture. Third, we present the performance evaluation work addressing L3 and the optical layer. Finally, we provide a summary of the scenario conclusions and future steps.

6.1. 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.

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. These solutions need to be presented as 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 119 depicts an example of the envisioned Cybersecurity scenario and of the threats in the context of an automated network. Attacks may target the IP and/or the optical layers at the data plane.

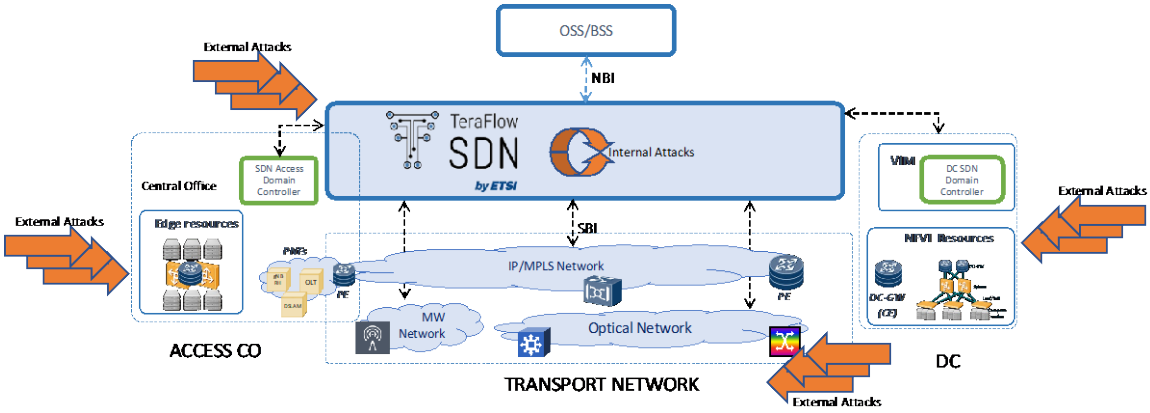


Figure 119. Scenario 3: Cybersecurity

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 in terms of the processing power needed to process packets. 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 exploiting the optical layer can target devices or fiber installations at any point in the network, which represents a large vulnerable surface for attacks. Taking advantage of current-generation transceivers, monitoring the security of each optical service in the network is possible. However, scalability becomes a problem due to the strict monitoring cycles (in terms of the maximum cycle processing time) necessary to monitor the optical services to detect attacks in a reasonable time. For instance, in a national network with 50 nodes, a single connection between each pair of nodes will account for 2,450 optical services to be monitored. Considering that an ideal monitoring cycle ranges from 30 seconds to a few minutes, it becomes crucial to provide a solution that can address such monitoring tasks in a scalable way.

6.2. Alignment with TeraFlowSDN architecture

We developed a Cybersecurity module to address the broad range of cybersecurity threats present in this scenario. The Cybersecurity scenario will validate several components and use cases. The Cybersecurity module within TeraFlowSDN comprises three components: *Centralized Attack Detector (CAD)*, *Attack Inference*, and *Attack Mitigator*. The main components involved in this scenario are highlighted in Figure 120. They are deployed in different containers to take advantage of cloud-native applications' scalability and reliability features. The Cybersecurity components integrate with TeraFlowSDN core components in several ways, as illustrated in Figure 110. The Service component is used for the service provisioning and (re)configuration tasks for attack mitigation. Integration with the Device component is also needed to perform changes to specific devices when mitigation actions are needed. The Context component detects service updates (i.e., creation and deletion) and retrieves service details. The Monitoring component retrieves monitoring data and stores the result of the security assessment process.

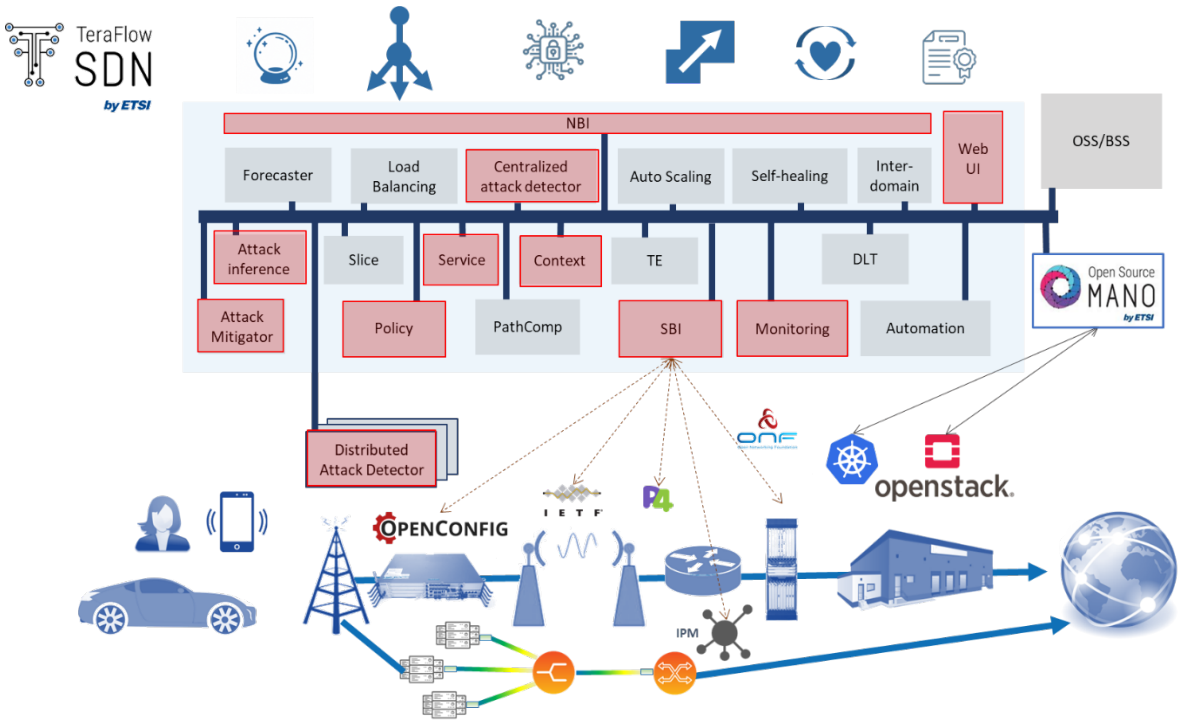


Figure 120. TeraFlow components used in the cybersecurity scenario

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, Section 3, 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, Section 2.

6.3. Performance Evaluation

Due to the broad nature of the attacks analyzed in this scenario, we report the performance evaluation of the L3 and optical cybersecurity modules in their own subsection. We start by discussing the performance evaluation of the L3 Cybersecurity module. Then, we evaluate the performance of the Optical Cybersecurity module. We finalize this section with a summary of the results, and future plans.

6.3.1. Layer 3 Cybersecurity

For L3 Cybersecurity, the main objective is to inspect packets as they traverse SDN-enabled L3 devices. This requires DAD instances to be located close to the L3 devices and receive a copy of the traffic traversing the L3 device. The following sub-sections present the testbed setup, workflows, and results.

6.3.1.1. MouseWorld Setup for Layer 3 Cybersecurity Experiments

Classical VPN services network operators provide are unaware of cybersecurity attacks, because such capability would require additional appliances or solutions to cope with attacks (i.e., Firewalls, Intrusion Detection Systems – IDS, etc.). The additional resources must be either located at the client facilities or reachable through traffic engineering (i.e., redirection to a cleaning center) on the network operator side. This has been considered a disadvantage if we compare it with Software-Defined Wide

Area Network (SD-WAN) [WAN19] or Secure Access Service Edge (SASE) [KAS20] overlay solutions. Those solutions provide their own devices and overlay network (agnostic to Multiprotocol Label Switching - MPLS) centralized in a cloud platform to optimize the delivery of network services over multiple locations and integrate network and security services that operates with their own devices close to endpoints. The MouseWorld 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 in the network without dedicated endpoint security devices. Cryptomining is a process of validating transactions on a decentralized cryptocurrency blockchain. The attack works by creating a botnet of devices, called miners, which are used to validate transactions and receive rewards in digital currencies, such as Ethereum (ETH) and Monero (XMR). Cryptomining attacks open the possibility of leveraging the computational resources provided by the network to the attacker's advantage when these resources are connected to the Internet. In the context of the massive adoption of cloud-native technologies and pervasive 5G connectivity, cryptomining malware-based attacks infecting the resources of those environments are strongly appealing.

The attacker can use devices already infected with malware or can infect new devices, hijacking their resources to create the botnet for mining. Attackers can use various ways to infect devices, such as spreading malicious links on social networks, using phishing attacks, or spreading malicious applications.

In addition, the attacker must choose the cryptocurrency to be mined and the mining pool they want to join to validate transactions. A mining pool is a service that allows miners to combine their resources to validate blocks of transactions and receive rewards in exchange. Once the attacker has collected all the pieces, they can set up the botnet to mine cryptocurrencies for the criminal’s benefit.

To detect cryptomining traffic in a timely and accurate manner, the network is the best place to identify these types of attacks [PAS20]. However, detecting cryptocurrency mining activity on the network can be challenging, as encryption methods have been adopted in most of today’s network protocols. For example, the attacker can use the Secure Socket Layer (SSL)/Transport Layer Security (TLS) encryption protocol to hide the cryptomining protocol in the payload of the encrypted communication. For this reason, classical techniques of Deep Packet Inspection (DPI) or identification of mining pool domain names (in the case of using encrypted Server Name Indication – SNI – or web proxies) are ineffective in detecting mining activity in today’s networks and more sophisticated techniques are needed to prepare today’s cybersecurity professionals to deal with these problems in real-life situations.

The setup considered for this experiment is illustrated in Figure 121. A Level 3 VPN service (L3VPN) is deployed using a TeraFlowSDN Controller instance in the Telefónica facilities (we refer for details to 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 SBI 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 (i.e., 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.

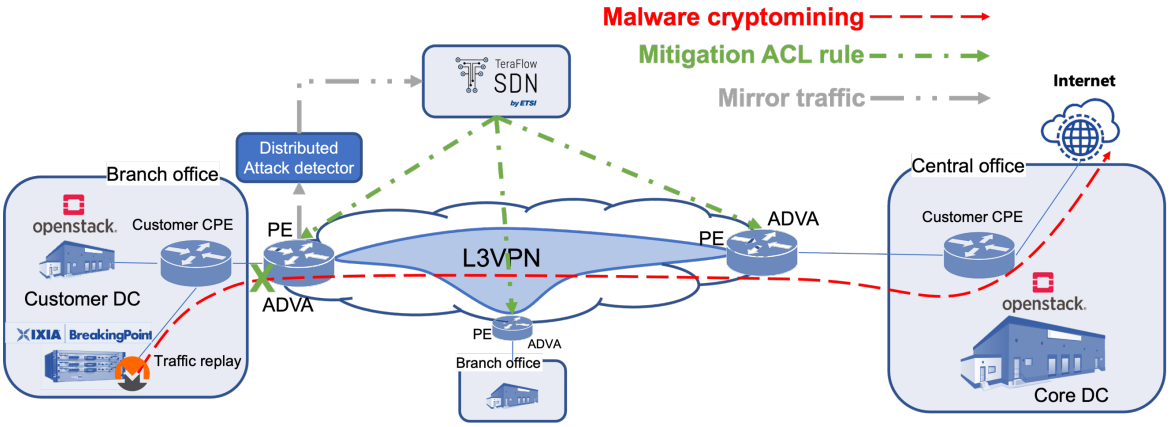


Figure 121. Deployment of the cybersecurity scenario focusing on L3

6.3.1.2. Workflows

In the case of Scenario 3, two complementary 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 illustrating how the Cybersecurity component interacts with other TeraFlowSDN core components. Later, the specifics of the Layer 3 and Optical workflows will be detailed.

