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

D4.1: Preliminary Evaluation of TeraFlow Security and B5G Network Integration

Deliverable type	R
Dissemination level	PU
Due date	31/12/2021
Submission date	29/12/2021
Lead editor	Rahul Bobba (NEC)
Authors	Pol Alemany (CTTC), Ricard Vilalta (CTTC), Ricardo Martínez (CTTC), Lluis Gifre (CTTC), Javier Vilchez (CTTC), Laia Nadal (CTTC), Michela Svaluto-Moreolo (CTTC), Ramon Casellas (CTTC), Luis Mata (UPM), Luis De Marcos (UPM), Alberto Mozo (UPM), Stanislav Lange (NTNU), Min Xie (TNOR), Carlos Natalino (CHAL), Paolo Monti, (CHAL), Antonio Pastor (TID)
Reviewers	Georgios Katsikas (UBI), Sergio González (ATOS)
Quality check team	Adrian Farrel, Daniel King (ODC)
Work package	WP4

Abstract

This report targets the design and the development of the TeraFlow OS Security and Integration components that are essential for future network security. These components will build upon and also enhance base technologies. For instance, the Cybersecurity component will use advanced machine-learning (ML) techniques for analysing network traffic to detect intrusions and malicious network traffic and protect the network against sophisticated attacks to ML components. In order to avoid scalability and latency problems, a hybrid central and edge ML architecture will be designed. The Distributed Ledger Technology component will provide novel security solutions based on Blockchain technologies that decentralize critical and sensitive data and services of the TeraFlow OS. Furthermore, the Compute Integration and Inter-domain components will provide mechanisms and interfaces for the integration of network resources, possibly from other domains, and the connection to other networks, respectively.

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.

Version History

Version	Date	Notes
0.1	01.Aug.2021	Table of contents created (CTTC)
0.2	01.Dec.2021	Contributions to tasks and updates (All Partners)
0.3	02.Dec.2021	Added Exec Summary (NEC)
0.4	03.Dec.2021	Added Introduction and Conclusion Sections (NEC)
0.5	06.Dec.2021	Added List of Figures and made minor corrections (NEC)
0.6	07.Dec.2021	Added Section 5.1.3 (CTTC)
0.7	13.Dec.2021	Enhanced Section 2 and made minor upgrades (NEC, CTTC)
0.8	17.Dec.2021	Internal review (UBITECH)
0.9	28.Dec.2021	Reviewers comments addressed (NEC + all partners). Quality review. Submission ready version

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

EXECUTIVE SUMMARY

The TeraFlow project aims to create a novel cloud native Software Defined Networking (SDN) controller for beyond 5G networks. This deliverable summarizes the activities of Work Package 4 (WP4) and describes the design and the development of the TeraFlow OS Security and Integration components. While Work Package (WP3) focuses on the **core** Teraflow OS components, WP4 focuses on the **security and integration** related Teraflow OS components. In order to tackle the security and integration aspects, WP4 has been structured into three tasks namely T4.1, T4.2, and T4.3. The preliminary results achieved for each task are described in detail in this report. The results include implementation details as well as publications of papers addressing the research challenges and implementation approaches.

The first task (i.e., T4.1) is concerned with the cyberthreat analysis and protection, and is targeted towards designing and implementing an advanced cybersecurity solution. This solution is crucial for protecting TeraFlow's network infrastructure in the SDN domain against sophisticated attacks at optical, network, and transport layers. Given the complexity of this task, it is performed over multiple services and specifically designed components address the individual requirements. Broadly speaking, the work is split across two different components: the Centralized Cybersecurity component and the Distributed Cybersecurity component. The Centralized Cybersecurity component is responsible for continuously assessing the security status of optical channels, as well as consolidating the security statuses reported by the Distributed Cybersecurity component. The Distributed Cybersecurity component is responsible for monitoring services at the network layer. As a preliminary result, the Cybersecurity component has implemented its fundamental procedures. The preliminary performance and scalability results of the modules have been published in a conference paper and are outlined in this report.

The second task (i.e., T4.2) describes the design and development of a Distributed Ledger Technology (DLT) focusing on distributed ledgers and smart contracts to secure 5G networks. The key features of DLT or Blockchains in particular, namely, decentralization, immutability, and transparency, make their use appealing for managing resources and services in multi-tenant networks. Blockchains replace centralized network management with replicated databases, which lead to a resilient and trustworthy platform for storing and processing sensitive data. In the TeraFlow project, DLT is used in the multi-domain scenario to record actions by the internal components and manage TeraFlow OS configurations. Additionally, the DLT serves as the data backbone for collaboration among multiple TeraFlow OS nodes by sharing the SDN resources available in their transport network infrastructures. As a preliminary result, this report describes the implementation of the DLT module based on the modular architecture of Hyperledger Fabric. Another result achieved is the publication of a paper presenting a Blockchain-based architecture to provide SDN actions to configure connectivity services in transport domains.

The third task (i.e., T4.3) provides the means for integrating the TeraFlow OS in other networks, in particular, beyond 5G (B5G) networks. Two components play a key role in the integration: the Compute Integration component and the Inter-domain component. The Compute Integration component provides integration with a B5G network, such as at edge and cloud resources and Network Function Virtualization (NFV) orchestrators. The Inter-domain component focuses on providing the necessary protocols consuming the previously defined data models to support the deployment and interconnectivity between transport network slices. As a preliminary result, a

Blockchain is used for the inter-domain management in which all the transport domains collaborate among themselves forming an end-to-end domain connectivity service (CS).

This report documents the first cycle of the TeraFlow project outlining the progress of design and development of the security components and concludes by outlining the goals and next steps of the tasks in the second cycle of WP4.

Content

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CNF	Containerized Network Function
CRUD	Create, Read, Update, Delete
CS	Connectivity Service
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DLT	Distributed Ledger Technology
E2E	End-to-End
E2E CS	E2E Transport Connection
IDC	Inter-Domain Component
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MDB	Monitoring DataBase
ML	Machine Learning
NEP	Node Edge Point
NFV	Network Function Virtualization
NFVO	Network Function Virtualization Orchestrator
OPM	Optical Performance Monitoring
OSM	Open Source Management and Orchestration
PDL	Permissioned Distributed Ledger
PE	Provider Edge
REST	Representational state transfer
RPC	Remote Procedure Call
SC	Smart Contract
SDK	Software Development Kit
SIP	Service Interface Point
SL	Supervised Learning
SSL	Semi-Supervised Learning

UL	Unsupervised Learning
UUID	Universally Unique Identifier
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VNF	Virtual Network Function
VPN	Virtual Private Network
WAN	Wide Area Network
WIM	WAN Infrastructure Manager

1 Introduction

TeraFlow delivers a next generation open source cloud native SDN controller providing smart connectivity services to B5G networks. Ensuring the security of the TeraFlow OS and its underlying components is paramount, and this deliverable describes the progress made in the design and development of the TeraFlow OS Security and Integration components that are needed for securing and protecting the network. This deliverable describes the TeraFlow OS Security and Integration components, including the underlying concepts, specifications, and an evaluation of the components' features. WP4 is structured into three tasks and the work in each task is subdivided into one or more components to achieve the stated goals. For each component in a task, this deliverable describes the design overview, the interfaces, and preliminary results. The design overview also describes the components and how they are integrated into the overall TeraFlow architecture. The interfaces section provides details of the workflows, the associated APIs, and information exchange between the components. The preliminary results section describes the integration and implementation details including descriptions of interfaces, such as APIs (Application Programming Interfaces) and SDKs (Software Development Kits). The exact implementation and integration details are provided in the Milestone 4.1 (MS 4.1) report. Finally, this deliverable provides an outlook and next steps for the forthcoming component features that will be included in the final release and described in deliverable D4.2.

1.1 Purpose

The purpose of this deliverable (D4.1) is to provide a preliminary evaluation of TeraFlow security and B5G network integration. It provides an overview of the progress of the TeraFlow OS Security and Integration components. More specifically, this deliverable describes each component's design architecture, interface development, integration details, and preliminary results. D4.1 provides the foundation for D4.2, which is the final evaluation of TeraFlow security and B5G network integration.

1.2 Relation with other Deliverables

This deliverable takes inputs from MS2.1 which includes initial use case definitions for the proposed scenarios (B5G, automotive, cybersecurity), requirement elicitation, and the draft architecture. D4.1 provides the foundation and inputs to D4.2 which is the final evaluation of TeraFlow security and B5G network integration. This deliverable also relates to D3.1 which provides design and implementation aspects of the core TeraFlow OS components and how they interface with the security and integration Teraflow OS components. Further, this deliverable relates to D5.1, which describes the validation and evaluation of the proposed scenarios (B5G, automotive, cybersecurity).

1.3 Structure

This deliverable is structured as follows: Section 2 presents an overview of the TeraFlow OS Security and Integration components that are developed in WP4. Sections 3 through 5 highlight the design overview, interfaces, and preliminary results of the various security TeraFlow OS Security and Integration components across the three tasks in WP4, namely T4.1, T4.2, and T4.3. Section 6 provides the conclusions and next steps.

2 Overview of TeraFlow OS Security and Integration Components

This section provides an overview of the TeraFlow OS Security and Integration components in the context of WP4. Table 1 shows how these components are mapped to the various WP4 tasks and indicates the partners that have been carrying out their design and implementation.

Table 1 Mapping of Security and Integration components to WP4 tasks and contributing partners

WP4 Task	Security and Integration Component Name	Involved Partners
T4.1	Centralized Cybersecurity Component	CHA
	Distributed Cybersecurity Component	TID, UPM
T4.2	Permissioned Distributed Ledger and Smart Contracts	NEC, CTTC, TNOR
T4.3	NFV Orchestrator	CTTC,
	Inter-domain Component	TNOR, NTNU, CTTC

In Section 3 we describe the Centralized Cybersecurity component and the Distributed Cybersecurity component. The Centralized Cybersecurity component (section 3.1) focuses on detecting and mitigating the security threats that the target the physical layer in optical networks. The Distributed Cybersecurity component (section 3.2) will provide the TeraFlow OS with a continuous assessment of the security status of IP services.

Section 4 presents distributed ledger technologies and smart contracts, and discusses how the DLT component (section 4.1) integrates and interfaces with the TeraFlow OS, enabling all TeraFlow OS components to record, read, and register information on the DLT as well as to use smart contracts for enhancing device and component security.

Section 5 introduces the NFV Orchestrator (section 5.1) and Inter-domain component (section 5.2). The front end of the NFV Orchestrator is the Compute component described in section 5.1.1. The Inter-domain component provides dedicated Quality of Service (QoS) aware inter-domain connectivity services and enables interaction among peer TeraFlow OS instances.

3 Cyberthreat Analysis and Protection

Cyberthreat analysis and protection is a crucial function that SDN controllers need to provide. TeraFlow OS has one of its apps devoted to performing this task. However, such a task is complex, and needs to be performed over a number of different services in the network. Each of these services might have specific cybersecurity needs, and therefore needs to be addressed by components specifically designed for the particular service.

TeraFlow devotes T4.1 to addressing these issues in a comprehensive way. The TeraFlow OS tackles cybersecurity with two different components: the centralized and the Distributed Cybersecurity components. The Centralized Cybersecurity component is responsible for continuously assessing the security status of optical channels, as well as consolidating the security statuses reported by the Distributed Cybersecurity component. The Distributed Cybersecurity component is responsible for monitoring services at the network layer, i.e., by analysing the exchanged packets. If malicious activities are detected by the Distributed Cybersecurity component, it may trigger countermeasures locally or notify the Centralized Cybersecurity component depending on the nature of the threat. Due to the complexity of analysing packets and the expected high volume of messages to be analysed, the TeraFlow OS implements a distributed component, which can be co-located with the monitored devices, reducing the network overhead, and enabling prompt responses to detected threats. This architectural decision enables more scalable deployments of the Distributed Cybersecurity component, in addition to quicker responses to threats.

The first version of the TeraFlow OS security and Integration components (v1 code freeze in MS4.1) focused on the internal workflows of the Cybersecurity components, with minimal integration with other TeraFlow OS components. The Centralized Cybersecurity component (see Section 3.1) focused on its internal workflow and its integration with the Context and Monitoring components to access the list of active services maintained by the context component and Optical Performance Monitoring (OPM) data obtained and stored by the Monitoring component. The Distributed Cybersecurity component (see Section 3.2) focused on its internal workflow, enabling packet-level monitoring, with the integration with other components being left for a future step. In the following sections, each the components in described in detail.

3.1 Centralized Cybersecurity Component

The Centralized Cybersecurity component focuses on detecting and mitigating the security threats that target the physical layer in optical networks. For this purpose, the component continuously assesses the security status of the optical services currently deployed under TeraFlow OS management.

This section describes the efforts, decisions, and implementation made towards the code freeze for v1 of the TeraFlow OS Security components as reported in detail in MS4.1. For this step, we focused on defining an internal architecture for the Centralized Cybersecurity component that takes into account the particularities of this component and takes the most advantage of the microservice architecture adopted by the TeraFlow OS.

In more detail, for this code freeze (v1) we focused on the integration of the Cybersecurity component with the Context and Monitoring components in order to obtain the relevant information to perform attack detection. In the following subsections, we describe in detail the design, interfaces, and preliminary results of the Centralized Cybersecurity component.

3.1.1 Design Overview

Figure 1 shows an overview of the TeraFlow OS Security components, highlighting the Centralized Cybersecurity component and the other core components which are used by the Cybersecurity component. First the Cybersecurity component uses the services provided by the Context component to gather up-to-date information regarding the optical services currently running. Then, the monitoring service is used to obtain current monitoring information related to each one of the running services. Finally, the Service component is used in the case where an attack is detected and needs to be mitigated by, e.g., reconfiguring a particular service. For the v1 code freeze, we focused on the integration with the Context and Monitoring components, while the implementation of mitigation strategies and the integration with the Service component is left for the next code iteration.