Figure 122 shows the general communication among the core and cybersecurity components when a new service is created. 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 Section 5.2.4 of 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 a new service. Then, 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.

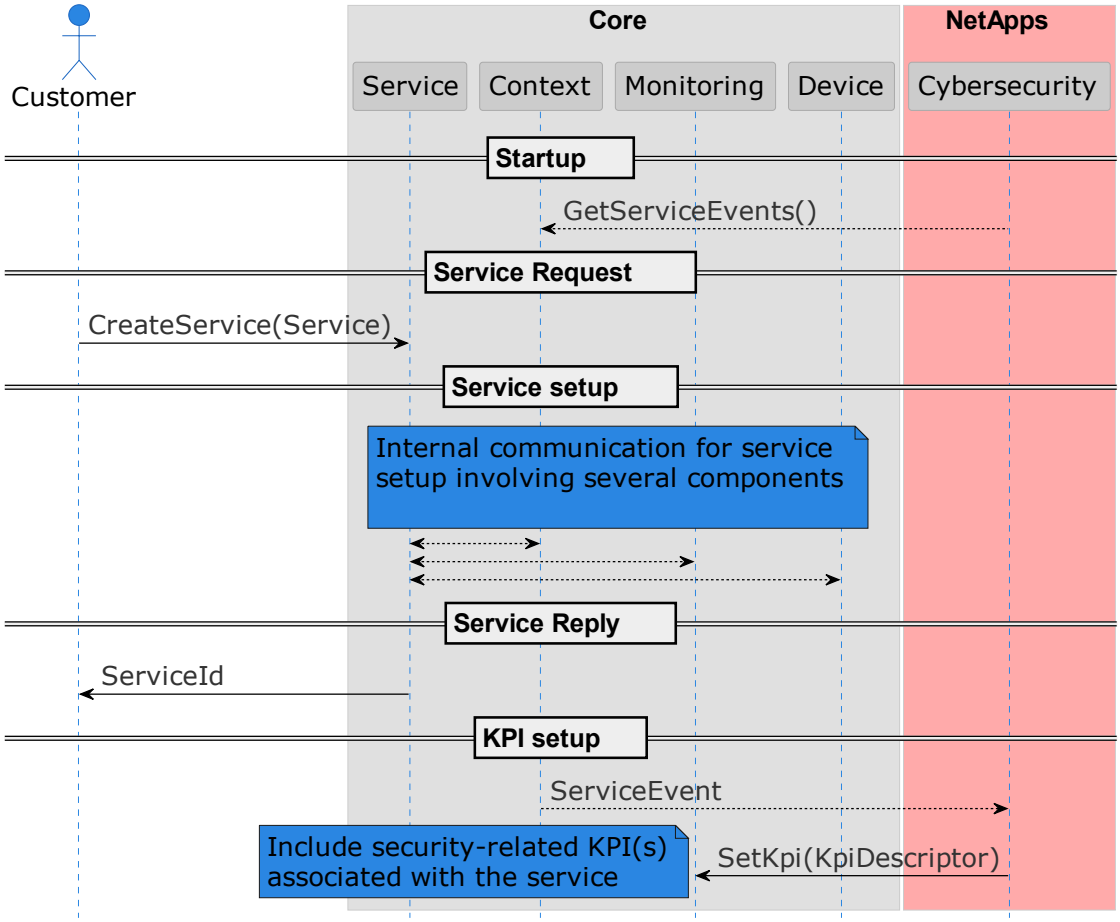


Figure 122. Scenario 3 workflow: General communication when creating a new service

We will now describe the specific workflows that implement the detection and mitigation of network attacks at the IP layer.

Traffic Capture and Feature Extraction at the Network Edge

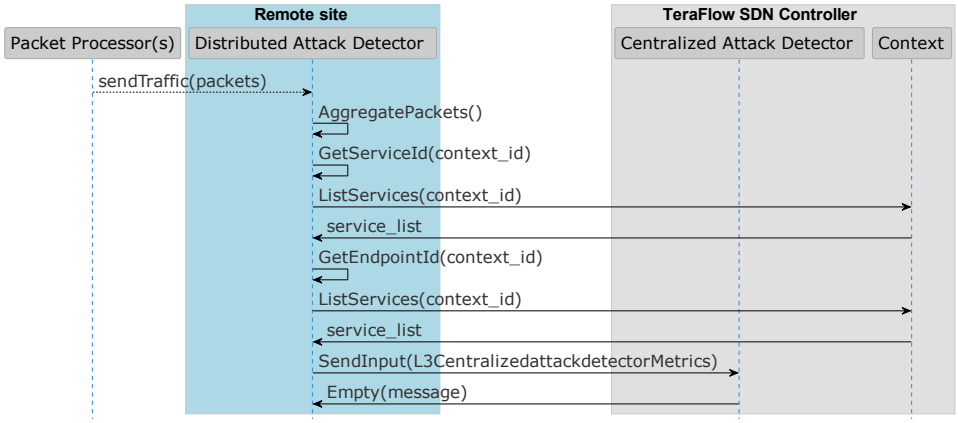


Figure 123. Scenario 3 workflow: Traffic Capture and Feature Extraction Workflow

In this scenario, 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 application (TCP STatistic and Analysis Tool). TSTAT allows real-time capture of packets, and their analysis. Figure 123 shows that the DAD communicates via gRCP 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 gRCP method *SendInput* to the CAD.

Detect Known Attacks using Supervised ML

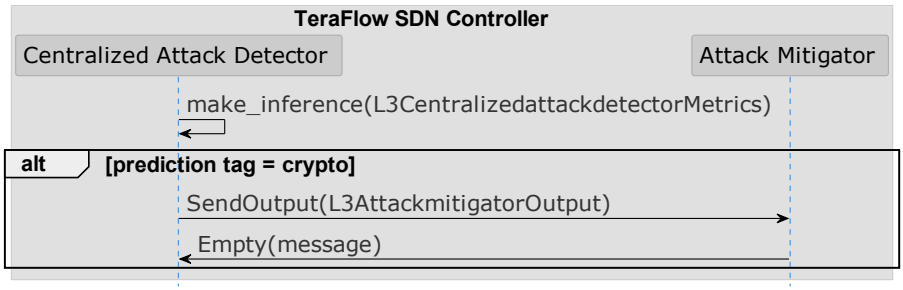


Figure 124. Scenario 3 workflow: Attack Detection Workflow (Layer 3)

Figure 124 shows the workflow for the detection of known attacks. The CAD component receives, and stores flow statistics from the *L3CentralizedattackdetectorMetrics* objects. A function is then called with these objects as the input to perform the ML inference that will classify the data as belonging to a cryptomining attack. If the flow statistic has been classified as a cryptomining attack, the *SendOutput* RCP method will be called. It will send the necessary flow and inference data to the Attack Mitigator component in a *L3AttackmitigatorOutput* object.

Mitigate Detected Attacks

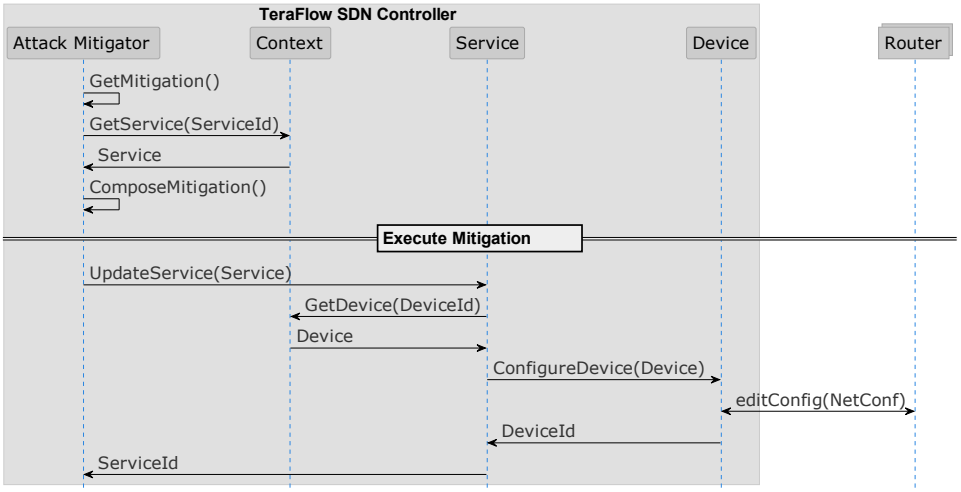


Figure 125. Scenario 3 workflow: Attack Mitigation Workflow (Layer 3)

Figure 125 shows that after the Attack Mitigator (AM) component receives the connection data from a cryptomining attack, it will create a mitigation strategy. Currently, the strategy consists of generating 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 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 TeraFlowSDN will propagate the changes to the Device component and modify the router's ACL rules to drop the connection, thus finishing the current mitigation strategy.

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 corresponding to an attack and predictions corresponding 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 unique attack connections 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.

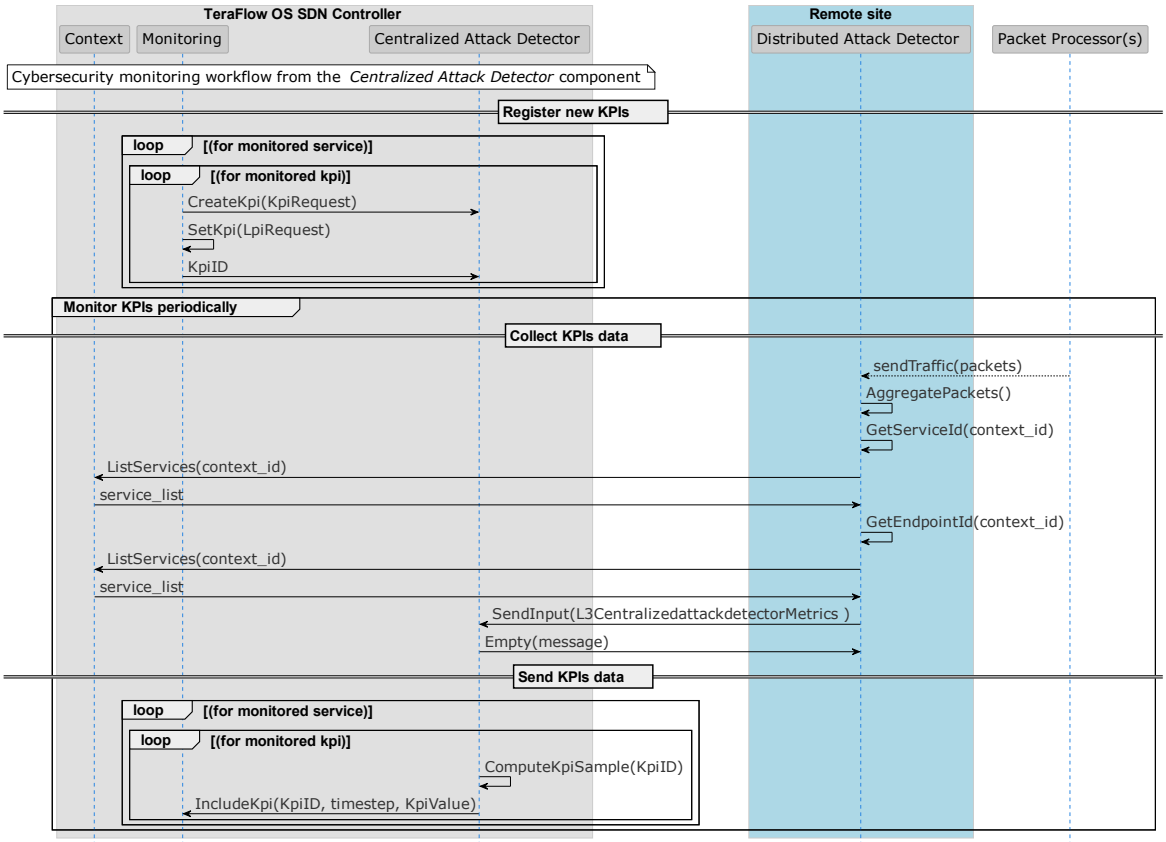


Figure 126. Scenario 3 workflow: Cybersecurity KPIs Monitoring Workflow (Layer 3)

Figure 126 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 each KPI's description and KPI sample type. 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.

6.3.1.3. Results

To assess the attack detection capabilities implemented in the scenario, we now focus on evaluating the resilience of the machine learning model deployed within the Centralized Attack Detector component against adversarial attacks. By understanding the model's response to adversarial attacks, we gain insights into its effectiveness in identifying and mitigating sophisticated intrusion to perform cryptomining attacks. Furthermore, we devote considerable attention to evaluating the scalability performance of the Layer 3 Cybersecurity components. This examination aims to determine the system's ability to sustain optimal performance levels as the volume of data traffic increases. By assessing scalability, we ensure the cybersecurity infrastructure remains effective and efficient even under heavy workload conditions.

Evaluation of the Resiliency of the Cyberthreat Detector Against Adversarial Attacks

The technique of adversarial training was chosen to secure the ML model deployed in the Centralized Attack Detector (CAD) component against the recently appeared adversarial attacks, which are inputs specifically designed to deceive the ML model and trigger incorrect predictions that benefit the attacker.

The adversarial training approach is a technique employed in the field of ML, which entails retraining a model with Adversarial Examples (AEs). This technique aims to enhance the model's robustness and ability to defend itself against potential attacks by exposing it to various adversarial scenarios. By undergoing this process, the model can better adapt to the challenges presented by such attacks, thereby increasing its overall efficacy in combating them. Specifically, we retrain the machine-learning model with high-quality AEs to create a resilient classifier that can defend itself against adversarial attacks.

Therefore, to strengthen the TeraFlowSDN ML-based attack detectors against adversarial attacks, we designed a Generative Adversarial Network (GAN)-based solution to generate high-quality AEs, which are very similar to real attack data but can fool the ML-based attack detector by misclassifying them. These high-quality AEs can be used later to retrain the TeraFlowSDN ML models and fortify the attack detectors against these sophisticated attacks.

Specifically, once generated, these AEs are incorporated into the training dataset, and the ML model is retrained with it to increase its robustness against the attack. In this way, the decision boundary will be adjusted to classify these AEs correctly. However, it should be noted that it is still possible for an attacker to generate new AEs that can cause the model to behave improperly. Nevertheless, it should be more difficult for the attacker to produce effective new AEs using only small perturbations. Moreover, this situation will be more challenging as the black-box model continues to be retrained with these AEs in the future. However, adding AEs to the training data set inevitably increases the variability of the data, which can hinder model learning and degrade model performance.

Our solution is inspired by the standard GAN architecture proposed in [GOO14] that consists of two main components: the generator and the discriminator. In [MOZ22] it is shown that this architecture can be used to generate synthetic network traffic data that can fully replace real data in the training of ML models without significant performance loss.

The generative model developed in this scenario extends the MalGAN architecture [HU23]. This design uses the discriminator to model a third component, the unknown black-model target (e.g., the attacked TeraFlowSDN ML model). The discriminator in this specific setting is referred to as a substitute model, as it will try to learn the black-box behavior. Consequently, this configuration implies a higher complexity in the training process than the standard GAN [GON21] as the behaviour of a given classifier that will act as a black-box model during the training phase must also be tracked.

Figure 127 shows an overview of the MalGAN architecture, in which each box contains an ML model that produces predictions, and each circle contains input or output data. These boxes are from left to right: *(i)* the generator, which is the Deep Neural Network (DNN) to be trained for AE generation, *(ii)* the black-box detector, which is the model that is the target of the attack, and *(iii)* the substitute model, which will try to learn the behavior of the target model and will also serve as a trainer for the generator to learn how to produce effective AEs.

Benign data represents the normal traffic transmitted on the network and malign data models the attack that will be manipulated to fool the ML (black box) model into misclassifying it. Unfortunately, experimental observations showed that although the AEs generated by a MalGAN achieved a very good ratio of misclassification when input to the black box model (very close to 100% evasion ratio), they were very different from both real malign and benign examples, which can favor their detection by using simple statistical filters (e.g., based on the mean of the real benign and malign data distributions).

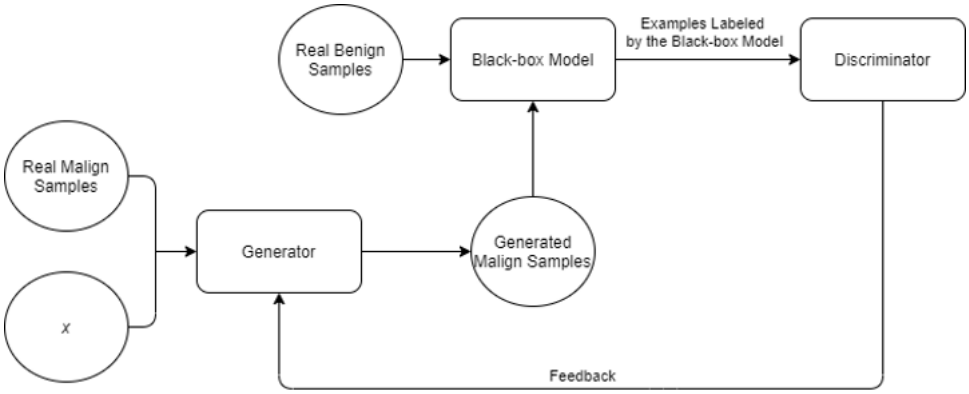


Figure 127. Overview of the enhanced GAN solution based on MalGAN

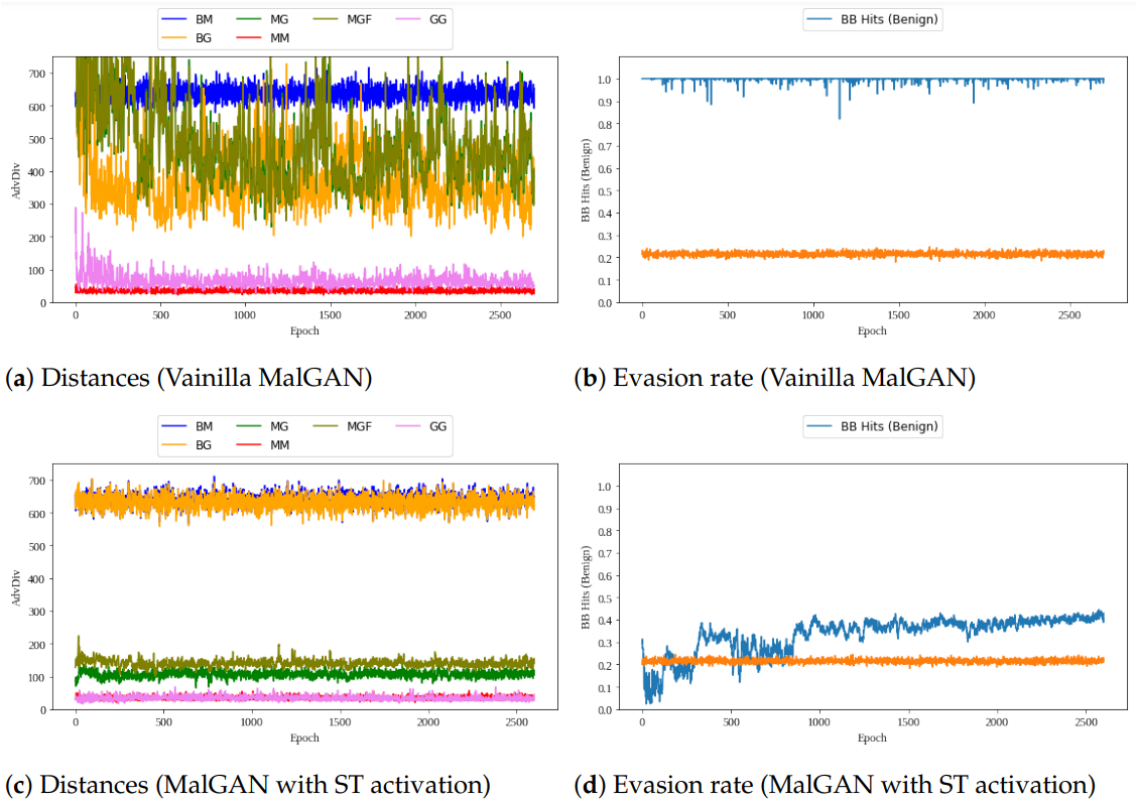
As a novelty, our enhanced version of the MalGAN architecture uses a custom activation function based on the Smirnov-Transform (ST) [GON22] as the last layer of the generator to help generating AEs that mimic the statistical behaviour of real malign examples, transforming the generator output variables into variables that from a statistical perspective are distributed exactly the same as the input variables.

Our proposal is related to a key problem with GANs. Typically, without further tuning, the output distribution of each of the random variables obtained in the generator output is approximately normal. This is related to the mode-collapse problem, a well-reported behavior of the GANs.

To address this problem, the generator's job is facilitated by using as an activation function of each output variable a customized function able to capture the statistical subtleties of each variable of the malign data.

Each customized function implements the inverse of the Smirnov-Transform of each malign data variable. This transformation converts random vectors with normal marginal distributions (the output of a normal GAN) into random vectors with approximately the marginal distribution of the malign data variable.

In addition, the ST activation function is fully deterministic and differentiable, which allows to be seamlessly integrated into the backpropagation step during the GAN training processes. In [GON22], experimental results demonstrated the significant improvement provided by this custom activation function when applied in GAN architectures in terms of the quality of the generated samples.



(a) Distances (Vainilla MalGAN) (b) Evasion rate (Vainilla MalGAN)

(c) Distances (MalGAN with ST activation) (d) Evasion rate (MalGAN with ST activation)

Figure 128. (Left column) Distances between samples of real and synthetic data distributions: BM (benign and malign data), MG (malign and generated malign data), MGF (malign and generated malign data that fool the black-box model), BG (benign and generated malign data), GG (two samples of generated malign data) and MM (two samples of malign data). (Right column) Evasion ratios: (blue) generated malign examples (AEs) and (orange) real malign examples that are classified as benign by the black-box model. In all figures, the x-axis represents the GAN training epochs.