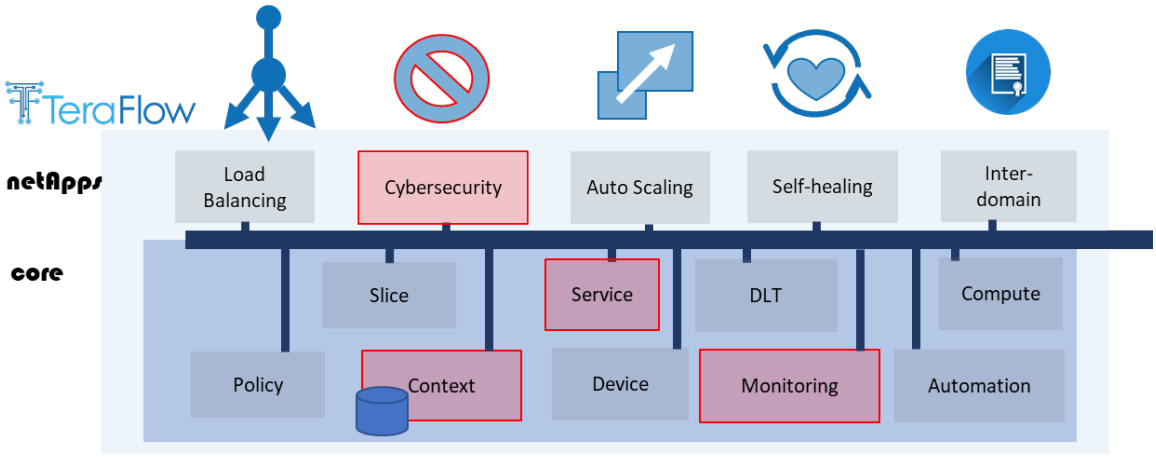

Figure 1: TeraFlow OS security components involved in the Centralized Cybersecurity component

The Centralized Cybersecurity component is divided into 3 modules: the Attack Detector, the Attack Inference module, and the Attack Mitigator. Figure 2 illustrates the 3 modules. The *Attack Detector* is the module that executes the security assessment loop. It is responsible for coordinating with other TeraFlow OS Security components, as well as using the functions of the other modules. The *Attack Inference module* has a very specific task: to host and execute the ML model, performing inference over the data received. Finally, the *Attack Mitigator* is responsible for deciding which mitigation strategy is more suitable given a detected attack, the service being targeted, and an attack description (when available).

The decision of dividing the component into 3 modules makes it possible for the component to take the most advantage out of the microservice architecture adopted by the TeraFlow OS. The Attack Detector needs to keep a consolidated view on the current security status of all services to enable more advanced use cases, i.e., it is deployed in a stateful fashion. For example, the consolidated view enables the analysis of correlation among different attacks, which may lead to more advanced attack mitigation strategies [2]. To tackle scalability concerns, the execution of certain parts of the security assessment loop can be parallelized using multi-threading, multi-processing, or async/await strategies.

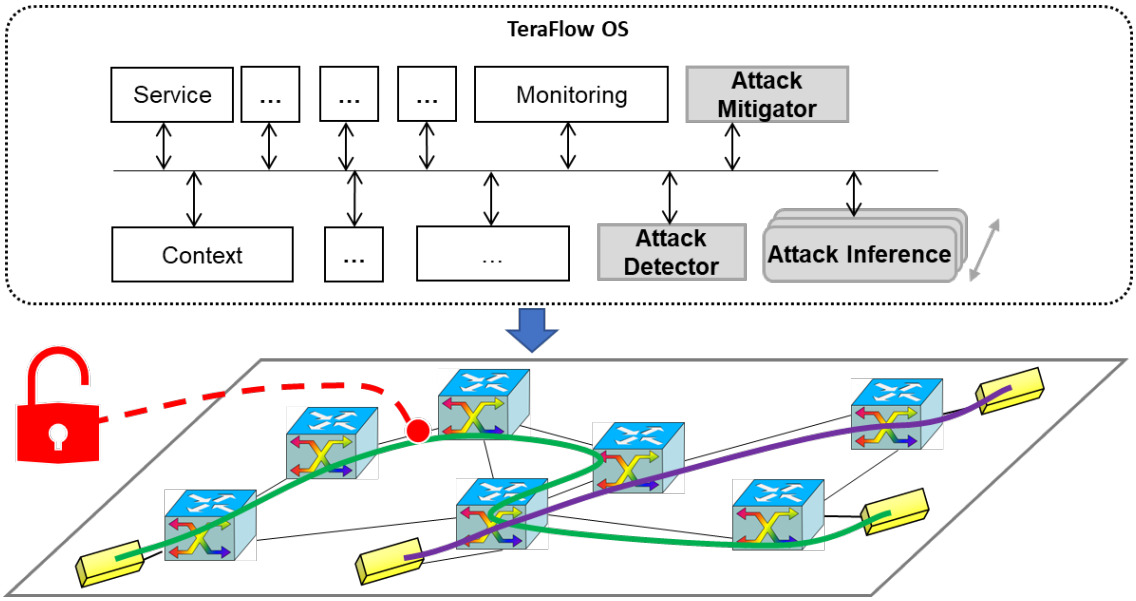

Figure 2: Centralized Cybersecurity components

On the other hand, the Attack Inference module does not need to keep a consolidated view over the statuses of the monitored services, i.e., the Attack Inference module can be deployed in a stateless fashion. This means that as the number of services being monitored scales up/down, the attack inference can scale accordingly to guarantee a prompt inference response to the Attack Detector. The replication and load balancing features of cloud native applications can be leveraged to enable efficient and scalable communication between the Attack Detector and Attack Inference modules.

Meanwhile, the Attack Mitigator needs to scale according to the number of countermeasures needed to mitigate the detected attacks. Again, each mitigation action can be executed independently of the others, meaning that the Attack Mitigator module can be deployed in a stateless fashion.

The analysis of the load incurred over each module, i.e., each of the components will have a different scaling need, justifies the design choice made with respect to the Centralized Cybersecurity component. Therefore, by dividing the responsibilities in this way, we ensure the correct functioning of the component, an efficient use of resources, and a scalable solution.

In the following subsections, we give a few more specific details about each of the modules composing the Centralized Cybersecurity component.

3.1.1.1 Attack Detector

Coordinates the security assessment loop by communicating with the other modules and components.

- Procedures
  - DetectAttack: RPC that allows external components to trigger the security assessment loop. The attack detection loop is also periodically executed.
  - NotifyServiceUpdate: RPC that enables components to notify the Attack Detector of service updates (provisioning/release). For this release (v1), the Attack Detector obtains the list of services direct from the Context component.
- Ports
  - Service port: 10005

    - Metrics port: 9192 (standard value)
- Exported metrics
    - LOOP_EXECUTION_DURATION: Exports how long the security assessment loop takes to run. This information can be used to indicate scalability issues.
- Test status
    - Unit tests for testing the entire code base by simulating the Context and Monitoring components.
- Requirements
    - Shall execute the security assessment loop periodically.
    - Shall obtain and maintain an updated list of active optical services.
    - Shall obtain current relevant monitoring information from the Monitoring component and database.
    - Shall consume the services of the Attack Inference module.
    - Shall inform detected attacks to the Attack Mitigator module.

3.1.1.2 Attack Inference Module

Responsible for executing the ML model for optical physical layer attack detection (and identification) and exposing this model through a gRPC (and optionally a REST) interface. The current implementation is based on the source code publicly available on GitHub[2], which implements the DBSCAN UL model.

- Procedures
    - Detect: Executes the DBSCAN UL model over the received samples.
- Ports
    - Service port: 10006
    - Metrics port: 9192 (standard)
- Exported metrics
    - MODEL_EXECUTION_DURATION: Measures how much time each inference procedure takes. This metric can be used for contextual automatic scaling of the component, and for indication of model performance.
- Test status
    - Unit tests are available for validating the execution of the model.
- Requirements
    - Shall expose one or more ML models to be called through RPCs.
    - Shall be implemented in a stateless fashion.

3.1.1.3 Attack Mitigator

Responsible for computing and coordinating mitigation responses to detected attacks. In the current stage (v1) the module only outputs log messages. A complete implementation is planned for a future stage.

- Procedures
    - Mitigate: Triggers the attack mitigation procedure based on a particular detected attack.
- Ports
    - Service port: 10007
    - Metrics port: 9192 (standard)

[2] DBSCAN serving: https://github.com/carlosnatalino/dbscan-serving-python

- Test status
  - Unit tests are available for validating the execution of the current implementation.
- Requirements
  - Shall receive notifications of attack detection.
  - Shall compute attack mitigation strategies based on the description of the attack detected.
  - Shall coordinate with other components (e.g., service, automation) the steps necessary to mitigate the attack.

3.1.2 Interfaces

Figure 3 illustrates the communication between the centralized cybersecurity microservices and the other TeraFlow OS Security components. In this figure, we focus on the Centralized Cybersecurity component, and its integration with the Distributed Cybersecurity component is left for the next stage of the project. The Monitoring component is responsible for establishing the procedures to periodically get the necessary monitoring information from the appropriate devices. In particular, the Centralized Cybersecurity component is interested in Optical Performance Monitoring (OPM) samples that are obtained from optical devices such as transceivers. These samples are stored in the Monitoring DataBase (MDB) and can be retrieved direct from the database. The access to OPM samples from the MDB in a read-only fashion allows the Attack Detector to obtain the samples as quickly as possible while incurring the lowest overhead possible. However, other alternatives might be suitable and will be further investigated in the future.

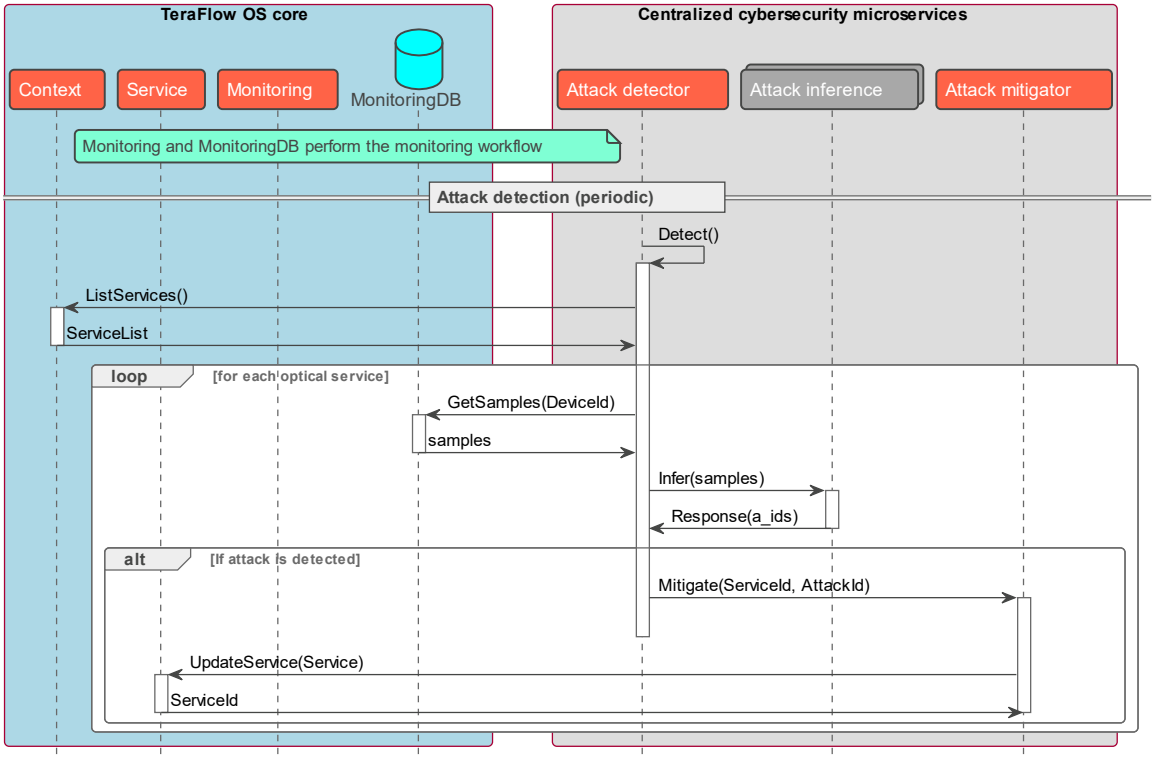

Figure 3: Centralized cybersecurity workflow

The attack detection workflow is periodically executed, triggered by a scheduler that invokes the Detect() procedure. The Attack Detector queries the Context component in order to get a list of the

services that are currently in operation. There are other alternatives to get the current list of services, such as using the streaming feature of gRPC. Such alternatives will be considered in the next version of the component. Once the list of services in operation is obtained, the Attack Detector obtains the OPM samples related to each of the services from the ODB.

Then, the Attack Inference module is used to get the security assessment for each of the channels, i.e., one call per service. Note that by performing one call per service, multiple instances of the Attack Inference module can perform the inference process without impacting the execution of the workflow. Note also that the attack inference can be performed by different ML techniques, i.e., supervised, semi-supervised, or unsupervised learning models. Each of these models may require a specific set of samples. For instance, supervised learning models are likely to require only a single sample to be able to perform attack detection and identification. Semi-supervised learning models are also likely to require only a single sample, but can only perform attack detection. Unsupervised learning models are more likely to require a set of samples (e.g., representing an observation window), and can only perform attack detection. The current implementation of the attack inference uses Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [3], an unsupervised learning model, that can detect attacks at the beginning of their introduction in the service. The Attack Detector analyses the response from the attack inference, which determines whether an attack was detected. If no attack is detected, the Attack Detection module idles until the next attack detection period.

If an attack is detected, the Attack Detection module notifies the Attack Mitigator, which is responsible for computing a suitable solution to mitigate the attack. In the v1 code freeze, the Attack Mitigator logs a warning message in the logging system. For the next release, the module will interact with the Service component to reconfigure the compromised service. More advanced mitigation strategies involving, e.g., the Automation component, can be investigated. Nevertheless, the Service (or Automation) component will be responsible for triggering the set of actions that will reconfigure the service, including all the devices involved.

3.1.3 Preliminary Results

The fundamental procedures of the Centralized Cybersecurity component are implemented in the v1 code freeze. In this initial release, the three modules composing the component are implemented, and their internal communication is validated. In addition to that, the following tasks are completed:

- Unit testing covering 100% of the code,
- Integration with the TeraFlow OS CI/CD pipeline in GitLab,
- Function tests with live data generation from emulated devices,
- Performance and integration assessment of the Attack Inference module using supervised and unsupervised learning models.