The GAN was trained with the same datasets used in 7.6.1.1.2 of D5.2 and previously described in [PAS18]. The generator and discriminator networks were defined as a stack of three FCNN (Fully Connected Neural Network) layers assuming a moderate complexity in the black box model. In case the black box model was supposed to be more complex, more layers and neurons could be added to the generator and discriminator. The details of the training process and hyperparameters used are similar to those described in [GON22].

Figure 123 compares the results obtained for a Vainilla MalGAN to our proposal (MalGAN equipped with ST activation functions). The top row shows the vainilla MalGAN distances (Figure 128a) and evasion ratios (Figure 128b) at each epoch, and the bottom row plots the distances (Figure 128c) and evasion ratios (Figure 128d) for a MalGAN equipped with ST activation functions.

It can be seen that although the evasion ratios of the Vainilla MalGAN are roughly 1.0, this architecture fails completely to generate synthetic AEs that are close to the malign data and far from the benign data since its distances between (i) maligns and generated malign (MG), (ii) maligns and generated maligns that fool the black box model (MGF), and (iii) benigns and generated maligns (BG) are very far from the expected: BG should be similar to the distance between benign and malign data (BM) and MG and MGF small and close to the distance between two samples of malign data (MM). This is a clear symptom that the generator is producing adversarial (synthetic) examples that are very different from real malign examples and, therefore, they could be identified in a real environment using a simple statistical filter. Note that to fortify an ML model against these types of attack effectively, AEs should be virtually indistinguishable from real malign data.

It is worth noting that the Vainilla MalGAN training process was slightly modified to avoid generating synthetic data that were very far from the real data by substituting the black box labels that were added to the synthetic AEs by their real labels. However, as shown in Figure 128a, the distances of the generated synthetic data concerning benign and malign data are still not achieving the expected good behaviour as they are far from both the benign and malign data.

In sharp contrast, our proposal (MalGAN equipped with ST activation functions) generates AEs that are close to the real maligns (Figure 128c), as (i) the MG and MGF distances are small and close to MM and (ii) BG distances are very similar to BM. The trade-off of this solution is that the evasion ratios (blue line in Figure 128d) are not as good as the obtained with the Vainilla MalGAN but at least greater than the ratio of misclassified malign data (orange line in Figure 128d).

Finally, after high-quality AEs were produced, the black-box model of the CAD component was retrained using these high-quality AEs to create a resilient ML-based classifier that can defend itself against the suggested threat model.

To test the degree of resilience of the retrained ML model, we reserved a dataset of malign data that was not used for training the GAN.

The reserved data has not been seen by the GAN during its training and can, therefore, be considered as data similar to that which could appear in a real scenario. By using examples from this dataset along with Gaussian noise vectors, we generate synthetic samples that are statistically very similar to an attack generated by a malicious attacker.

To measure the degree of resilience that the ML model will offer in real-time, we count the synthetic samples that manage to deceive the new version of the ML model strengthened with our AEs. This approach enables us to assess the robustness of the model to adversarial attacks in a real-world scenario.

Our scenario's result was a retrained ML model, in which the accuracy in detecting new AEs generated with different MalGANs increased to 99% (i.e., the evasion ratio decreased from the original 48 to 1%).

Conclusions

In conclusion, the evaluation of the resiliency of the cyberthreat detector against adversarial attacks focused on using adversarial training to secure the ML model deployed in the CAD component. Adversarial training involves retraining the ML model with AEs to enhance its robustness against potential attacks. The goal was to create a resilient classifier to defend itself against adversarial attacks.

To strengthen the ML-based attack detectors, a GAN-based solution was designed to generate high-quality AEs that closely resemble real attack data. These AEs were used to retrain the ML models in the TeraFlowSDN system, fortifying the attack detectors against sophisticated attacks. By incorporating the AEs into the training dataset, the ML model's decision boundary was adjusted to correctly classify these AEs, making it more difficult for attackers to generate new effective AEs using small perturbations.

The proposed solution utilized an enhanced version of the MalGAN architecture, which employed a custom activation function based on the ST. This activation function helped generate AEs that mimic the statistical behavior of real malign examples, addressing the mode-collapse problem typically associated with GANs. The ST activation function transformed the output variables of the generator into variables distributed exactly the same as the input variables from a statistical perspective.

Experimental results demonstrated the superiority of the enhanced MalGAN equipped with ST activation functions over the vanilla MalGAN. While the evasion ratios of the vanilla MalGAN were high, the synthetic AEs it generated were very different from both real malign and benign examples, making them potentially detectable using simple statistical filters. In contrast, the enhanced MalGAN produced AEs that were close to real malign examples and exhibited distances similar to the expected behavior between benign and malign data.

Next, the black-box model of the CAD component was retrained using these high-quality generated AEs to create a resilient ML-based classifier. The degree of resilience was tested by evaluating the ability of the retrained model to detect synthetic samples generated from a dataset of unseen malign data. The results showed a significant improvement in the accuracy of detecting new AEs, with an evasion ratio decreasing from 48% to 1%.

Scalability Performance Evaluation

In this section, we focus on the scalability performance evaluation of the Layer 3 Cybersecurity components.

Experimental Setup

The proposed experiment aims to evaluate the performance of a Centralized Attack Detector (CAD) component under different load levels. The experiment will be conducted using a set of 10 Distributed Attack Detectors (DADs), which will generate traffic to the CAD component. The performance of the CAD will be evaluated based on the loop time, which is the time required for a request to be processed by the CAD component. The loop time is measured as the time taken for a request to travel from a DAD instance to the CAD instance, which in turn sends a message to the Attack Mitigator (AM) component, back to the CAD instance, and finally to the DAD instance.

To ensure that components can adapt to varying network traffic, the experimental setup leverages the horizontal pod autoscaling capabilities of Kubernetes, in addition to the service provided by the Linkerd service mesh. By dynamically adjusting the number of replicas of the centralized components (CAD and AM) based on the current load level, the horizontal autoscaling service allows for maintaining near-to-optimal performance and responsiveness of the attack detection and mitigation process.

To saturate the CAD, the experiment starts with a single DAD instance and gradually increases the number of DAD instances over time. The number of DAD instances will be increased every 30 seconds until the maximum number of DAD instances is reached. The component will then wait for a stabilization time of 30 seconds before gradually reducing the number of DAD instances every 30 seconds until only one instance remains. Figure 129 shows the behavior just described where the number of DAD instances varies over time.

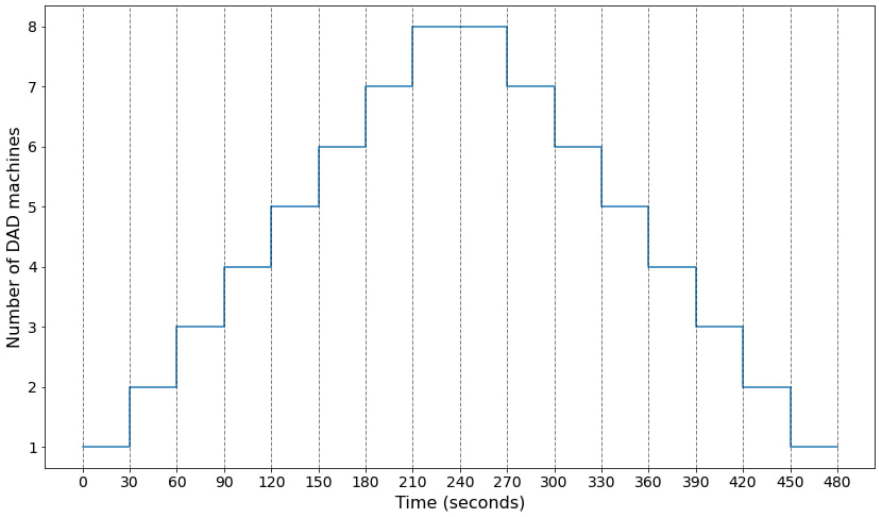


Figure 129. Experimental setup of scalability experiments.

In addition, the experiment involves the injection of network traffic under both normal and attack conditions. The rationale behind this approach is to include the evaluation of the Attack Mitigator component within the assessment of the scalability performance. By doing so, the experiment ensures that the components are subject to realistic conditions, effectively simulating normal and attack traffic scenarios.

Graphical Representations

The following plots will be generated offline using the data generated during each test:

- **Nº of Active CAD Instances Over Time**: This line plot will show the number of active CAD instances over time. The x-axis will show the elapsed time in seconds, and the y-axis will show the number of active CAD instances. The plot will also show the mean CPU usage calculated by the horizontal pod scaling service.
- **Distribution of Loop Times Over Time**: This box plot will show the distribution of loop times for different number of active CAD instances. The x-axis will show the elapsed time in seconds, and the y-axis will show the maximum, minimum, mean, and median loop times for each interval of time without overlap.
- **Number of DAD requests processed Over Time**: This line plot will show the number of requests processed by all active DAD components over time. This diagram will mainly be used to illustrate the increase, stabilization and decrease in load that we want to create in our experiment.
- **Mean and Standard deviation of Loop Times Over Time:** This bar plots will illustrate the behavior of the mean and standard deviation of the Loop Times of all DAD components in 20 second intervals over time.

Parameters

The following parameters will be used in the experiment:

- **Number of DAD Instances**: The experiment will start with a single DAD instance and gradually increase the number of instances over time. The maximum number of DAD instances will be predetermined and should be sufficient to saturate the CAD component. The number of DAD

instances used in the experiment was determined to be 8 after careful experimental evaluation. The objective was to select several instances that would provide a sufficient load to saturate the CAD component, ensuring that it operates under demanding conditions representative of real-world scenarios.

- **Time Interval**: The experiment will increase or decrease the number of DAD instances every 30 seconds. This time interval allows for gradual changes in load and provides stability periods for measurement.
- **Stabilization Time**: After reaching the maximum number of DAD instances, the experiment will wait for a stabilization time of 30 seconds. This period allows the system to stabilize under the maximum load before the gradual decrease in DAD instances begins.
- **Target CPU Usage**: The experiment incorporates the mean CPU usage of the CAD replicas as a crucial metric for dynamic autoscaling of the CAD component. This metric provides valuable insights into the computational resources consumed by the CAD component. By continuously monitoring the mean CPU usage, the autoscaling system can assess the efficiency and scalability of the CAD component in processing the workload generated by the DAD instances. The target CPU usage represents the desired level of CPU utilization that ensures the CAD component is appropriately stressed while still operating within acceptable resource limits. This target value is strategically set based on the system's capacity and performance requirements, aiming to achieve an optimal balance between resource utilization and performance. In this experiment, the Target CPU Usage has been specifically set to 80% of CPU utilization, indicating the desired level of resource utilization that triggers dynamic adjustments in the number of CAD replicas.
- **Maximum Number of CAD Replicas:** The number of CAD replicas is a critical parameter that reflects the scalability of the CAD component. The maximum number of CAD replicas in the experiment will be monitored and controlled. This parameter determines the upper limit of the CAD component's scalability and its ability to handle increasing load levels. By gradually increasing the number of CAD replicas, the experiment assesses the component's ability to distribute the workload effectively across multiple instances and maintain performance under high demand. The maximum number of CAD replicas is determined based on system constraints, such as available computational resources, network capacity, and the desired level of redundancy. It ensures that the experiment covers a range of scalability scenarios, from the minimum number of replicas to the point where adding more replicas no longer yields significant performance improvements. The maximum number of CAD replicas has been set to 20.
- **Inference Batch Size**: The batch size parameter plays a crucial role in the experiment by determining the number of data instances processed simultaneously by the ML model within each CAD replica. The batch size affects the ML model's computational efficiency and memory utilization during inference. A larger batch size allows for parallel processing of more data instances, which can enhance the overall throughput and speed of the model. However, using a very large batch size may lead to higher memory requirements and increased inference time per batch.

  On the other hand, a smaller batch size reduces memory usage and inference time per batch. However, it may result in slower overall processing due to the need for more frequent model updates and potential overhead in processing smaller batches. Through careful

experimentation and evaluation, we determined that a batch size of 10 balances computational efficiency and overall throughput. It ensures that the ML model within each CAD replica efficiently processes a moderate number of data instances simultaneously, achieving good performance without excessive memory demands or inference delays.

Results

As we specified in the experimental setup, we chose to activate up to 10 Distributed Attack Detector machines at constant intervals to show an increase in load in the CAD component. After a stabilization time, we gradually shut down these machines to show the inverse effect, causing a load decrease. This behavior can be observed in Figure 130, where we show the number of requests processed by all DAD machines over time.

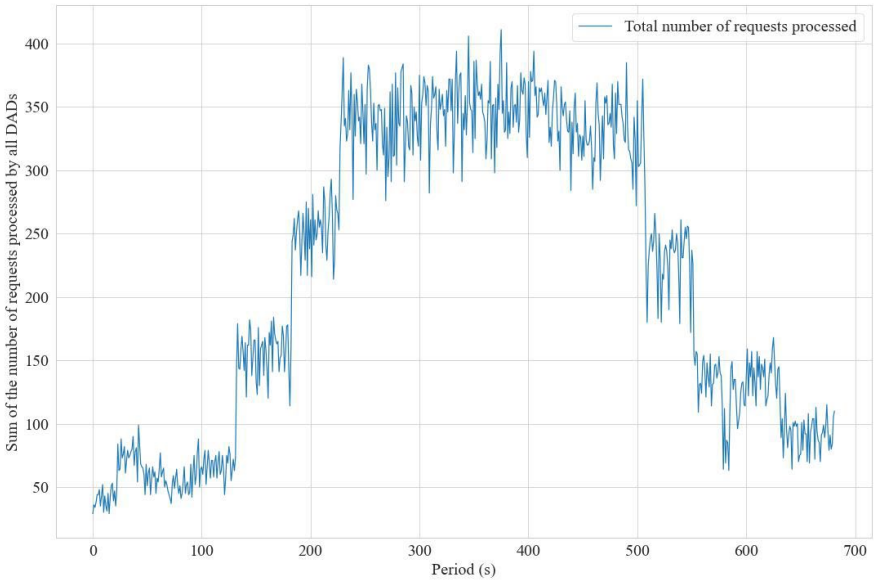


Figure 130. Sum of the number of requests processed by all DADs over time.

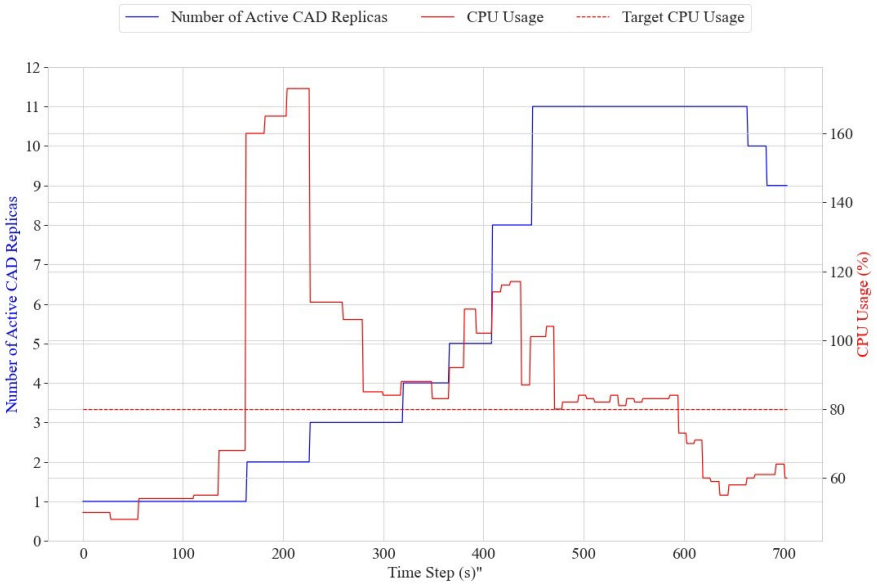


Figure 131. Evolution of CAD replicas and CPU usage over time

In Figure 131 we can observe the evolution of the number of existing CAD replicas along with their mean CPU usage calculated by the horizontal pod scaling service along with the target CPU usage, which in this case was set to 80%. With a limited number of active DADs, CPU usage remains below the set threshold, and the number of CAD replicas remains constant. However, as the number of active DADs grows, the CPU usage starts to exceed the specified threshold. In response, the horizontal pod autoscaling service initiates the creation of new CAD replicas.

The creation of additional CAD replicas allows for efficient distribution of the workload among the CAD components. These newly created replicas become available to the DAD machines through Linkerd's load balancing mechanism, ensuring a balanced distribution of requests.

As the traffic escalates and the number of active DADs keeps rising, the CPU usage consistently stays above the threshold. Consequently, the horizontal pod scaling service continues to generate new CAD replicas, aiming to keep up with the increasing demand for processing power.

However, once the traffic reaches a stabilized state, we observe a plateau in CPU usage (i.e., between 500 and 600 seconds). The CPU usage no longer exhibits rapid growth beyond the specified threshold, indicating that the current number of CAD replicas is sufficient to handle the incoming requests effectively. Consequently, the horizontal pod scaling service halts the creation of new replicas.

Over time, as the workload fluctuates, there may be instances where the CPU usage decreases below the threshold. In such cases, the horizontal pod scaling service identifies the surplus CAD replicas that are no longer required and deletes them. This intelligent scaling mechanism ensures optimal resource utilization, dynamically adapting the number of CAD replicas based on demand.

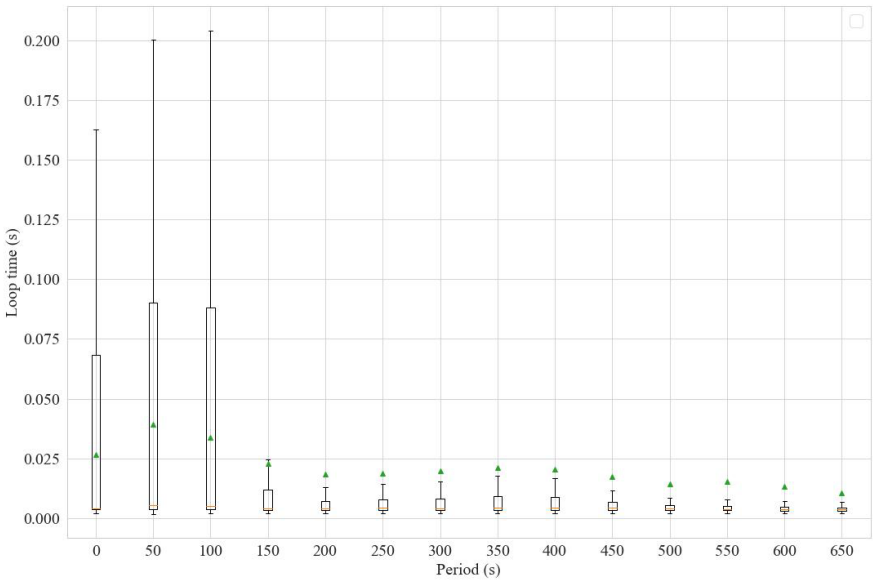


Figure 132. Box plots describing the DAD loop times in 50 second intervals

In Figure 132, we can observe that thanks to the horizontal scaling service that enables the creation of new CAD replicas as needed, the median loop time stays constant throughout the experiment. One noteworthy observation in the experiment is that a larger range of values is present in the third and fourth quartile groups in the first three 50-second intervals. This can be justified because as the traffic suddenly increases, the horizontal pod scaling service takes some time to adjust to the sudden change by creating new replicas. This can increase the loop time for a short time initially as the CAD

component is not able to handle all the traffic as quickly. However, as this upward trend in CPU load increase continues, we also observe in Figure 131 that new replicas are created more quickly as the service considers this increase a trend. Therefore we do not see these big increases in loop time in further stages of the experiment.

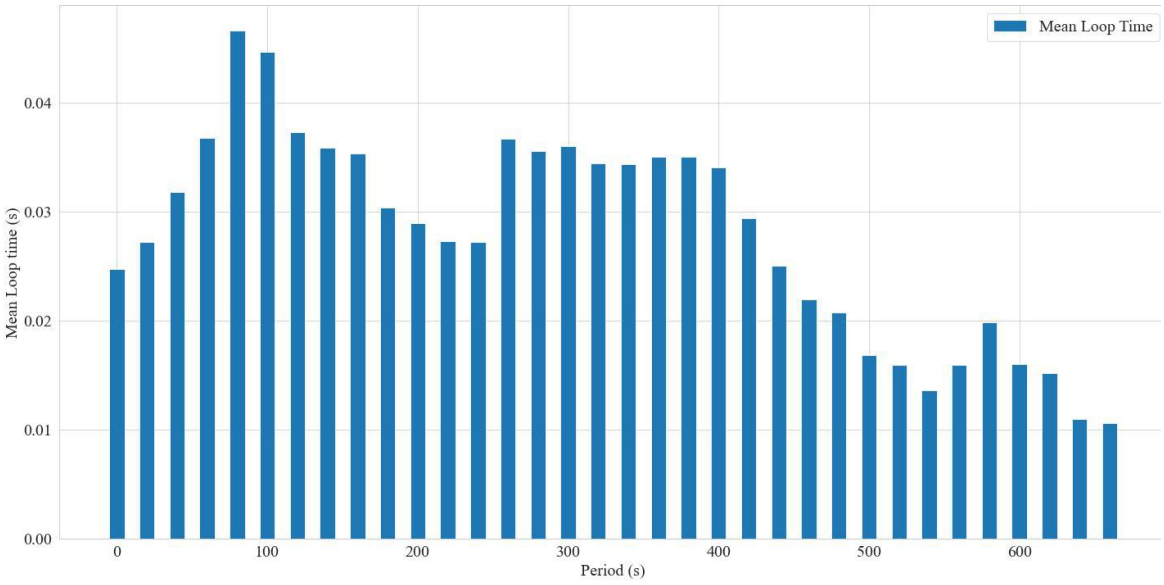


Figure 133. Mean loop time of DAD machines over time in 20 second intervals

In Figure 133 we can observe the mean loop time in all DAD machines over time in 20 second intervals. The figure shows that the mean loop time stays mostly stable throughout the experiment thanks to the creation of new CAD replicas that allow this component to handle the increased traffic. We can notice a small spike in the mean loop time at the beginning of the experiment when the horizontal pod scaling service still had not noticed the increase in traffic and had therefore not triggered the creation of new replicas, causing the requests to be answered more slowly. In a similar fashion, we can observe that at the end of the experiment, the mean loop times are smaller than average, this is because the system is experiencing reduced traffic and still has a high number of replicas to handle it, which will slowly be removed as the horizontal pod scaling service detects the decrease in CPU usage.