The preliminary performance and scalability results of the modules using a supervised and an unsupervised learning model are reported in two conference publications, [4] and [5], the details of which are given in the following subsections.

3.1.3.1 Scalable Physical Layer Security Components for Microservice-Based Optical SDN Controllers

To validate the design and interfaces of the Centralized Cybersecurity component, we implemented the 3 modules of the component and integrated it with a preliminary version of the TeraFlow OS. The components were deployed over a Kubernetes instance. The built-in Kubernetes automatic scaling

facility that leverages CPU usage and load balancing was enabled for the Attack Inference module. The Attack Inference module was configured to have a minimum of one replica and a maximum of ten replicas. In a real-world deployment, for availability reasons, a minimum of two replicas is recommended.

The attack inference was implemented by a TensorFlow Serving[3] instance serving a feed-forward neural network over HTTP. The modules were tested using a dataset obtained from a real-world testbed [6], representing normal operating conditions, and two types of power jamming attacks: in-band and out-of-band jamming. The attack detection loop was executed every 30 seconds.

We observed the performance of the Attack Inference module from the perspective of the Attack Detector module while varying the number of services (denominated lightpaths in this case) from 10 to 100,000 in steps of one order of magnitude. Two metrics were observed: the number of replicas and the response time in milliseconds.

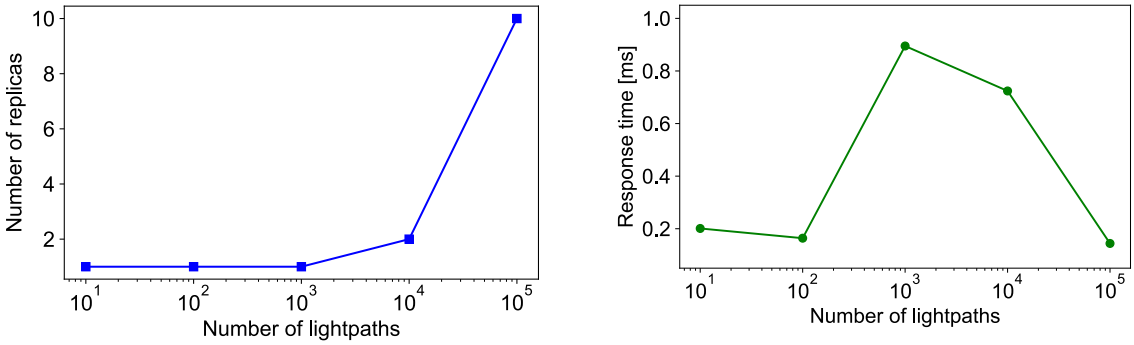

Figure 4: Preliminary performance results using a neural network model

Figure 4 shows the performance results obtained. Figure 4(a) shows how, with the current setup, a single replica of the Attack Inference module can support the security assessment of up to 1,000 services (i.e., lightpaths). With 10,000 and 100,000 services the Attack Inference module requires an increase in the number of replicas. The response time of the Attack Inference module is reported in Figure 4(b) and shows a robust adaptation to the number of services being monitored. A maximum of 1 millisecond is observed for 1,000 services, point at which the scaling due to CPU usage was not yet necessary. With the scaling to two and ten replicas, the response time is reduced to 0.8 and 0.2 milliseconds, respectively. In both Figure 4(a) and Figure 4(b) only specific datapoints were recorded and the line between the points is inferred.

These preliminary results were reported in a conference publication at the European Conference on Optical Communication (ECOC) [4]. They show that the decision of dividing the responsibility of the Centralized Cybersecurity component into 3 modules allows these components, which are under different loads, to scale accordingly and independently. Moreover, given the different objectives of each module, these can be evolved and upgraded independently, reducing the implementation complexity of each one. Finally, the Attack Inference module can take advantage of the optimizations being performed on inference engines such as TensorFlow Serving.

The significantly low response times observed in this preliminary study are obtained in a best-case scenario, i.e., components deployed on the same machine, and an optimized Attack Inference module.

[3] TensorFlow Serving: https://www.tensorflow.org/tfx/guide/serving

However, for the cybersecurity use case, supervised learning models are not always available and/or recommended. For unsupervised learning models, a substantially higher response time is expected. Finally, the preliminary results were obtained using an HTTP/1.1 setting. The adoption of gRPC may introduce a small additional time to the response time due to the need of using a service mesh to perform load balancing.

3.1.3.2 Microservice-based Unsupervised Anomaly Detection Loop for Optical Networks

In this second study, we focused on the use of unsupervised learning techniques for the attack detection task, the interplay between the number of samples used for the inference and the model accuracy, the deployment options, and the scalability properties of the proposed solution. The results of this study are reported in [5].

The interplay between the number of samples used for the inference and the model accuracy is an importance aspect when considering unsupervised learning for the attack inference, since the algorithm needs to analyse a number of samples to be able to detect anomalies. A low number of samples might be insufficient for an accurate detection, while too many samples may negatively affect the performance of the algorithm and increase its execution complexity.

When it comes to deployment options, the Attack Inference module can be deployed as a stateless or stateful component. This decision directly impacts two properties of the Attack Inference module. Firstly, it defines the complexity of the Attack Inference module, given that deploying the component as stateful will require its integration with a cache. Secondly, it defines the number of samples, and therefore the network load, between the Attack Detector and the Attack Inference modules. In [5] we argue that deploying the attack inference as a stateless component presents the best trade-off in terms of complexity of the component and the network load incurred by the component.

Finally, when developing and deploying a critical component such as the Attack Inference module, we need to make sure that the component presents the proper scalability performance. In particular, we need to make sure that the component behaves well when deployed as a stateless component, and that the load balancing of the gRPC service works as expected. The component should provide a stable response time regardless of the load conditions. We designed a set of experiments to assert these properties and the results are reported below.

The study was performed over a development version of the TeraFlow OS deployed over a Kubernetes cluster. The built-in Kubernetes automatic scaling and load balancing processes were enabled for the Attack Inference module. The Attack Inference module was implemented using the Rust programming language to execute the DBSCAN algorithm [3] and expose it as a gRPC service. The Linked network mesh[4] was used to execute load balancing between the Attack Detector and the Attack Inference module.

Figure 5 shows the results of the study. First, we evaluated how the number of samples input to the DBSCAN algorithm affect its accuracy. For the accuracy study, we fine-tuned the two DBSCAN parameters (i.e., the minimum number of points to form a cluster, and the maximum distance between neighbours) to maximize the accuracy in terms of the f1 score. The f1 score summarizes the inference accuracy in the range [0, 1] with 1 being the best accuracy. It considers both the false positive and false negative rates such that both false metrics need to be minimized for the f1 score to increase. The dataset used is the same reported in the previous study [6]. For the experiments

[4] Linkerd network mesh: https://linkerd.io/

reported, 30 attack samples were randomly selected from the dataset. We then evaluated how the number of samples used in the inference (W) impact the accuracy.

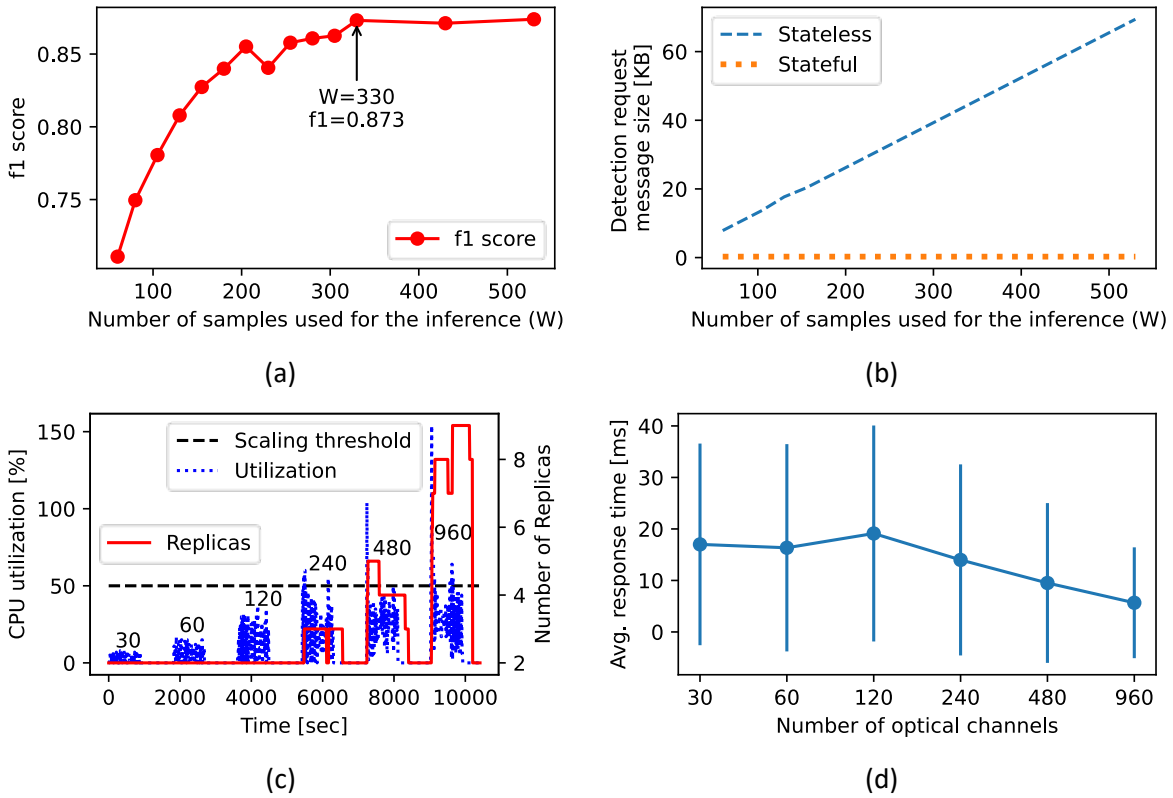

Figure 5 Preliminary performance results using a DBSCAN model

Figure 5(a) shows the accuracy in terms of f1 score for a number of samples varying from 60 to 530 averaged over 50 randomly sampled experiments. We can observe that accuracy significantly increases when the number of samples increases from 60 to 330. From this point, there are no significant gains in further increasing the number of samples.

Figure 5(b) shows the impact of the deployment option on the detection request message size. As expected, when the attack inference is deployed as a stateless component, the message size increases linearly with the number of samples, given that the majority of the message is made up of the data of the samples. When the component is deployed as stateful, the message size is constant, given that the remaining samples are retrieved from a local cache. The total bit rate between the Attack Detector and the Attack Inference module can be derived from this plot by setting an attack monitoring interval and the number of optical channels being monitored.

Figure 5(c) shows the relative CPU utilization (the % of CPUs used over the total CPUs associated with the component) and the number of replicas over time. We set the monitoring interval to 30 seconds for 15 minutes and vary the number of optical channels being monitored over time. Between tests, we stop the requests for 10 minutes to allow the system to return to the initial state. The Kubernetes scaling threshold is set to 50% CPU utilization, i.e., when the average CPU utilization is over 50%, the number of replicas is scaled. First, we can see that the CPU utilization is always reported under 50%, with exception of the beginning of the workload. The minimum number of replicas is set to 2, which is able to support the monitoring of up to 120 optical channels. Then, the number of replicas scales to up to a maximum of 9 replicas to support the monitoring of 960 optical channels.

Figure 5(d) shows the average response time and its standard deviation. We can see that, as expected, compared to the supervised learning model in Figure 4(b), the unsupervised learning model (i.e., DBSCAN) presents a much longer response time. Nevertheless, the results show that the response time is relatively stable, and has answers in the range of 20 milliseconds, which is acceptable in the context of this project.

3.2 Distributed Cybersecurity Component

The Distributed Cybersecurity component focuses on the capture, identification, and mitigation of network threads, implementing a protection layer that is crucial for the correct functionality that SDN controllers need to provide. The Distributed Cybersecurity component consists of three main modules (*Distributed Attack Detector, Centralized Attack Detector, and Attack Mitigator*). The distributed attack detection and mitigation workflow will provide the TeraFlow OS with a continuous assessment of the security status of IP services.

It is worth noting that the Distributed Cybersecurity component includes an instance of the Centralized Attack Detector that is different from the one used in the Centralized Cybersecurity component. The Centralized Attack Detector was designed to be instantiated multiple times in the TeraFlow Controller to deal with different types of attacks and scenarios.

This section describes the efforts, decisions, and implementation made towards the v1 code freeze for the TeraFlow OS Security components. We focused on defining the backbone infrastructure while connecting these parts together. Along with this work, for this code freeze, we focused on the integration of the CI/CD pipeline which has been tailored together with the services while maintaining a minimum of functionality, that is, there are stub functionalities pending to be further implemented in subsequent code freezes, but in general, both the components and the machine learning are all connected.

3.2.1 Design Overview

The Distributed Cybersecurity component maintains two core centralized components along with a distributed component to be placed at a remote site. Figure 6 shows an overview of the current status, all the components live in the TeraFlow OS environment and are connected in the following manner.

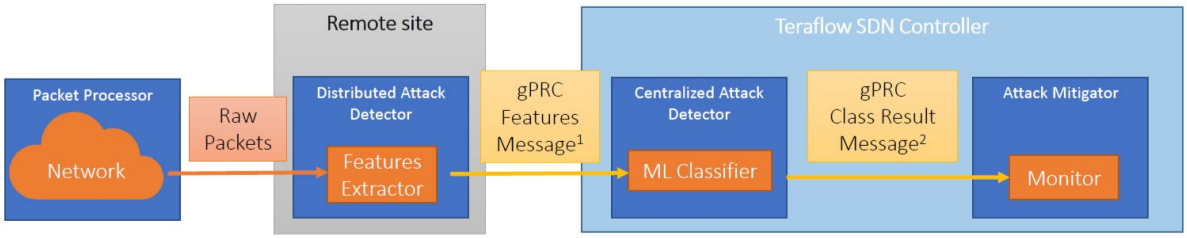

Figure 6: Distributed Cybersecurity components architecture

The *Distributed Attack Detector* will be placed at a remote site (e.g., central office, edge datacentre) and will receive IP traffic from co-located packet processors. In the current deployment the Packet Processor is emulated by reinjecting network packets previously stored in a file by using tcpreply, a standard tool available in Linux systems. The monitoring of IP traffic is expected to use a considerable amount of bandwidth between the packet processors and the Distributed Attack Detector, but avoiding the transmission of huge amounts of telemetry to the TeraFlow Controller. The current

implementation of the Distributed Attack Detector makes use of the TSTAT tool[5] to group packets into TCP connections and extract the corresponding statistical features of each connection to be sent to the Centralized Attack Detector. In the current implementation (v1 code freeze), it uses unary messages to report summarized versions of traffic monitoring information gathered by TSTAT to the *Centralized Attack Detector* component, which will perform the machine learning inferences to detect attacks at the IP level. This approach of sending summarized statistical features to the Centralized Attack Detector also promotes scalability as it removes the need to send complete information about IP traffic to the centralized controller. In subsequent versions, stream messages will be implemented to avoid the delay constraints that can appear with the current unary messages.