Conclusions

Based on the experiment results, it can be concluded that the implemented experimental setup successfully demonstrated the effect of load variation on the CAD component's performance. The number of requests processed by the DAD machines showed an increase during the load increase phase and a decrease during the load decrease phase, validating the experimental setup.

The evolution of the number of CAD replicas and their mean CPU usage over time showed that the horizontal pod scaling service effectively created new CAD replicas as the load increased, maintaining the CPU usage above the threshold. This ensured that the CAD component could handle the increased traffic, demonstrating the effectiveness of dynamic autoscaling. Once the traffic stabilized, the CPU usage reached a steady state, and the number of CAD replicas remained constant, indicating the system's stability under the given load conditions.

The distribution of loop times over time showed that while there were slight increases in loop time during the initial stages of load increase, as the system adapted by creating new replicas, the loop

time stabilized and remained consistent throughout the experiment. The mean loop time in all DAD machines also remained stable, with only a small spike observed at the beginning when the scaling service had not yet triggered the creation of new replicas.

In summary, the experimental results confirmed the scalability and performance of the CAD component under varying traffic loads. The dynamic autoscaling mechanism effectively adjusted the number of CAD replicas based on CPU usage, allowing the component to handle increased traffic while maintaining stable loop times and mean processing times. These findings demonstrate the viability and effectiveness of the proposed system architecture in real-world scenarios involving cybersecurity components and emphasize the importance of load testing and scalability evaluations in ensuring the robustness and efficiency of cybersecurity systems.

6.3.2. Optical Cybersecurity

Optical Cybersecurity's main objective is to perform physical layer attack detection over all the optical services running in the network, within a predefined monitoring cycle (e.g., 30 seconds). This requires the workflow to be divided among a few components, providing the scalability necessary for the task. In D5.2, Section 7.6.2.1, the results of accuracy of the models used to detect and identify the physical layer attacks were reported. In D5.3, we focus on the scalability results. The following presents the testbed setup, dataset, workflows, and results.

6.3.2.1. Emulated Optical Setup for Optical Cybersecurity Experiments

The objective of this setup is to enable us to reproduce Optical Performance Monitoring (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.

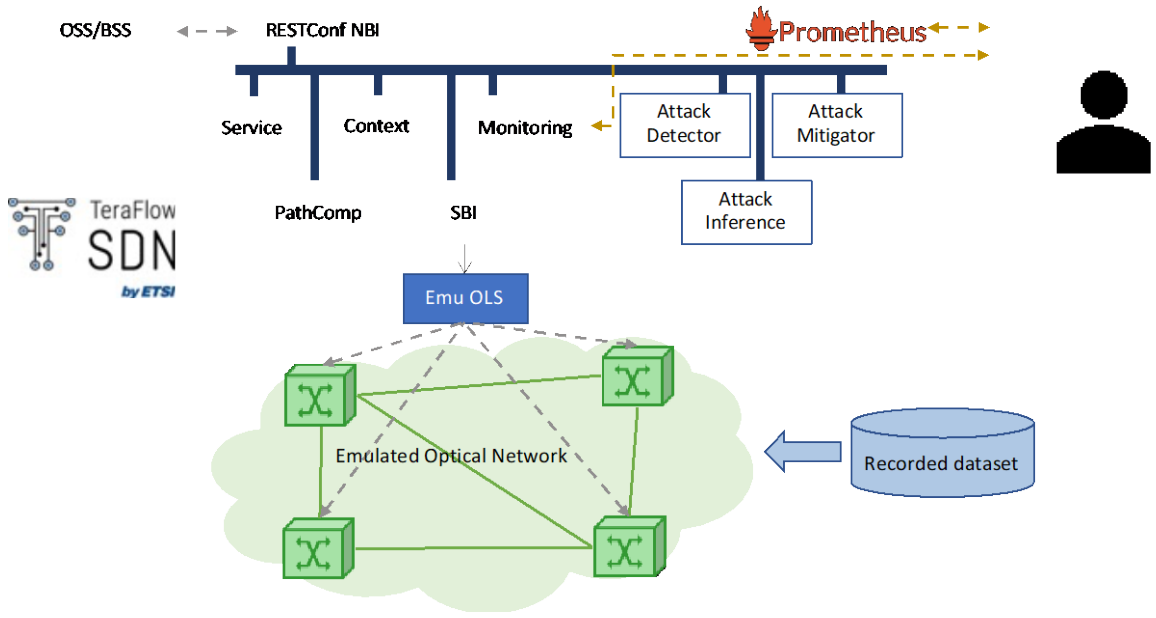


Figure 134. Simplified view of the emulated deployment

Figure 134 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 generate 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 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 were:

- 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 Optical Line System (OLS) that emulates the optical network and communicates with TeraFlowSDN. The emulation happens in the optical service provisioning and the optical service monitoring phases. The emulated OLS assumes an optical data plane with infinite spectrum resources. This enables us to perform stress tests and validate the cybersecurity component's scalability properties without worrying about the service blocking ratio performance of the service provisioning strategy.

The OLS reports OPM values to TeraFlowSDN during the optical service monitoring 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.

6.3.2.2. Workflows

The Optical Cybersecurity scenario follows a similar service creation workflow as the L3 one. However, when initializing the Attack Detector, the component retrieves a list of current services. Figure 135 shows the initialization of the Attack Detector component. The component retrieves a list of all services current running in the network. The Attack Detector filters only those related to optical services and keeps them in an internal list. The internal list maintains the current optical services running in the network by using the events API from the Context component, as explained next.

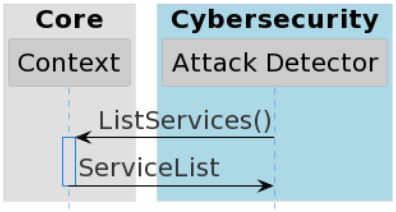


Figure 135. Initializing the Optical Attack Detector component

Figure 136 is similar to the one introduced in the L3 Cybersecurity but reproduced here for completeness. Figure 136 shows the communication among the core and cybersecurity components when a new service is created. First, 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, Section 4.2.4. 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 point, the Cybersecurity component is notified by the Context component about creating a new service. Then, the Cybersecurity component will create relevant KPIs in the Monitoring component.

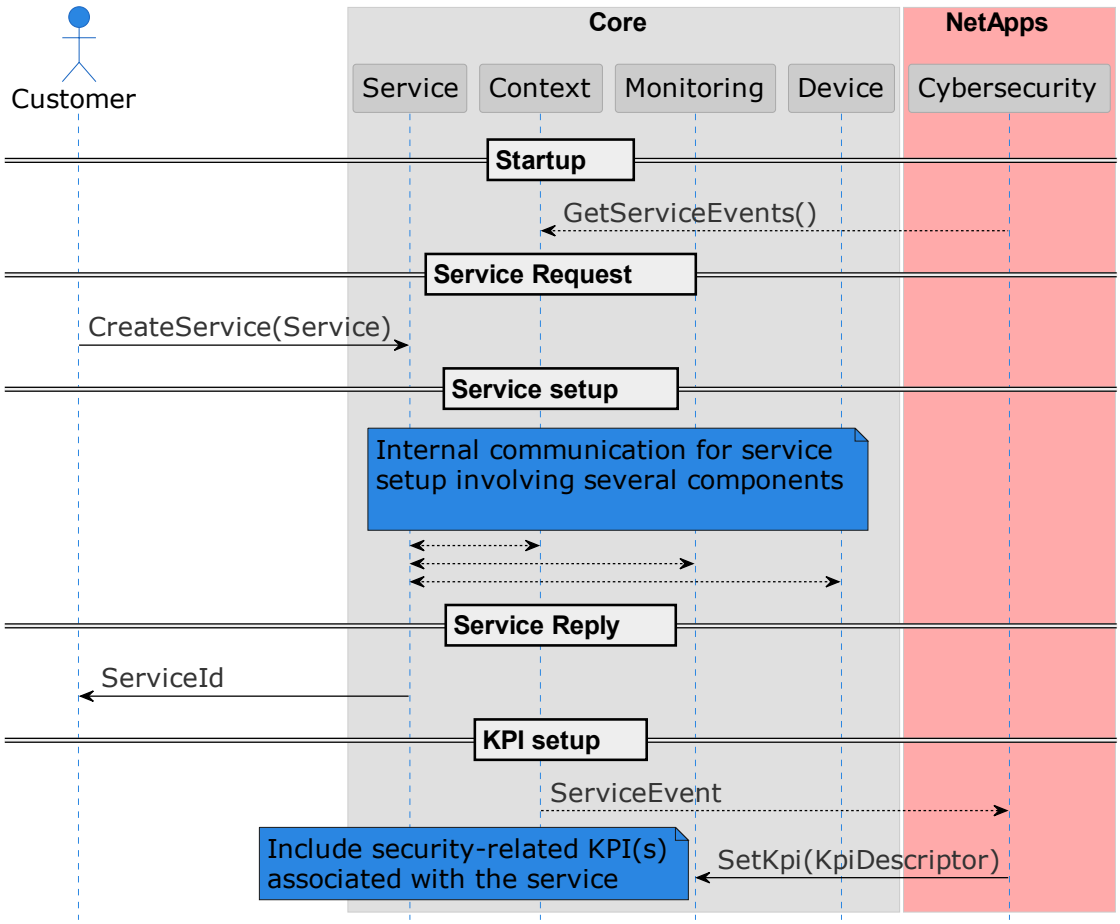


Figure 136. Scenario 3 workflow: General communication when creating a new service

Given that the Attack Detector has an updated list with all the active optical services, the Optical Cybersecurity module periodically monitors all the services, checking whether or not they are under security threats. The operator can select the monitoring cycle (i.e., the interval between two consecutive monitoring loops) based on the pulling rate of OPM data from the optical devices. In current networks, we believe a setting between 30 seconds and 1 minute is appropriate, given that shorter cycles should not yield significant benefits [Vel17, JLT2019].

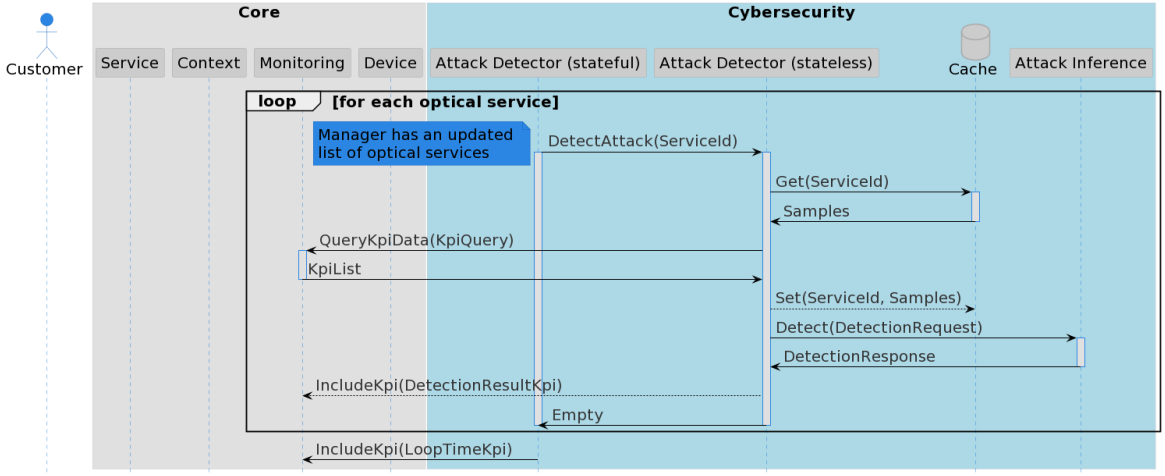


Figure 137. Cybersecurity monitoring of optical services

Figure 137 shows the workflow followed during each monitoring cycle. To better represent the functionality of the Attack Detector, we separate the Attack Detector into two parts: stateful and stateless. The stateful part maintains the current list of optical services and triggers the monitoring cycle. The stateless part is responsible for performing the monitoring cycle for each individual service. This design provided us with a scalable yet flexible Optical Cybersecurity module.