The *Centralized Attack Detecto*r consolidates the information gathered from multiples instances of the Distributed Attack Detector. This enables the monitoring of the malicious network traffic, while also forming a view of the security status of IP traffic. Based on the summarized KPIs received from the *Distributed Attack Detector*, this component performs attack detection by implementing a machine learning model embedded in its definition and loaded in ONNX (Open Neural Network Exchange) format. This format allows the embedding of a compiled model and can reduce the overall size of a model as well as accelerate the inference time of the predictions. However, the model is not optimized yet, but this could be a feature to include in the subsequent code freezes.

The machine learning model is then called for every unary message received (but we have considered changing the messages in the future to be streams messages, see preliminary results section 3.2.3) to perform inferences. From this inference, the *Centralized Attack Detector* produces a description of the connection, including within a confidence range as to whether it is an attack or regular traffic. Upon detection of an attack, the *Centralized Attack Detector* notifies the mitigation process with the attack description, providing a comprehensive characterization of the attack properties in order to perform a mitigation strategy.

Upon receiving an attack notification, the *Attack Mitigator* is responsible for computing a viable attack mitigation, depending on the detected attack. However, due to the current status of the development process, the Attack Mitigator has only been tailored to communicate correctly with the aforementioned components. A placeholder has been crafted in place of the mitigation strategy (i.e., a print trace) while the communications have been tested.

These three modules now are connected and in the following months the mitigation strategies are going to be implemented. The realization of the attack mitigation strategy will be coordinated with the Automation component, which will be responsible for performing the necessary actions across the devices in the network to make sure that the mitigation takes place.

3.2.1.1 Distributed Attack Detector

This component detects attacks at remote sites (network edge) in a distributed fashion and classifies them in real time. The focus of attack detection in this component is analysis of packets to identify data plane attacks.

The *Distributed Attack Detecto*r does not deploy a gRPC server but runs a script for obtaining and processing network traffic. This is achieved by utilizing the TSTAT tool, which obtains additional information about the connection status. In addition, it generates log files based on the captured connections, differentiating whether the connection was completely closed or whether it is UDP or

[5] The TSTAT tool: http://tstat.polito.it/

TCP. Then, the information received in real time is read from the selected log file, processed and adjusted to apply the gRPC message format. Finally, it is sent to the *Centralized Attack Detecto*r.

Implementing a machine learning model inside this component to detect attacks is left to future development in subsequent code freezes.

The details of the implemented component are as follows:

- Procedures
  - SendInput: RPC in *Centralized Attack Detector* that allows external components to trigger the machine learning prediction function.
  - load_file: loads the latest TSTAT file. If no file is found, it retries until a log file is found.
  - follow: follows the TSTAT loaded file as an iterator.
  - process_line: processes the received information to match the machine learning model input.
  - run: makes use of the previous procedures to loop through all the received information in order to generate a gRPC message which is sent to the *Centralized Attack Detector*.
- Ports
  - Service port: 10000
  - Metrics port: 9192 (standard)
- Tests status
  - Unitary test to send TSTAT messages to the *Centralized Attack Detector* component (this step will fail due to the lack of connection to this component).
- Requirements
  - Shall receive detailed monitoring data from packet processor devices.
  - Shall generate summarized flow KPIs and send them to appropriate/subscribed components such as the *Centralized Attack Detector*.
  - Shall report detected attacks to the *Attack Mitigator* .

3.2.1.2 Centralized Attack Detector

This component provides attack detection capabilities at the IP layer and a consolidated attack detection mechanism based on reports from the *Distributed Attack Detector*. The *Centralized Attack Detector* utilizes machine learning algorithms to classify the input data received from the *Distributed Attack Detector* component and sends the predictions to the *Attack Mitigator.*

The details of the implemented component are as follows:

- Procedures:
  - SendInput: RPC that obtains the *Distributed Attack Detector* data (as summarized KPIs) to perform predictions with the machine learning model.
  - make_inference: Loads a model and performs the machine learning prediction to generate inferences from the data received.
  - SendOutput: RPC that returns inference information to the *Attack Mitigator* service.
- Ports
  - Service port: 10001
  - Metrics port: 9192 (standard)
- Tests status
  - Unitary test to send dummy TSTAT messages simulating the *Distributed Attack Detector* component in order to check if the system receives them properly.

Additionally, sends the machine learning prediction to the *Attack Mitigator* (this step will fail due to the lack of connection to this component).

- Requirements
  - Shall consume/subscribe to security-related data from the TeraFlow Monitoring core component.
  - Shall process the summarized flow KPIs from the Monitoring component and generate a consolidated data plane security status.
  - Shall report detected attacks to the *Attack Mitigator*.
  - Shall report security status to the TeraFlow Monitoring core component.

3.2.1.3 Attack Mitigator

This component is responsible for computing viable attack remediation solutions, depending on the attack detected by other components. It receives per-connection information from the Centralized Attack Detector. In the current version, the Attack Mitigator has only been tailored to communicate correctly with the Automation component and a placeholder has been crafted in place of the mitigation strategy (i.e., a print trace).

The details of the implemented component are as follows:

- Procedures
  - SendOutput (retrieve information from centralized, answer ok),
  - GetMitigation (obtain mitigation strategy)
- Ports
  - Service port: 10002
  - Metrics port: 9192 (standard)
- Tests status
  - Unitary test to send dummy machine learning prediction messages to the GRPC server to check if the server deployed correctly.
- Requirements
  - Shall consume/subscribe to security-related data from the TeraFlow Monitoring core component.
  - Shall receive attack notifications from the Centralized and Distributed Attack Detection components. Currently only receives from the Distributed Attack Detection component.
  - Shall communicate with the Automation & Policy Manager component to perform attack countermeasures. This is not yet implemented.

3.2.2 Interfaces

The purpose of this code freeze has been to establish the communications between the components correctly. The three components make use of service and monitoring pb2 and gRPC files as well as context pb2. For distinction with the centralized components developed by Chalmers, we have indexed the distributed components with the prefix **I3_**.

As it can be seen in Figure 7, the communication flow starts at the *Distributed Attack Detector* and continues to the *Centralized Attack Detector* which has a dual role, also serving as the client of the *Attack Mitigator*.

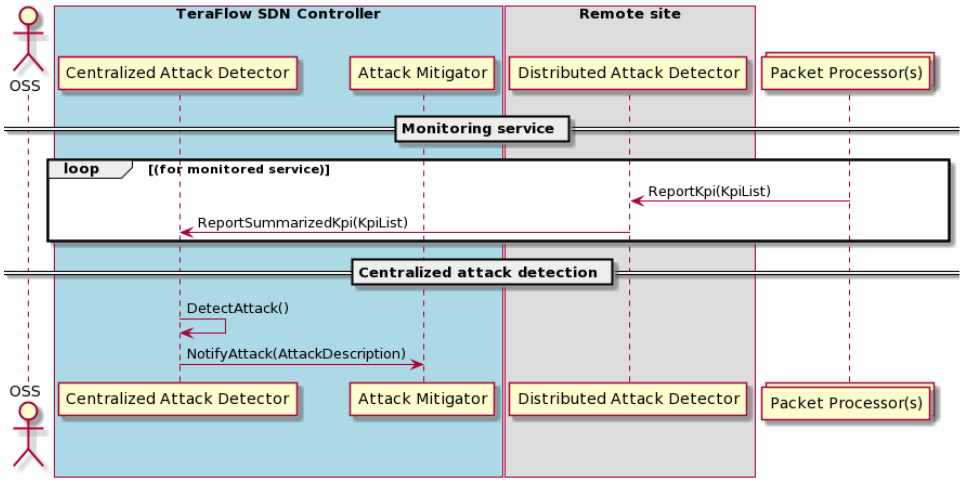

Figure 7: Distributed Cybersecurity component (From deliverable D21)

The *Distributed Attack Detector* receives network traffic packets from the packet processor and extracts information to generate gRPC messages. These messages match the machine learning model input data format plus connection identifiers in order to facilitate the prediction process. Then, the messages are sent to the *Centralized Attack Detector* via gRPC using the SendInput procedure. For this code freeze (v.1), the DetectAttack procedure shown in Figure 7 has not been yet implemented. We plan to integrate this procedure along with its corresponding machine learning model in subsequent code freezes.

The *Centralized Attack Detector* receives the messages from the *Distributed Attack Detector*, so that it provides a service. It then splits the information to select the corresponding network traffic data. Next, it inferences the input and formats the output into a gRPC message, adding the connection identifiers. As mentioned earlier it has a dual role, because it acts also as a client i.e., the messages are sent to the *Attack Mitigator* via the SendOutput procedure to perform the required mitigation strategy for the malicious connections.

The *Attack Mitigator* provides services to the *Centralized Attack Detector* component. To implement such behaviour, it needs its own service definition and thus it has its own RPC. As stated in the previous section, the mitigation strategy, which corresponds to the CreateUpdateService procedure, is currently a stub, meaning that it has not yet been implemented.

Along with these interfaces, the ones resulting from metrics are also not yet implemented and are left for subsequent code freezes.

3.2.3 Preliminary results

The Distributed Cybersecurity component has its fundamental procedures implemented for the code freeze (v1). In this initial code release, the three modules composing the component are implemented, and their internal communication is validated. The results following the implementation up to October 2021 include the following milestones:

- Fully integration of communications in laboratory environment.
- Fully integration of ML in laboratory environment.
- Function testing and integration in TeraFlow environment (GitLab Pipelines).
- Function unit testing covering the higher functions and 100% of the interfaces.

Additionally, some tests were deployed locally to validate the deployment and evaluate the ML learning model.

The scenario implemented for the current version (v1) of the code freeze is shown in Figure 8 and consists of 2 standard Linux-based computers connected by a repeater. Computer A will be in charge of network traffic generation using the tcpreplay tool and a pcap file that contains a capture of real network traffic including a cryptomining attack. All the traffic stored in the pcap file will be reinjected into the network by Computer A, and Computer B will receive it as it was transmitted in real time. Computer B will utilize the tstat tool to capture and group into connections all the received packets on the specified interface. It will then extract network statistics per connection that will be used as input features by the ML component. To send these features to the Centralized Attack Detector, the Distributed Attack Detector will generate a gRPC message with corresponding protobuf format. Each time the Centralized Attack Detector receives a gRPC message from the Distributed Attack Detector, it will extract the connection statistics and use them as input features to the ML classifier that will generate a prediction on the type of connection (i.e., whether the connection is a cryptomining attack or not). A new gRPC protobuf message will be created and sent to the Attack Mitigator with the connection identifiers and the ML inference containing the type of connection that was detected and the confidence level of the prediction. The current version of the Attack Mitigator is a basic placeholder that only logs the messages received from the Centralized Attack Detector to be able to run and demonstrate the integration of all components. The messages received by the Attack Mitigator will be printed on screen if detected as cryptomining attack connections. A more realistic Attack Mitigator will be developed for the second version of the code freeze.

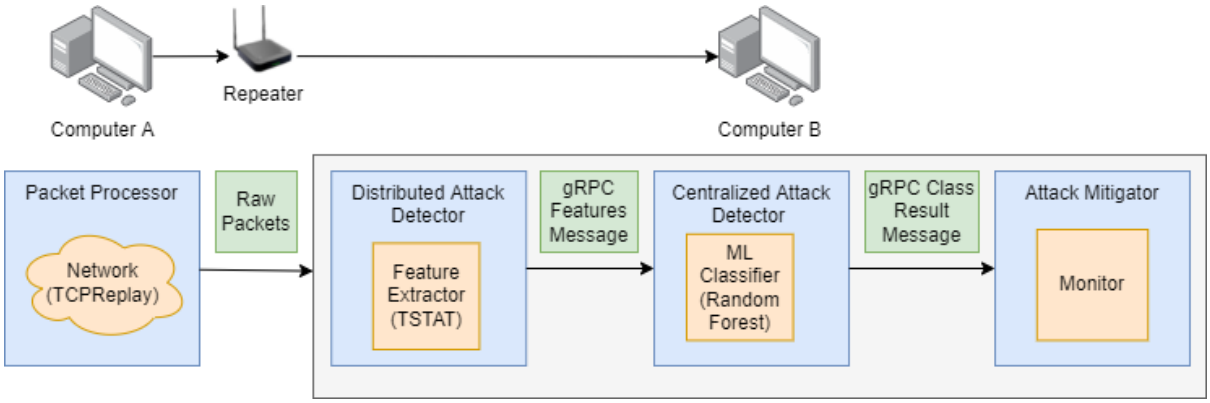

Figure 8 Local deployment of the Distributed Cybersecurity component

The initial validation of this component was done using the tcpreplay tool to reinject a pcap file containing 347k packets from normal traffic and cryptomining attack connections that were received and initially processed by the *Distributed Attack Detector*. Note that 346947 packets were from normal traffic connections and only 51 from cryptomining attack connections. The way in which tstat was configured in this initial validation was to generate a snapshot of a connection each time a packet was received. This aggregation can be inefficient when a burst of packets from the same connection are received in a short period of time as each packet will force tstat to generate a new message to be sent to the Centralized Attack Detector. Moreover, in response to the burst of messages, the Centralized Attack Detector will generate a burst of predictions for the same connection in a very short period of time.

In addition, and as stated in the design overview, an issue appeared while testing the experiment due to the inefficient unary gRPC implementation. To overcome this limitation, the unary gRPC implementation will be replaced with stream gRPC channels in the next release of the code freeze.

In addition, we used two well-known metrics (f1 score and accuracy score) to validate the results of testing that the Random Forest model obtained with the pcap file we re-injected. Note that the Random Forest model was previously trained with a different pcap file captured in a different instant of time but using similar traffic patterns.

- F1 score (range 0 to 1, being 1 the best): 0.9052
- Accuracy score (range 0 to 1, being 1 the best): 0.9999

These results show the potential of the system to detect attacks with accuracy when a well-trained model is used. However, the model in use is a particular placeholder and may need to be replaced with a different model to obtain the required performance in a different scenarios.

It should be noted that the preliminary results obtained only reflect the performance of the proof of concept we developed for the first version of the code freeze. In particular, the TSTAT tool, which we used for packet aggregation and feature engineering, was designed for research purposes and, therefore, it is not adequate for production deployments. Realistic performance in a production environment can be obtained using commercial aggregators based on the Netflow protocol. Recently technologies such P4[6] or eBPF[7] should be explored to implement highly flexible packet aggregators to achieve one step beyond Netflow performance. In this regard, we will investigate the applicability of eBPF technology for the second version of the code freeze.

[6] The P4 protocol: https://p4.org
[7] The eBPF kernel extender: https://ebpf.io/

4 Distributed Ledger and Smart Contracts

This section introduces all aspects related to the Distributed Ledger Technology (DLT) component used within the TeraFlow project, more specifically in T4.2.

DLT allows a set of nodes to become a distributed database by sharing information in a verifiable and transparent manner and recording the data to be stored using a consensus mechanism. Blockchain [7] is the best known example of a DLT. Blockchain is not only a distributed data base, but it also allows the execution of code (i.e., smart contracts) allowing the peers to work together without the need of a central element commanding any action. Blockchain is being used in multiple research SDN aspects, such as the security of flow management [8] or the recovery of SDN nodes after a failure [9].

The use of Blockchains replaces centralized network management consisting of conventional database management systems. Major advantages are the elimination of trusted third-parties that maintain the databases with single points of failure, and data provenance including data immutability and traceability.

4.1 Permissioned Distributed Ledger and Smart Contracts

TeraFlow will deliver a permissioned distributed ledger (PDL) that utilizes Blockchains for network management. A PDL allows pre-selected participants, such as TeraFlow partners and stakeholders, to record and validate transactions. This model is suited to industry consortia where the identity of the participants is known. It will be privacy-aware as well as transparent, resulting in an open, traceable, and fair sharing of network resources and services between stakeholders.

Smart contracts provide a universal basis to automate, simplify, and secure network management tasks that involve possibly sensitive data from multiple stakeholders in the network. Trust and multi-tenancy are improved in the SDN controllers by introducing novel security mechanisms through the use of smart contracts and secure consensus algorithms. Network and device data is not stored and processed in a central location; instead, a Blockchain stores the data and the operations across multiple nodes, and each node updates its Blockchain to reflect a requested change, often by executing a smart contract.

4.1.1 Design Overview

In the scope of the TeraFlow project, DLT is planned to be used in the multi-domain scenario as presented in Figure 9. Multiple TeraFlow OS domains are part of a Blockchain network (see grey cloud in Figure 9). A specific module within the TeraFlow architecture has been designed to let a TeraFlow OS node become a peer in a Blockchain (see yellow circles in Figure 9). This module is called the DLT component and is responsible for dealing with any action related to the Blockchain. The key objective of the Blockchain network is:

- To provide a trustworthy and resilient platform for storing, querying, and processing critical data about network resources and services owned and governed by different network entities.
- To record information such as software status (e.g., software/firmware version) and runtime information (e.g., remote attestation, tamper detection). This use case will enhance device and component security, enable tamper detection, and ensure the independent verification of device status, history and details.

- To allow the collaboration among multiple TeraFlow OS nodes by sharing the SDN resources available in their transport network infrastructures.

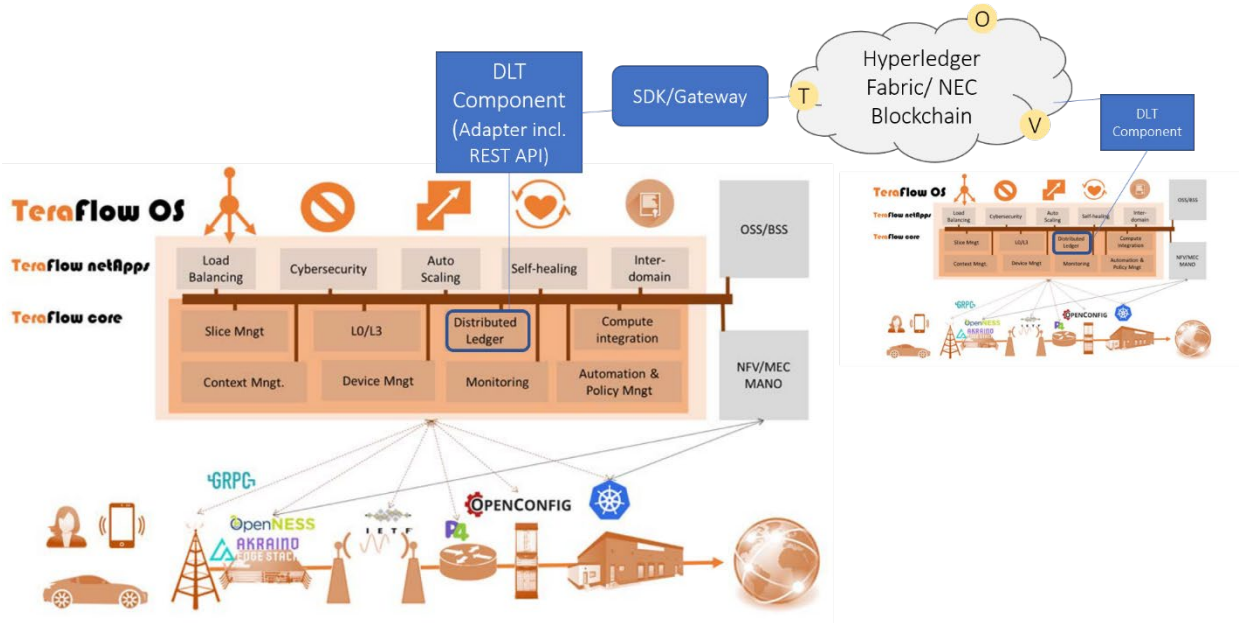

Figure 9: TeraFlow Multi-domain scenario interacting across Blockchain.

Another key aspect to accomplish the previous objectives is the use of Smart Contracts (SCs). An SC is a piece of executable code with a set of rules/actions that any peer within the Blockchain can use to manage the information within the Blockchain. Each Smart Contract has a specific identification which the peers must know in order to call its functions.

4.1.2 Interfaces

Hyperledger Fabric has been chosen as the permissioned Blockchain of choice. Hyperledger Fabric is an open-source enterprise-grade permissioned DLT platform, designed for use in industrial contexts that delivers some key differentiating capabilities over other popular distributed ledger or Blockchain platforms.

Hyperledger Fabric has been specifically architected to have a modular architecture. At a high level and as illustrated in Figure 10, Hyperledger Fabric is comprised of the following modular components:

Hyperledger Fabric Architecture

- **Chaincode/Smart Contracts:** Smart contracts ("chaincode") run within a container environment (e.g., Docker) for isolation. They can be written in standard programming languages, but do not have direct access to the ledger state.
- **Peers**: A Blockchain network is comprised primarily of a set of peer nodes (or, simply, peers). Peers are fundamental elements of the network because they host ledgers and smart contracts.
- **Orderer**: A pluggable ordering service establishes consensus on the order of transactions and then broadcasts blocks to peers.
- **Certificate Authority**: Peers have an identity assigned to them via a digital certificate from a particular certificate authority.

Interface to TeraFlow Domains

The Blockchain built on Hyperledger Fabric interfaces with the TeraFlow domains using the following set-up:

- **Fabric SDK/Gateway:** The Fabric Gateway SDK allows applications to interact with a Fabric Blockchain network and connects the DLT component and the Blockchain network
- **DLT Component:** This is the key component that facilitates the interaction between the Blockchain and the TeraFlow domains. It provides an API to submit transactions to a ledger or query the contents of a ledger.
    - The DLT component consists of the DLT Adapter based on a REST API. The details of this API and its integration are described in the Preliminary Results section, 4.1.3.1.

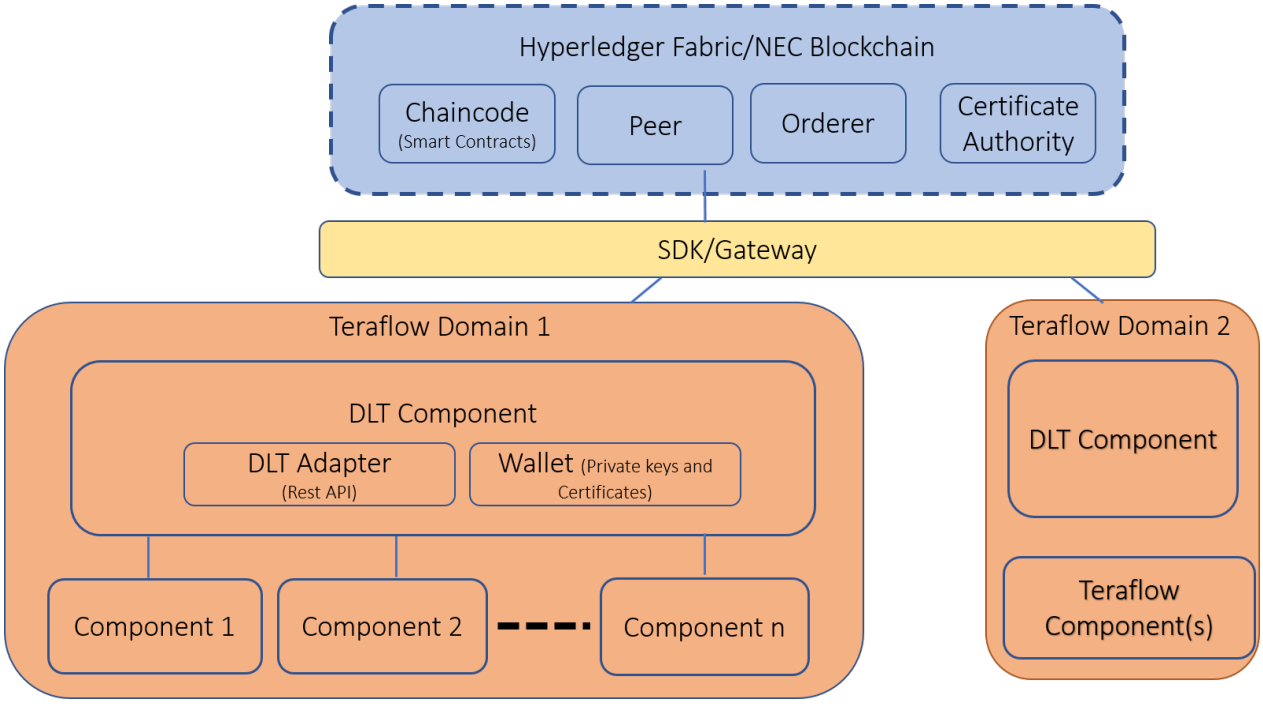

Figure 10: DLT component internal and multi-domain architectures..

Figure 11 illustrates how the DLT component interacts with the other TeraFlow components and how multiple TeraFlow domains interact with the individual DLT components. The following methods are used to interact with the DLT component:

- RecordToDlt is used to record and register information on the DLT/ Blockchain
- GetDltStatus is used to retrieve information registered and recorded on the Blockchain

The DLT records are shared among all the DLT components.

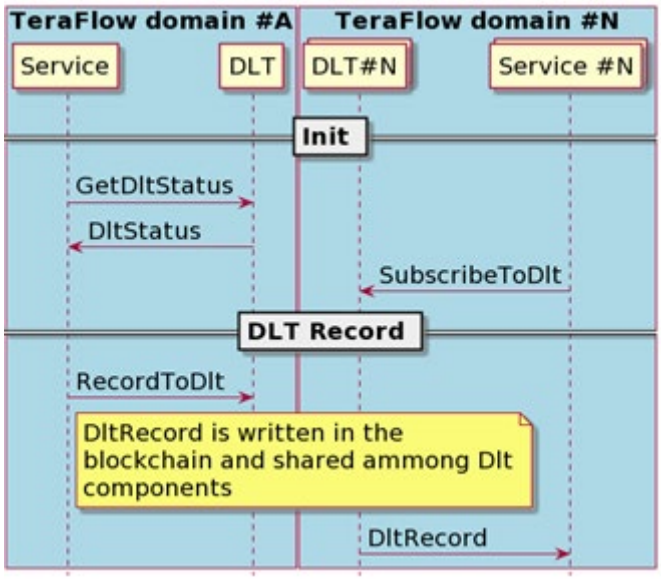

Figure 11: DLT Component workflow

Figure 12 shows the DLT component data model which is used to record, register, and retrieve information.