First, for each optical service, the stateful Attack Detector requests the stateless part, delegating the monitoring of each service. Then, the stateless part of the Attack Detector queries the cache for the latest monitoring samples. The cache allows us to offload the Monitoring component from queries that encompass a large number of samples. Then, we can query the Monitoring component requesting only the samples since the last monitoring cycle. The Attack Detector also updates the cache with the latest samples. Then, based on a window of samples, the Attack Detector queries the Attack Inference, which returns whether or not an attack is being observed in the service. The Attack Detector saves the result of the inference in the Monitoring component, enabling the analysis of the monitoring data and the ML inference. Based on this result, the Attack Detector may also trigger the Attack Mitigator.

The figure does not represent the interaction with the Attack Mitigator due to its straightforward implementation. In summary, upon receiving the attack report, the Attack Mitigator will establish a new optical service following the make-before-break principle. Once established, the new optical service will receive all the traffic from the service under attack. Finally, the service under attack will be torn down.

6.3.2.3. Results

In this section, we present the performance evaluation of the Cybersecurity component used to detect physical layer attacks in 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, Section 3.3.4). Then, we also show results related to the scalability properties of the Cybersecurity optical performance analysis loop designed and implemented in TeraFlowSDN. The scalability aspects of the solution have also been presented as a proof-of-concept demonstration [OFC23b].

Accuracy Performance Evaluation

A detailed accuracy performance evaluation has been presented in D5.2, Section 7.6.2.1. In this deliverable, we summarize the accuracy results in Table 15. We adopted an artificial neural network trained over the dataset provided in [JLT2019] to detect known attacks. While the results for specific attack classification (i.e., identifying which specific attack is being launched) have accuracy lower than the targeted 99%, when considering the task as an attack detection (i.e., whether or not an attack is present), the accuracy surpasses the targeted value.

Table 15. Summary of the accuracy performance for the optical physical layer attack detection

KPI	Target	Validation results	Comments
Security	> 99% accuracy (known attacks)	• Accuracy: 0.996 • F1 score: 0.996	Result obtained considering the attack detection task.
	> 90% accuracy (unseen attacks)	• Accuracy: 0.817 • F1 score: 0.803 • Accuracy w/ WAD: 0.99 • F1 score w/ WAD: 0.99	Applying WAD from [JLT2020] with $\tau = 3$ and $\delta = 10$ allowed us to achieve the desired accuracy level.

For the case of detecting unseen attacks, we adopted an unsupervised learning method which allowed us to achieve an accuracy close to the target. Then, to increase accuracy and reduce the presence of false positives, we adopted the Window-based Attack Detection (WAD) method proposed in

[JLT2020]. This method allows for the detection of not only the latest but a window of the latest samples. This allowed us to increase the accuracy beyond the targeted level. The drawback is that the detection may take slightly longer (e.g., four samples in our case). Therefore, considering a monitoring cycle of 30 seconds, the attack detection time increases from 30 seconds to up to 2 minutes.

Scalability Performance Evaluation

In this section, we focus on the scalability performance of two components: the Attack Detector and the Attack Inference. In the case of the Attack Mitigator, scalability is expected to be less of an issue due to the sporadic triggering of events. In order to evaluate the scalability of the entire Cybersecurity module, we designed tests that drastically increase the number of optical services being served by TeraFlowSDN. These drastic increases represent extreme stress cases, due to the fact that optical devices are relatively slow in the configuration. This leads to optical services being added to the network much more slowly than services like L3VPN.

For the scalability performance evaluation, we designed a script to generate many optical service requests to TeraFlowSDN. Moreover, 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 6.3.2.1. 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.

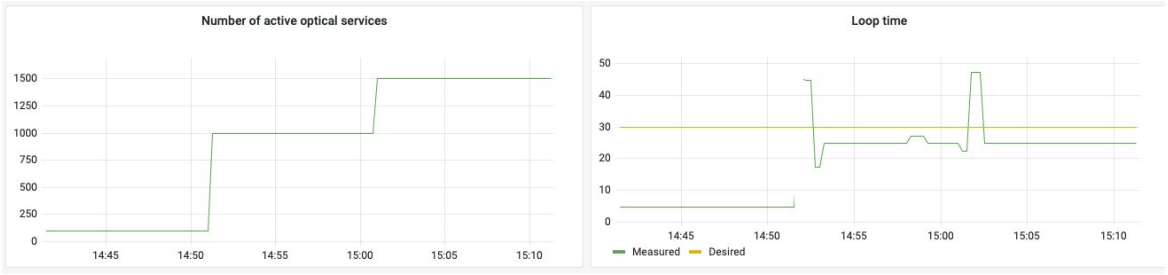


Figure 138. Number of optical services over time and measured loop time

Figure 138 shows the number of optical services in the network (left) and the measured loop time (right, in seconds). For this experiment, the target monitoring cycle is 30 seconds. The experiment starts with 100 optical services, and a loop time below 10 seconds. Then, a drastic increase in the number of optical services (from 100 to 1,000) is reflected by a drastic increase in the loop time. However, the loop time quickly stabilizes below the targeted value. A second drastic increase, now from 1,000 to 1,500 services, shows that the loop time is violated for a few of the cycles. Again, the loop time stabilizes below the targeted value. These results show that the Cybersecurity module is able to quickly adapt to drastic changes in the number of services. In [JOCN23], more detailed experiments, with other loads, are analyzed.

Figure 139. Response time and number of replicas

Figure 139 shows the average response time taken to analyze each service (left) and number of replicas (right) observed during the experiment. We can see that the cache is responsible for a very minor part of the time taken to perform the security analysis of the optical services. Most of the time is taken by the ML inference that uses an unsupervised learning (denoted as UL in Figure 139) algorithm. Moreover, sudden increases in response time are related to the increase in the number of services reported in Figure 138. The number of replicas explains why the loop time remains within the target even for a drastically different number of optical services. We can see that when 1,000 services are present, the Attack Detector increases to 8 replicas, while the Attack Inference scales to 5 replicas. However, when the number of services increases to 1,500, the Attack Inference scales to 8 replicas, allowing the Attack Detector to scale down to 6 replicas.

The results show that the scaling of the Attack Detector and Attack Inference are closely related. When a lower number of Attack Inference replicas is deployed, the Attack Detector needs to wait longer for an answer from the Attack Inference, leading to more replicas of the Attack Detector. When the scaling threshold for the Attack Inference is crossed and new replicas are deployed, the Attack Detector needs to wait less time for an answer from the Attack Inference. This leads to a scaling down of the Attack Detector. In general, when substantial changes in the number of services are observed, it takes only a few minutes for the components to balance the number of replicas.

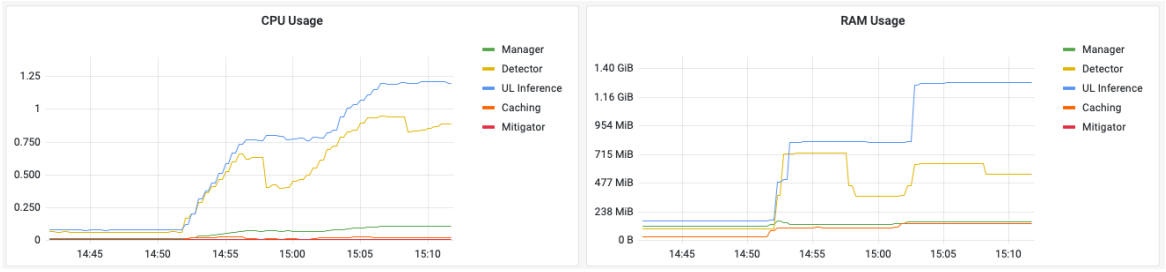


Figure 140. CPU and RAM usage of the Cybersecurity module

Next, we move to the resource consumption of the Cybersecurity module, characterized by Random Access Memory (RAM) and the Central Processor Unit (CPU) usage. Figure 141 shows the evolution of the usage of these resources over time. We can see that, overall, the resource consumption of the module is relatively low. For instance, the overall CPU consumption for the entire module is around 2 CPUs and 2 GB of RAM for 1,500 services. We can also observe that resource consumption is very low when a few services are present (the first part of the plots).

In summary, the developed Cybersecurity module is able to quickly adapt to drastic changes in the number of optical services and remain below the targeted monitoring cycle time. This is achieved by leveraging the autoscaling capabilities of Kubernetes, in addition to a carefully designed architecture of components.

6.4. Scenario conclusions

Table 16 summarizes the KPIs and KVIs achieved by the final implementation of Scenario 3.

Table 16. Target and achieved KPIs and KVIs for Scenario 3.

KPI	**Target**	**Validation results**	
		Layer 3	**Optical**
Security	> 99% accuracy (known attacks)	- Accuracy Score: 0.99966 - False positive: 0 - False negative: 144 - True positive: 1494 - True negative: 421968 - F1 Score: 0.95402	- Accuracy Score: 0.996 - F1 Score: 0.996
	> 90% accuracy (unseen attacks)	N/A	- Accuracy Score: 0.99 - False Positive Rate: 0.00063
Reliability	> 90% accuracy in detecting and avoiding known adversarial attacks.	- 99% accuracy in detecting and avoiding known adversarial attacks (i.e., 1% evasion rate).	N/A
Energy	> 25% resource consumption	- 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	N/A
Scalability	< 20% increase in the mean loop time between low and high traffic load	The mean loop time varies 15.7% (between 0.0247 and 0.0293) between low and high traffic load conditions.	N/A
	< 5 minutes violation of the targeted monitoring cycle time	N/A	The module is able to stabilize from a drastic increase in the number of services (in the order of 10x) in less than 5 minutes.

7. Conclusions

This deliverable reports the efforts in demonstrating and evaluating the performance of TeraFlowSDN release 2.1. Three scenarios are leveraged to drive the integration of TeraFlowSDN, offer concrete workflows to be evaluated, and benchmark TeraFlowSDN across all its components. The main outcomes are a set of testbed setups, workflows, and performance assessment results. These outcomes characterize the performance of TeraFlowSDN across a variety of use cases. The results show that TeraFlowSDN can complete the multiple proposed use cases. The set of KVIs and KPIs achieved has been highlighted in the conclusion section of each scenario.