DLT Component Data Model

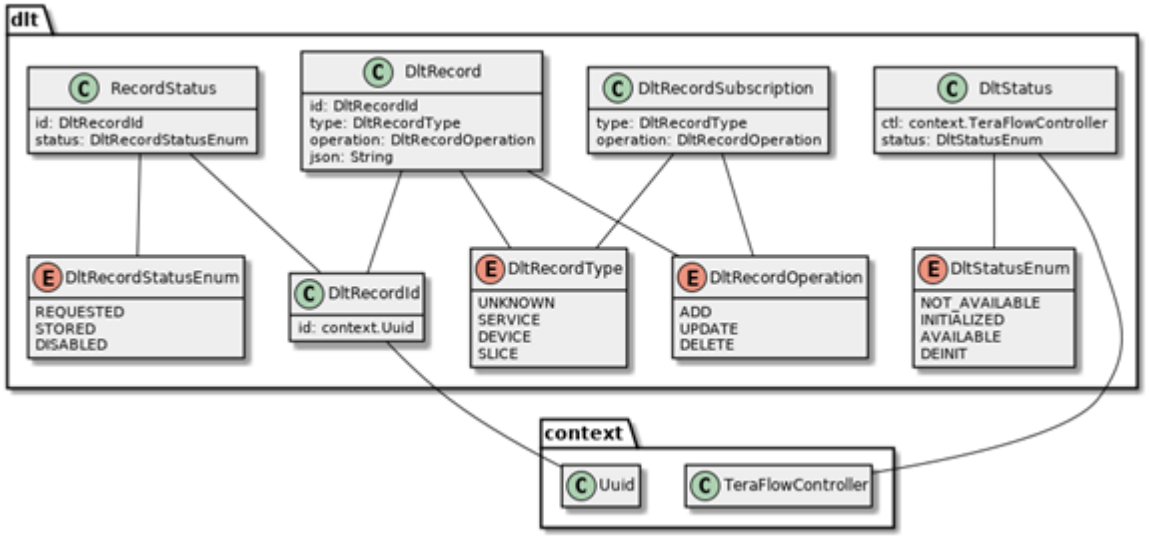

Figure 12: DLT component data model

4.1.3 Preliminary Results

The following subsections present the preliminary results obtained up to the date of this report. Both the design of the DLT module to be implemented within the TeraFlow OS, and the first single component results are shown. This demonstrates how Blockchain is used in the management of transport connectivity resources.

4.1.3.1 DLT Module Integration

As described in Section 4.1.2, the interfaces between the Blockchain network and the TeraFlow components in the DLT module are shown in Figure 10. Here we provide an overview of the DLT component module, its structure and software packages. The integration of the DLT component is documented in more detail in Milestone 4.1 (MS4.1).

DLT Component Module:

The DLT module is used to provide access to the underlying Fabric deployment. It allows clients to add, retrieve, modify, and delete Blockchain-backed data, essentially working as a key-value database. External clients should use the REST API to communicate with this service. Its detailed description available below.

DLT Module Structure

The whole DLT module consists of several packages:

- Fabric package

  The most important class in this package is FabricConnector. First, it establishes connection with the underlying Fabric network using the Java Gateway SDK. After that, it could be used as a CRUD interface. Other files contain auxiliary code for FabricConnector which allows it to register/enrol users and to obtain smart contract instances.

- HTTP package

  Contains the server-side HTTP handler. It accepts requests from the outside and performs the requested operations. For a detailed description, see section 4.1.3.2.

- Proto package

  The proto package contains a Config.proto file which contains messages for the REST API. The most important ones are DltConfig (defines the whole DLT configuration) and DltRecord (represents data to store in the Blockchain).

- Client example

  This code is not necessary to the service, but it could be used to test the service. It contains a sample REST client which connects to the service and performs all the CRUD operations.

REST API Description

Table 2 describes the REST API Methods, URL details, and HTTP Response codes.

Table 2 HTTP response codes for REST API

Method	URL	Input Data Objects	HTTP Response Code	Output Data Object
POST	/dlt/configure	Configuration object	201 – Peer configuration done 400 – Configuration not ready	Status value
GET	/dlt/configure	-------	201 – Status retrieved 404 – Peer not found	Peer status object

POST	/dlt/record/	Record object	201 - Record created 400 – Record not created 403 – Peer not known in the DLT	Record status object
GET	/dlt/record/	UUID value	200 – Record retrieved 404 – Record not found	Record object

4.1.3.2 Blockchain-Based Connectivity Provisioning in Multiple SDN Domains

The work done in WP4 up to now aims to present the initial tasks regarding a Blockchain-based SDN architecture to allow different transport domains to collaborate and to avoid any dependence on an E2E SDN controller. With respect to the tasks presented in [10], [11] and [12], the latest results were focused on the idea of having a collaborative SDN architecture and to validate how a set of SDN domains may interact with each other to compute and deploy E2E transport Connectivity Services (CSs).

To allow SDN controllers to join the Blockchain system, a new module called the PDL-Transport Manager was designed, allowing an SDN Transport controller to become a Blockchain peer (called PDL_SDN). Figure 13 showing multiple SDN domains as peers of a Blockchain system. Each peer (called PDL-SDN) is composed of an SDN Controller and a PDL-Transport Manager module. The PDL-Transport Manager takes assigned events from the Blockchain requests (called transactions) generated by other peers and maps them into ONF Transport API (T-API) [13] requests for the underlying Transport SDN Controller. The use of T-API was selected as it allows the deployment of per-domain CS to configure an E2E transport connection (from now on E2E CS). A T-API CS request allows the configuration of transport connections between a pair of client ports known as Service Interface Points (SIPs) of an SDN transport domain.

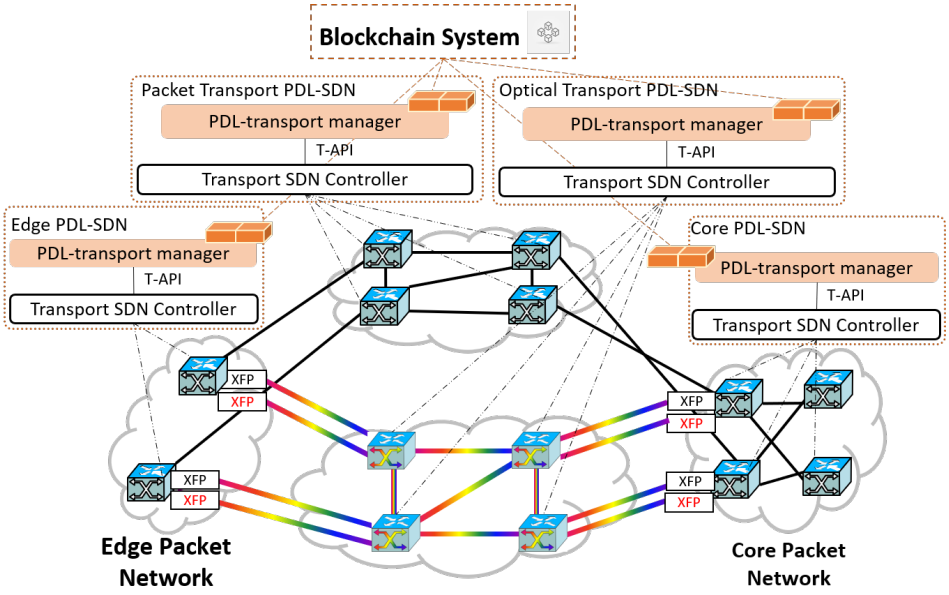

Figure 13: SDN Transport Blockchain-based infrastructure example.

To better understand the rest of these preliminary results, it is important to clarify that the PDL-Transport Manager element in Figure 13 should be understood as the first design of the DLT component within the TeraFlow OS, and that each PDL-SDN should be considered as an instance of a

TeraFlow OS but with only DLT capabilities. We are currently working to integrate the work done up to now with the other TeraFlow OS Security components.

The use of Blockchain brings the following advantages:

a) Any request for networking resources is public, transparent, and immutable once done, which makes it highly difficult to tamper it.
b) The avoidance of a hierarchical E2E architecture and so, there is no central point of failure that may block E2E actions.
c) When a peer joins the Blockchain network, its information is dynamically added, and the other peers update their vision of the whole infrastructure.
d) As there is no hierarchy and all the peers are equally important, if a peer becomes unavailable, the others can still work together.
e) The way the architecture is designed, only one transport SDN domain can configure each domain CS because its creation request is linked to a unique Blockchain address identifier.

On the other hand, the current architecture has some drawbacks:

a) A domain that is not available may be included in a path computation because the domain computing the path does not have updated infrastructure information.
b) Blockchain is designed to avoid unfinished transactions. To achieve this, it makes use of an associated cost per transaction, which means that any transaction must be generated with precision and security if it is to be accomplished.
c) Due to the use of costs, the information in a transaction must be as precise as possible and avoid redundancies.

Despite their importance, these drawbacks can be solved by checking the availability of each domain once the path has been computed, or by improving the design of the requests to select the essential information.

Before any CS can be requested, when a transport SDN Controller domain joins the Blockchain network, it must distribute its SDN T-API context to the other Blockchain peers. The minimum information in a T-API context is a set of Service Interface Points (SIPs) which are used by an optical SDN controller to request CSs. Furthermore, a T-API context may also define the topology of the network domain with a real or abstract vision. In both cases the topology is defined by nodes and links. Nodes include a set of node ports called Node Edge Points (NEPs), and the links are defined by the pair of NEPs they connect. So, a SIP is associated to a NEP at the edge of a network domain.

As presented in Figure 14, the process begins when the domain Operations and Business Support Systems (OSS/BSS) specifies (1) to the PDL-Transport Manager the context to share. The PDL-Transport Manager gets (2) the context information and with the response (3) from the Transport SDN Controller, it selects (4) the necessary parameters for the Blockchain. Then, it starts a transaction to distribute the information (5,7,9) and each domain updates their vision of the whole infrastructure (6,8,10). Finally, the original requester waits for the transactions to be accepted (11) and the PDL-Transport Manager to inform about the correct context distribution (12).

To be aware of the E2E topology, each SDN domain has a graph generated with the shared information. Each SDN domain is mapped as a node and the edge NEPs are related a local SIP and a remote SIP from another transport SDN domain. For each E2E CS requested, a domain CS list is computed using domain selection.

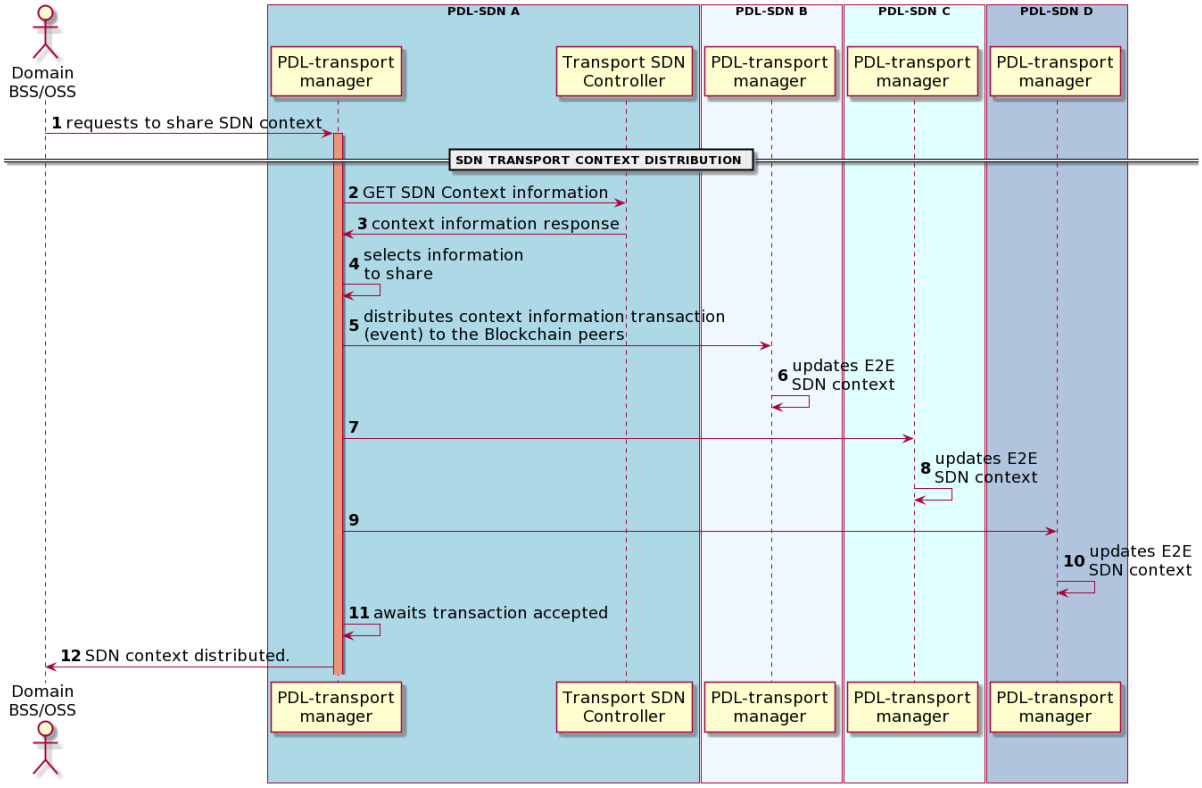

Figure 14: Transport SDN context and topology distribution.

Figure 15 presents the procedure to create a Blockchain-based E2E CS across transport SDN domains. The process starts when an E2E CS is requested by a domain OSS/BSS to its PDL-Transport Manager (1). A set of domain CSs is computed based on the graph generated when the transport SDN domains join the Blockchain network. Currently, to select the domains involved in the E2E CS, a shortest path algorithm is used. So, starting from one end of the path, and checking one-by-one the SDN domains along the path, the PDL-Transport Manager uses the shared context information to select the right pair of SIPs (i.e., source and destination) and to request each one of the domain CSs by distributing their information in the Blockchain. If the domain CS must be done in the local domain, the request is forwarded (2) to the local Transport SDN Controller which configures the domain CS (3) and replies (4). If, on the other hand, the domain CS must be done in another SDN domain, the PDL-Transport Manager generates a transaction and distributes it (5, 6, 7) specifying the address of the peer in charge.

The specified peer is the only one able to take the request and forward it to its local Transport SDN Controller (8) and send a notification of its acceptance (10). Meanwhile, the domain CS is configured (9) and its information sent back (11) to its domain PDL-Transport Manager. Then, the updated domain CS information is distributed through a new transaction (12,13,14) with the peer address that originally requested the domain CS. Once the transaction is distributed and accepted (15) and if all the domain CSs are ready, the PDL-Transport Manager informs the OSS/BSS about the complete configuration of the E2E CS.

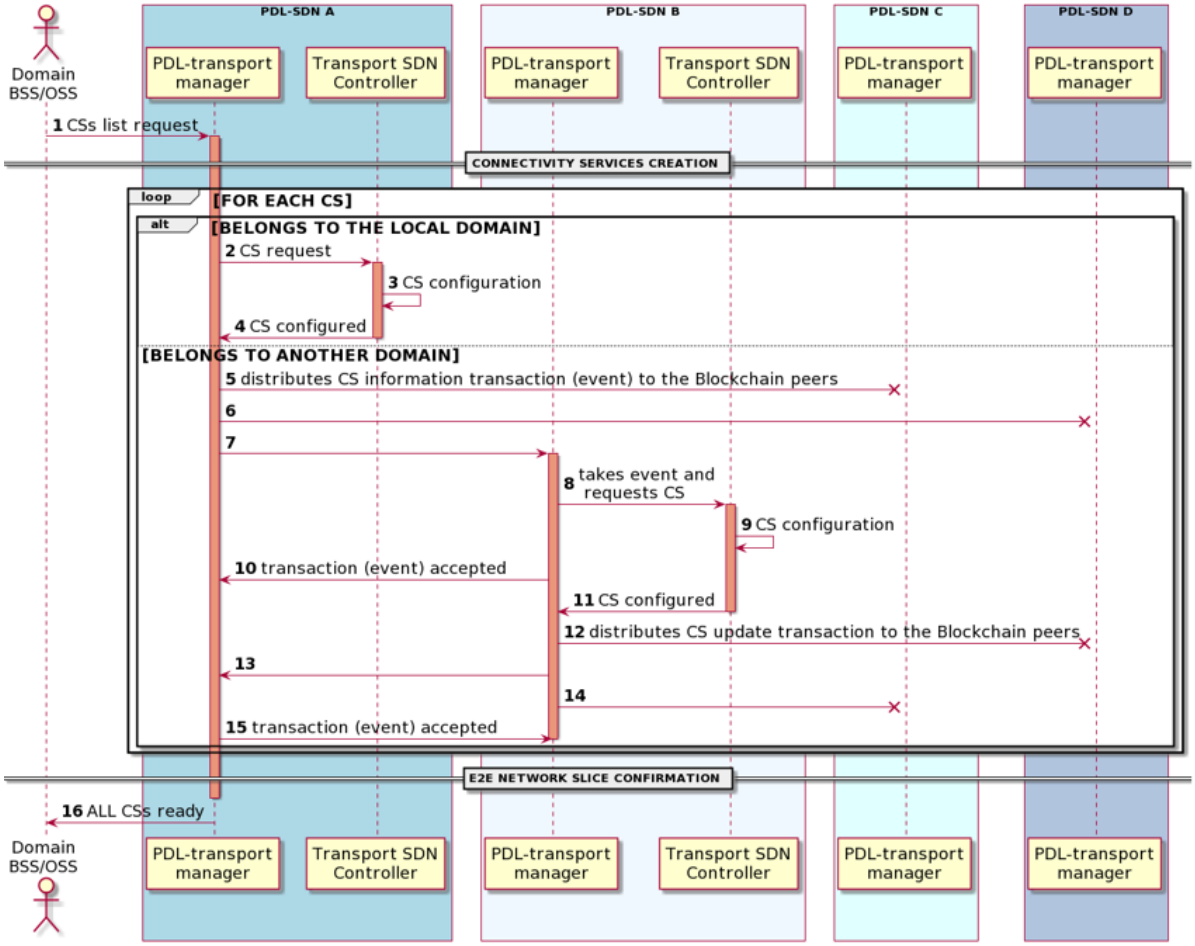

Figure 15: Blockchain-based Transport CS deployment

The solution described has been implemented as presented using the ADRENALINE [14] testbed with four different SDN domains: an edge, a transport, a core packet-based domain, and an optical-based transport domain. Each domain with an SDN controller managing the incoming domain CSs requests. Finally, over each domain SDN controller, there is the PDL-Transport Manager. To validate the proposed solution, an experimental evaluation was done by sharing the context of three different domains (edge, transport, and core) and requesting a bidirectional CS (i.e., two unidirectional CSs) between the edge and core domains. Finally, for these initial experimental validations, the Blockchain system was implemented using an Ethereum emulator called Ganache [15].

Figure 16 and Figure 17 show the HTTP requests and the transactions generated among the Blockchain peers in the setup and deployment procedures previously described. Figure 16-A shows the workflow to distribute the context of each SDN domain with three HTTP requests to share the context of each SDN domain. Once each HTTP request reaches the corresponding PDL-Transport Manager, a Blockchain transaction is generated and distributed as demonstrated in Figure 17-A.

Figure 16-B shows the deployment of CSs with the HTTP request that triggers the whole process (i.e., step 1 in Figure 15) and the HTTP requests to create the different CSs (step 8 in Figure 15) between the PDL-Transport Manager and its associated Transport SDN Controller. The distribution of the CS requests across the Blockchain network is presented with the two pairs of transactions logged in Figure 17-B/C. Each transaction log belongs to the CS creation transaction distributions (steps 5, 6, 7 in Figure 15) and the CS update transaction distributions (steps 12, 13, 14 in Figure 15).

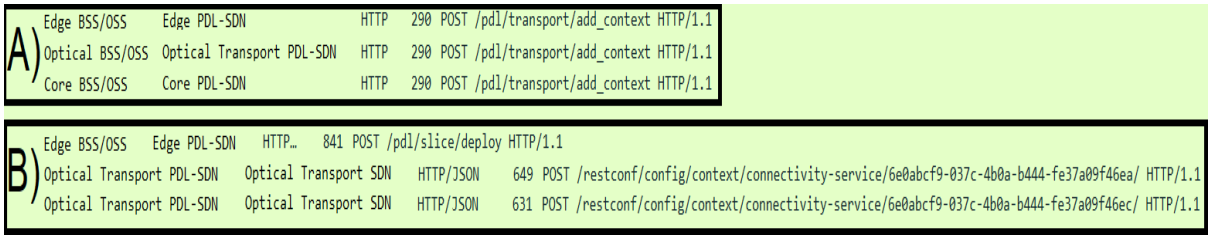

Figure 16: Context distribution and CSs deployment HTTP requests

Figure 17: Blockchain transactions log

Table 3 presents the mean and standard deviation values for each E2E CS deployment, their total deployment time, and the total time associated to the different Blockchain transactions (i.e., CS creation and update). A complete E2E CS configuration requires around 2 seconds, giving a total mean time value of 4.08 seconds to create the two unidirectional domain CSs (CS1 and CS2 in Table 3). The Blockchain transaction mean time value is of 2.34 seconds, which adds an increment of 50%.

Table 3 CS deployment time vs related Blockchain transactions time

	Time (s)			
	Connectivity Services Deployment			Blockchain Transactions
	CS 1	CS 2	Total	
Mean Value	2.01	2.07	4.08	2.34
Std. Deviation	0.11	0.22	0.19	0.49

Despite the fact that this time increment is significant, compared to possible SDN situations such as a reconfiguration of optical amplifiers that may take minutes, it becomes less significant. Based on this, the trade-off to implement this new architecture might be an increment of seconds per each CS creation to keep the Blockchain advantages.

5 Interworking Across Beyond 5G Networks

The interactions of the TeraFlow OS with external entities is explored in this section dealing with the T4.3 objectives. In a nutshell, this entails the interworking of the deployed TeraFlow OS with other control/orchestration elements to offer network connectivity services for two objectives: i) different domains (i.e., inter-domain connections), and ii) deploying network services involving both cloud and network resources. In the former case, the adopted approach relies on enabling the interaction between peer TeraFlow OS instances that manage their own network domains. To this end, within the TeraFlow OS architecture, a dedicated entity referred to as Inter-domain component (IDC) communicates with other IDCs in remote TeraFlow OSs with the purpose of rolling out network services traversing two or more domains. On the other hand, the second objective is to allow the interaction of the TeraFlow OS with an NFV Orchestrator. The NFV Orchestrator handles the deployment of network services which in general are made up of a set of compute-deployed functions, i.e., virtual network functions (VNFs)/containerized network functions (CNFs), and virtual links (VLs). The VLs determine the connectivity to be achieved between those VNFs/CNFs in the network service. Thus, a network service in the NFV Orchestrator specifies a set of compute and networking resources with specific requirements (e.g., CPU, storage, memory, guaranteed bandwidth, maximum latency, etc.) to be accommodated on top of a common NFV infrastructure formed by distributed cloud premises (e.g., datacentres) and a transport network. Consequently, if the NFV Orchestrator places the VNFs/CNFs of a specific network service in remote cloud premises, the VLs interconnecting such functions need to be deployed over the transport network. The deployment of these VLs is delegated by the NFV Orchestrator to the TeraFlow OS. In other words, the NFV Orchestrator requests the TeraFlow OS to deploy connectivity services to support the deployment of VLs forming the network services being rolled out. The following subsections provide the details of these two TeraFlow OS capabilities to enable the interaction with external entities.

5.1 Compute Component

This section tackles the designed interactions between an externally-selected NFV Orchestrator and the TeraFlow OS at the time of automatically deploying network services embracing both compute (e.g., VNFs or CNFs) and networking (i.e., wide-area network) resources. Specifically, the NFV Orchestrator and the TeraFlow OS interact according to a client-server relationship. That is, the NFV Orchestrator (acting as a client) first considers incoming network services demands. After placing the VNFs/CNFs in a distributed pool of datacentres / cloud premises, it obtains the network connectivity between such remote and distributed compute premises from the TeraFlow OS (acting as a server). The requested connectivity service may specify heterogeneous network requirements in terms of minimum guaranteed bandwidth, maximum tolerated latency, ensured reliability, etc. The following focuses on describing the preliminary design/setup enabling the interaction between the Open-Source Management and Orchestration (OSM) NFV Orchestrator [1] and the Compute component element deployed within the TeraFlow OS.

5.1.1 Design Overview

Figure 18 illustrates the basic architectural aspects of the Compute component within the TeraFlow OS. The Compute component as, previously mentioned, operates as a front-end for the NFV Orchestrator (e.g., OSM release). The communication between them is done via a northbound interface implemented over a defined REST API with JSON encoding. This interface allows the NFV

Orchestrator to handle the lifecycle management of the network connectivity services between remote datacentres. The operations covered are:

- Get connectivity service status (*get_connectivty_service_status* via GET method)
- Create connectivity service (*create_connectivity_service* via POST method)
- Delete connectivity service (*delete_connectivity_service* via DELETE method)
- Edit (update) connectivity service (*edit_connectivity_service* via PUT method)
- Clear all connectivity services (*delete_all_connectivity_service* via DELETE method).

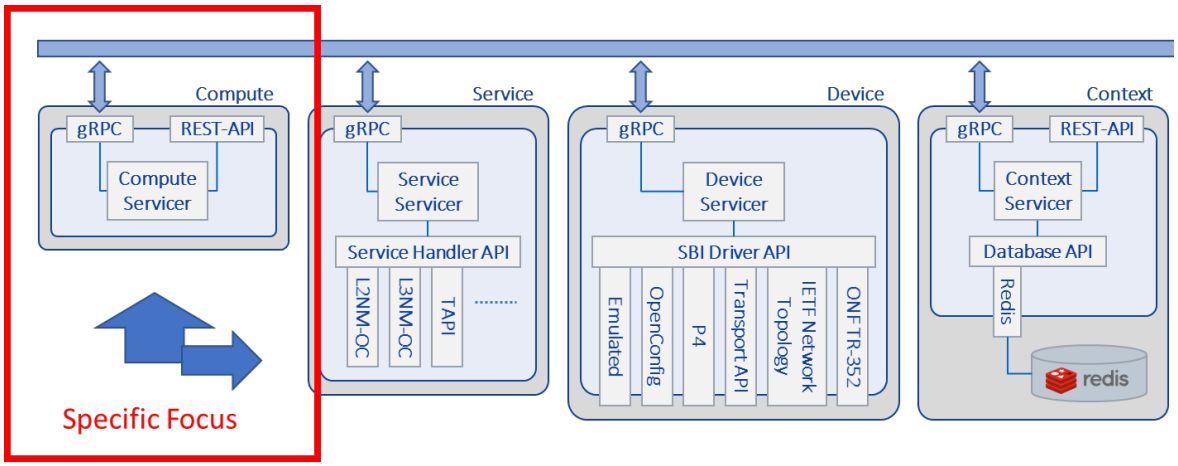

Figure 18: TeraFlow OS Compute component

The macroscopic objective of the Compute component is to process the listed incoming operations from the NFV Orchestrator and conduct a translation/mapping function relying on gRPC towards the Service component, also within the TeraFlow OS scope. Such a Service component would be responsible for completing the required operations and responding to the Compute component with the resulting output.

Considering the above, a very basic scenario encompassing the interactions between the NFV Orchestrator (OSM), Compute and Service components is depicted in Figure 19.

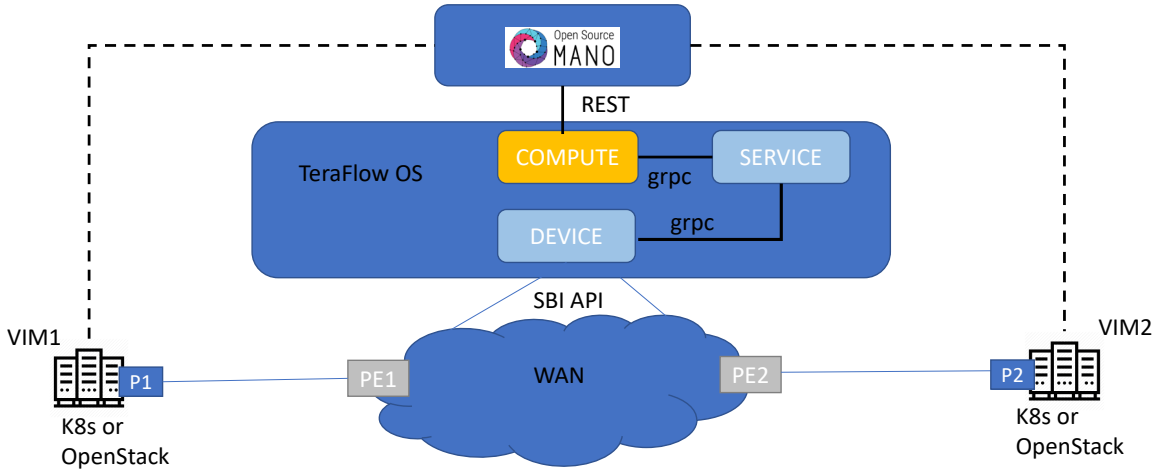

Figure 19: OSM, Compute, and Service components interworking

In this scenario, the underlying Network Function Virtualization Infrastructure (NFVI) is made up of two distributed datacentres (referred to as Virtualized Infrastructure Manager 1 and 2 - VIM1 and VIM2) interconnected through a wide-area transport network (i.e., WAN). Both datacentres are

controlled by their own compute controllers (e.g., OpenStack or K8s) coordinated by the OSM. The WAN infrastructure is controlled by the TeraFlow OS acting as a WAN Infrastructure Manager (WIM). The WAN is assumed to encompass diverse transport switching technologies such as packet, optical, etc.