Regarding testbed setups, each scenario provided a set of setups located at different partners and tailored specifically for testing specific workflow(s). In the case of scenario 1, testbed setups were developed in CTTC, TID, and UBI. For scenario 2, CTTC, ADVA, TNOR, and NEC were responsible for setting up testbeds. Finally, in scenario 3, CHAL, UPM, and TID setup up testbeds to perform the performance evaluation. The testbeds represent the multi-domain, multi-vendor, multi-layer, and multi-protocol networks expected for beyond 5G.

The set of workflows devised ranges from more isolated operations (e.g., setting up devices) to broad, distributed, and coordinated actions, such as setting up slices across multiple network domains using devices from multiple vendors. Some tests were also tailored to test the scalability performance of TeraFlowSDN. A new Load Generator component was developed and integrated into the TeraFlowSDN ecosystem to facilitate the scalability tests.

The performance evaluation work presented in this deliverable shows that TeraFlowSDN provides the necessary support for the designed use cases. In scenario 1, the performance results show that TeraFlowSDN is ready to be integrated into the autonomous operation of networks beyond 5G. The assessment shows that TeraFlowSDN is not only efficient but also scalable. The numbers shown also highlight the stability of the operating delays, represented by small margins between minimum and maximum processing times measured in several operations.

In scenario 2, TeraFlowSDN was tested against several workflows that evaluate its ability to serve as an SDN controller in inter-domain and multi-tenant environments. The results show that TeraFlowSDN provides all the necessary interfaces for multi-domain scenarios. Moreover, advanced functionalities such as providing latency-bound and energy-aware services across multiple domains were demonstrated. Finally, two different ways of implementing inter-domain communication were validated: through direct connection or DLT.

In scenario 3, the preliminary accuracy results presented in D5.2 were substantially extended. This deliverable reports the efforts of taking the AI/ML models previously introduced and improving their scalability, energy efficiency, and reliability while being used within TeraFlowSDN. In other words, the focus was on integrating the AI/ML models within the TeraFlowSDN workflows. In particular, the cybersecurity monitoring of L3 services was enhanced with attack detection mechanisms. These mechanisms were optimized for energy efficiency. Moreover, their ability to ignore adversarial attacks that maliciously modify their input data was demonstrated. The cybersecurity monitoring of optical services was enhanced by devising a scalable architecture that ensures that the monitoring cycles are respected regardless of the number of optical services being monitored.

In summary, TeraFlowSDN excelled in all the different aspects tested. In the future, the activities in the H2020 TeraFlow project will continue in the ETSI Open Source Group for TeraFlow SDN (ETSI OSG TFS). As an open-source initiative, TeraFlowSDN seeks contributions from different partners (e.g.,

academic, and industrial). The objective is to develop an open-source cloud-native SDN controller enabling smart connectivity services for future networks beyond 5G. Since the perspective is that networks continue to increase their heterogeneity, TeraFlowSDN will continue its development of state-of-the-art functionalities that also integrate with existing solutions in the fixed and wireless networks and the cloud computing space.

References

[Aho22] P. Ahokangas, M. Matinmikko-Blue and S. Yrjola, “Visioning for a Future-Proof Global 6G from Business, Regulation and Technology Perspectives,” in IEEE Communications Magazine, doi: 10.1109/MCOM.001.2200310.

[Bar15] G. Barlacchi et al., “A Multi-Source Dataset of Urban Life in the City of Milan and the Province of Trentino,” in Scientific Data, vol. 2, Oct. 2015, art. no. 150055.

[BMV2023] Behavioural model (bmv2) reference P4 software switch, Mar. 2023, Available: https://github.com/p4lang/behavioral-model.

[Bol11] R. Bolla, R. Bruschi and P. Lago, “The hidden cost of network low power idle,” in Proc. 2013 IEEE International Conference on Communications (ICC), Budapest, Hungary, pp. 4148-4153.

[Bol20] R. Bolla, R. Bruschi, F. Davoli and J. F. Pajo, “A Model-Based Approach Towards Real-Time Analytics in NFV Infrastructures,” in IEEE Transactions on Green Communications and Networking, vol. 4, no. 2, pp. 529-541, June 2020.

[Bos2021] Bosk, M., Gajić, M., Schwarzmann, S., Lange, S., Trivisonno, R., Marquezan, C., & Zinner, T., “Using 5G QoS mechanisms to achieve QoE-aware resource allocation,” in 2021 17th International Conference on Network and Service Management (CNSM) (pp. 283-291), 2021.

[Bou2019] M. Boucadair, Q. Wu, Z. Wang, D. King, and C. Xie, “Framework for Use of ECA (Event Condition Action) in Network Self Management,” IETF, Internet-Draft, Nov. 2019, work in Progress.

[COW] Cowboy, https://github.com/ninenines/cowboy

[CTTC22Par] R. Parada, F. Vazquez-Gallego, R. Sedar, and R. Vilalta, “An inter-operable and multi-protocol v2x collision avoidance service based on edge computing”, in 2022 IEEE 95th Vehicular Technology Conference (VTC2022-Spring), IEEE, 2022, pp. 1–5.

[DepG] "TeraFlowSDN deployment guide” [Online]. Accessed: 2023-06-28. Available: https://labs.etsi.org/rep/tfs/controller/-/wikis/1.-Deployment-Guide/1.1.-Introduction

[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

[Gna21] V. Gnanasekaran, H. T. Fridtun, H. Hatlen, M. M. Langøy, A. Syrstad, S. Subramanian, and K. De Moor, “Digital carbon footprint awareness among digital natives: an exploratory study,” [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2985003/919-Article\%2bText-2210-1-10-20211115.pdf?sequence=1 (Last accessed: 10 Feb 2023)

[GOO14] I. Goodfellow *et al.*, “Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. Accessed: May 08, 2023. [Online]. Available: https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html

[GON21] Á. González-Prieto, A. Mozo, E. Talavera, and S. Gómez-Canaval, “Dynamics of Fourier Modes in Torus Generative Adversarial Networks,” Mathematics, vol. 9, no. 4, Art. no. 4, Jan. 2021, doi: 10.3390/math9040325.

[GON22] Á. González-Prieto, A. Mozo, S. Gómez-Canaval, and E. Talavera, “Improving the quality of generative models through Smirnov transformation,” Information Sciences, vol. 609, pp. 1539–1566, Sep. 2022, doi: 10.1016/j.ins.2022.07.066.

[Hos22] T. Hoßfeld, M. Varela, L. Skorin-Kapov and P. E. Heegaard, “What is the trade-off between CO2 emission and video-conferencing QoE?,” ACM SIGMM Records. [Online]. Available: https://records.sigmm.org/2022/03/31/what-is-the-trade-off-between-co2-emission-and-video-conferencing-qoe/ (Last accessed: 10 Feb 2023)

[HPSR23] P. Famelis et al., “P5: Event-driven Policy Framework for P4-based Traffic Engineering,” IEEE 24th International Conference on High Performance Switching and Routing (HPSR), Albuquerque, NM, USA, 2023, pp. 1-3, doi: 10.1109/HPSR57248.2023.10148012.

[Ibm13] “IBM EnergyScale for POWER7 Processor-Based Systems,” [Online]. Available: https://www.ibm.com/downloads/cas/AEOV6OL4 (Last accessed: 1 Feb 2023)

[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

[JLT2020] M. Furdek, C. Natalino, F. Lipp, D. Hock, A. D. Giglio and M. Schiano, “Machine Learning for Optical Network Security Monitoring: A Practical Perspective,” in Journal of Lightwave Technology, vol. 38, no. 11, pp. 2860-2871, June, 2020, doi: 10.1109/JLT.2020.2987032.

[JOCN23] Carlos Natalino, Lluis Gifre, Francisco-Javier Moreno-Muro, Sergio Gonzalez-Diaz, Ricard Vilalta, Raul Muñoz, Paolo Monti, and Marija Furdek, “Flexible and scalable ML-based diagnosis module for optical networks: a security use case,” J. Opt. Commun. Netw. 15, C155-C165 (2023). DOI: 10.1364/JOCN.48293

[KAS20] Kasten, M., & Tewari, N. (2020). SASE: Enabling the Cloud-First Enterprise. Cisco Press.

[Les10] E. Le Sueur and G. Heiser, “Dynamic voltage and frequency scaling: the laws of diminishing returns,” in Proc. 2010 international conference on Power aware computing and systems (HotPower'10), Vancouver, BC, Canada, pp. 1–8.

[Lon2022] Lønsethagen, H., Lange, S., Zinner, T., Øverby, H., Contreras, L. M., Ciulli, N., & Dotaro, E., “Towards Smart Public Interconnected Networks and Services – Approaching the Stumbling Blocks,” Techrxiv, 2022

[MOZ22] A. Mozo, Á. González-Prieto, A. Pastor, S. Gómez-Canaval, and E. Talavera, “Synthetic flow-based cryptomining attack generation through Generative Adversarial Networks,” *Sci Rep*, vol. 12, no. 1, p. 2091, Dec. 2022, doi: 10.1038/s41598-022-06057-2.

[NFVSDN21] S. Merle, et al., Scalable and resilient network traffic engineering using erlang-based path computation element, 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN).

[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.

[OFC23a] Ll. Gifre, et al., “Slice Grouping for Transport Network Slices Using Hierarchical Multi-domain SDN Controllers,” 2023 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 2023, pp. 1-3, doi: 10.1364/OFC.2023.M3Z.9.

[OFC23b] C. Natalino, L. Gifre, R. Muñoz, R. Vilalta, M. Furdek and P. Monti, "Scalable and Efficient Pipeline for ML-based Optical Network Monitoring," 2023 Optical Fiber Communications

Conference and Exhibition (OFC), San Diego, CA, USA, 2023, pp. 1-3, doi: 10.1364/OFC.2023.M3Z.12.

[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," in IEEE Access, vol. 8, pp. 158036-158055, 2020, DOI: 10.1109/ACCESS.2020.3019658.

[PRD] “Propagation Delay,” [Online]. Available: https://wiki.geant.org/display/public/EK/PropagationDelay

[Ren19] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, “A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet”, ACM Computing Surveys (CSUR), vol. 52, no. 6, pp. 1–36, 2019.

[RFC8466] G. Fioccola, et al., “A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery,” IETF RFC 8466, October, 2018.

[SPI22] Spirent SPT-N12U Mainframe Chassis. [Online]. Accessed: 2022-12-15. Available: Spirent SPT-N12U Mainframe Chassis datasheet – Spirent

[Van12] W. Van Heddeghem, F. Idzikowski, W. Vereecken, et al., "Power Consumption Modeling in Optical Multilayer Networks,'' in Photonic Network Communications, vol. 24, pp. 86–102, Jan. 2012.

[Vel17] A. P. Vela et al., "BER Degradation Detection and Failure Identification in Elastic Optical Networks," in Journal of Lightwave Technology, vol. 35, no. 21, pp. 4595-4604, 1 Nov.1, 2017, doi: 10.1109/JLT.2017.2747223.

[WAN19] Wang, Q., Chen, M., Wu, Y., & Sun, Y. (2019). Software-defined wide area networks: a survey. IEEE Network, 33(6), 216-223.

[Zha18] Q. Zhang, J. Liu, and G. Zhao, “Towards 5G Enabled Tactile Robotic Telesurgery,” arXiv preprint arXiv:1803.03586, 2018.

[ZOO] “The Internet Topology Zoo,” [Online]. Available: http://www.topology-zoo.org/dataset.html (Last accessed: 21 Apr 2023)