The objective is that once the VNFs/CNFs are deployed in the datacentres by the OSM, communication with the Compute component is established to demand connectivity between the datacentre endpoints, i.e., P1 and P2. This description considers the network connectivity service is requested as a layer 2 VPN (L2VPN) using VLAN tagging. The interconnection between the VIMs' ports and the WAN edges (i.e., PE1 and PE1) should be explicitly determined within the OSM Resource Orchestrator module. This information is formed of:

- *Datacentre name*: VIM1
- *WAN service endpoint identifier*: PE1 (using UUID)
- *Site id*: P1 (using UUID)
- *Bearer reference*: P1 (using UUID)

The requested operation sent from the OSM entity to the Compute component is handled locally by the Compute component to map the received operations/commands into those supported between the TeraFlow OS Integration components: the Compute and Service components. Thus, implicitly, the Compute component makes a translation of the received OSM REST API-based contents into those supported by the gRPC messages exchanged between the Compute and Service components. The following table reflects this mapping:

Table 4 Mapping between OSM-Compute component API and Compute-Service component APIs

OSM – Compute Component (REST API)	Compute – Service (gRPC)
get_connectivty_service_status	*GetServiceList // GetServiceById*
create_connectivity_service	*createService*
delete_connectivity_service	*DeleteService*
edit_connectivity_service	*UpdateService*

5.1.2 Interfaces

Figure 20 describes the workflow between the NFV Orchestrator and the TeraFlow OS Compute component to create a layer 2 network connectivity service. Other supported commands over this API (e.g., deletion or update of active network services) are not discussed. Thus, focusing on the creation of the network service, this entails 6 steps as shown in Figure 20. Step 1) is bound to the request sent by the NFV Orchestrator (OSM) via a POST message to trigger the connectivity service (i.e., *create_connectivity_service*()). This is referred to as the creation of the *vpn service*. Such a POST message carries a set of required JSON-encoded objects, namely, the *vpn_id* (which contains a UUID), the *vpn_svc_type* (set to virtual private wire service, vpws), the *svc_topo* (set to any-to-any), and the *customer-name* (carrying OSM). After processing this message, the Compute component creates an entry for that network connectivity service. To unambiguously refer to that entry, a service identifier (serviceid) is created and set to the received *vpn_id.* Next, the Compute component responds to the NFV Orchestrator that the registration of the vpn service succeeded (i.e., step 2).

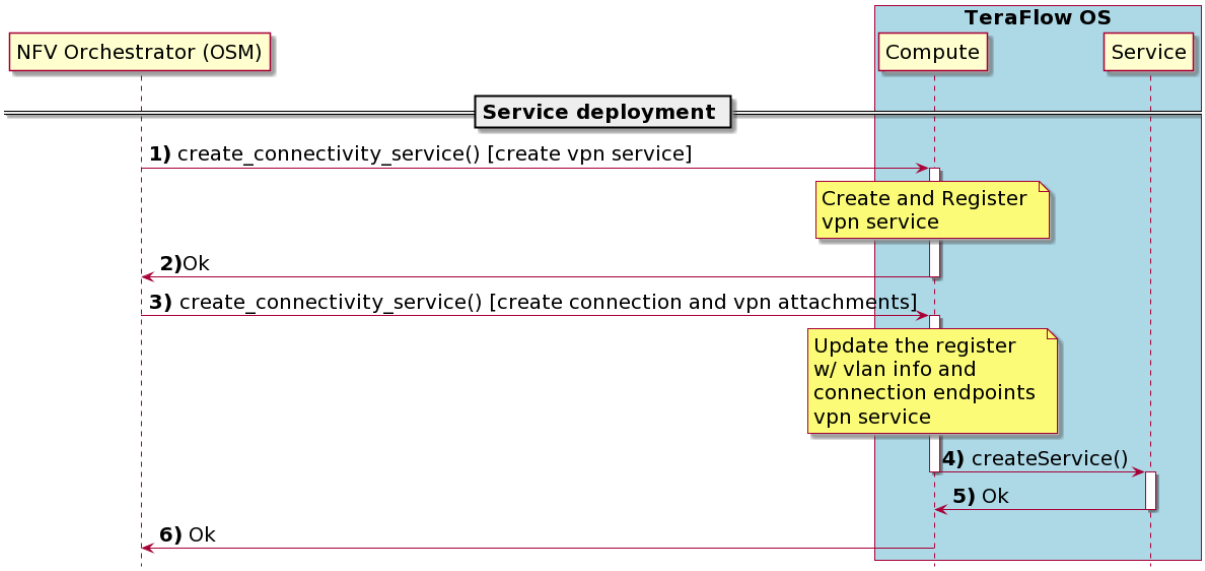

Figure 20: NFV Orchestrator (OSM) – TeraFlow OS compute workflow

Next, the OSM sends another POST message in step 3) with the required information about the connection service and its vpn attachment endpoints. This entails specifying the *connection_point* (i.e., vlan), and the endpoints using the *bearer* (e.g., UUID of P1 in Figure 19) and the *site* (e.g., UUID of P1 in Figure 19). These network connectivity service details are then updated at the Compute component and used to construct the gRPC createService message (step 4). Once the network connectivity service is successfully established by other TeraFlow OS components (step 5), the Compute component responds to the NFV Orchestrator informing that the network connectivity service is active (step 6).

5.1.3 Preliminary Results

The preliminary results of the deployment of the Compute component of the TeraFlow OS aim at validating the creation of a connectivity service demanded by the NFV Orchestrator (OSM) supporting a specific network service instantiated between two remote VIMs, as depicted in Figure 19. To this end, it is required that the NFV Orchestrator (OSM) identifies the Compute component as the element to steer the connectivity service demands. In other words, the Compute component acts as the WIM element for receiving, processing, and deploying the arriving connectivity service requests. Figure 21 shows a screenshot of the GUI used by the OSM reflecting that the NFV Orchestrator identifies the underlying Compute component of the TeraFlow OS as the corresponding WIM.

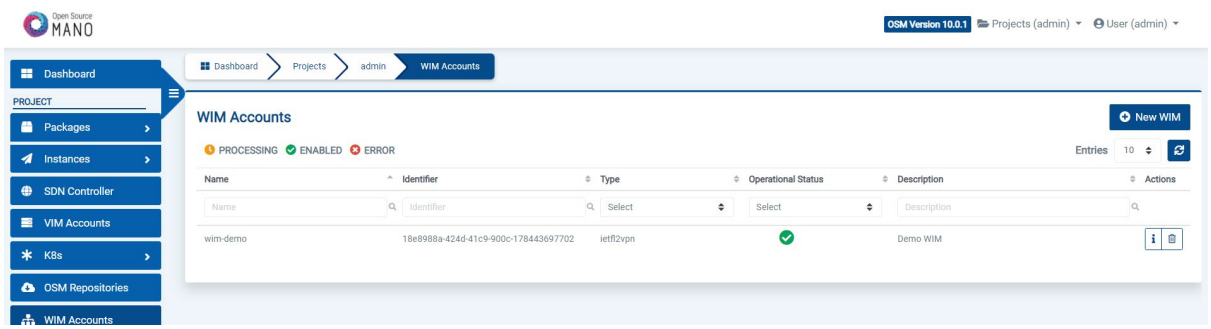

Figure 21: OSM GUI: identified and registered WIM

For the targeted scenario to conduct the validation, it is also needed that the NFV Orchestrator identifies (at least) two VIMs (i.e., VIM1 and VIM2) as the controllers to handle the cloud resource

allocation (i.e., VNFs of the network service). Figure 22 shows the GUI of the OSM reflecting the pair of registered VIMs. Recall that the aim is to allow VNFs deployed over such a pair of VIMs to be interconnected via a connectivity service handled by the WIM (i.e., Compute component of the TeraFlow OS).

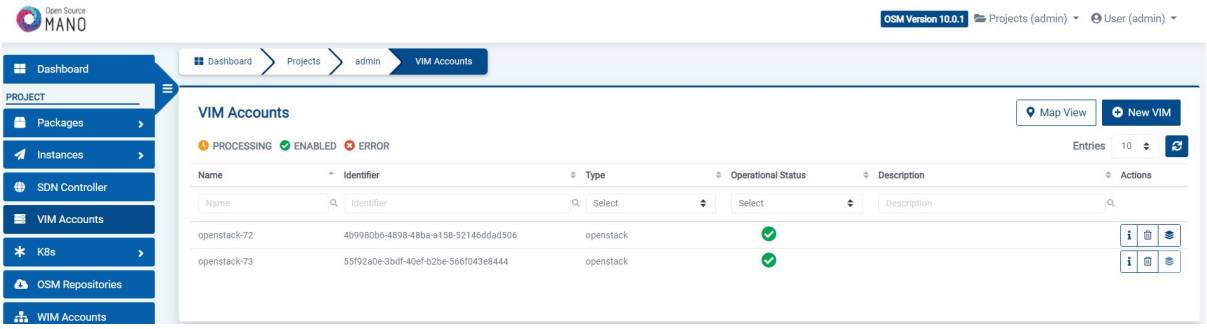

Figure 22: OSM GUI: identified and registered VIMs

At the time of writing, we are currently working on some supporting and backwards compatibility issues of the last OSM release 10 when triggering the creation of the connectivity service to the Compute component (i.e., relying on the REST API). In other words, the support of this function in the OSM is covered by previous releases, but not the latest one. We are in contact with the OSM developers to solve this problem.

5.2 Inter-domain Component

As defined in MS2.1, the main goals and responsibilities of the IDC include providing dedicated QoS-aware inter-domain connectivity services and enabling interaction between TeraFlow OS instances and peer TeraFlow OS instances which manage different network domains to create E2E TN slicing services. Based on these goals, this section presents the design of the IDC along with its main interaction components and interfaces. Furthermore, we provide an overview of requirements from the IDC towards other TeraFlow components as well as preliminary results in the context of a multi-domain Blockchain-based SDN controller architecture.

5.2.1 Design Overview

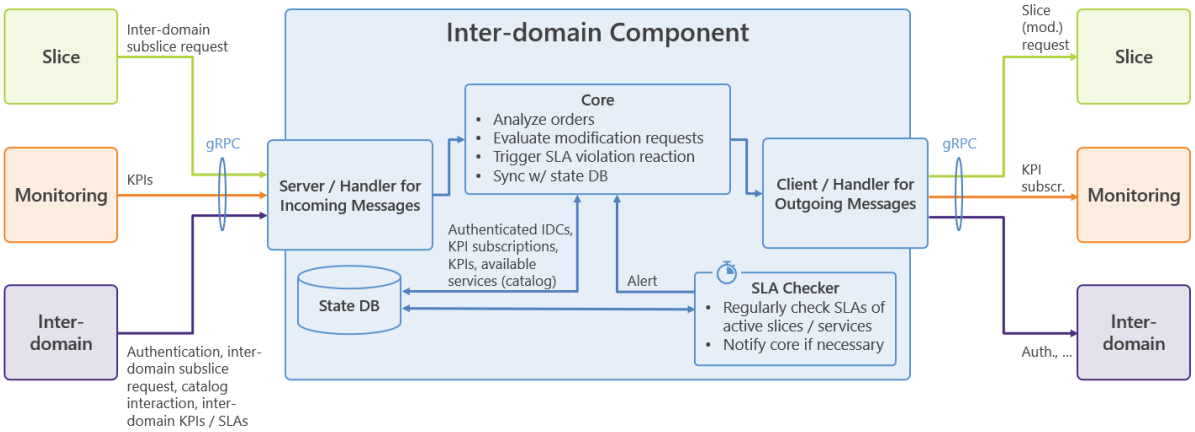

Figure 23: Design and architecture of the Inter-domain component.

The design of the IDC is based on three use case workflows which have been defined for the Inter-domain component in MS2.1. These workflows are: *service preparation and activation*, *service modification*, and *synchronization of service monitoring data between domains*. We have identified and amended interactions that involve the IDC and provide an overview of the resulting IDC architecture in Figure 23..

While external TeraFlow components are shown on the left and right hand side of the graphic, the IDC and its internals are located in the centre. We have identified three main interaction components for the IDC, namely the Slice component, the Monitoring component, and other Inter-domain components.

In order to achieve a separation of concerns within the IDC, we propose a modular design with the following sub-components:

- *Handlers for receiving and forwarding incoming and outgoing gRPC-based messages* act as the main interaction points with other TeraFlow components.
- A *state database* to keep track of authenticated remote Inter-domain components, active subscriptions to service KPIs alongside actual KPI values, as well as a service catalogue and inventory. In order to maintain a clear separation of concerns, the IDC does not perform active measurements or discovery, but the state DB merely represents a local cache that provides fast access to data related to KPI values, catalogue, and inventory. In particular, the latest and most reliable version of the data resides in the respective TeraFlow components, such as monitoring or slice management. For keeping the state DB up-to-date, two options are considered: (a) the IDC can behave like a proxy and forward requests towards the responsible component on demand, storing the most recent data; or (b) the IDC can keep a local copy and synchronize it regularly with the authoritative data source.
- The *Core module* constitutes the main decision making entity within the IDC. It is in charge of making decisions based on received messages related to service creation or modification requests, reacting to detected SLA violations, and keeping the state DB up to date.
- The *SLA Checking* module regularly queries monitoring information from the state DB to make sure that E2E SLAs are met, and alerts the Core module to trigger corrective actions in case of violations. While the graphic provides a high-level overview of the types of messages and information that are exchanged with external TeraFlow components, more details are provided in Section 5.2.2. In particular, it is worth noting that the IDC does not have to share detailed KPI and infrastructure information between domains; instead, it can transmit high-level, aggregated, and potentially anonymized information on SLA conformance status. Preliminary results regarding different options for sharing aggregated topology information are discussed in more depth in Section 5.2.4.

5.2.2 Interfaces

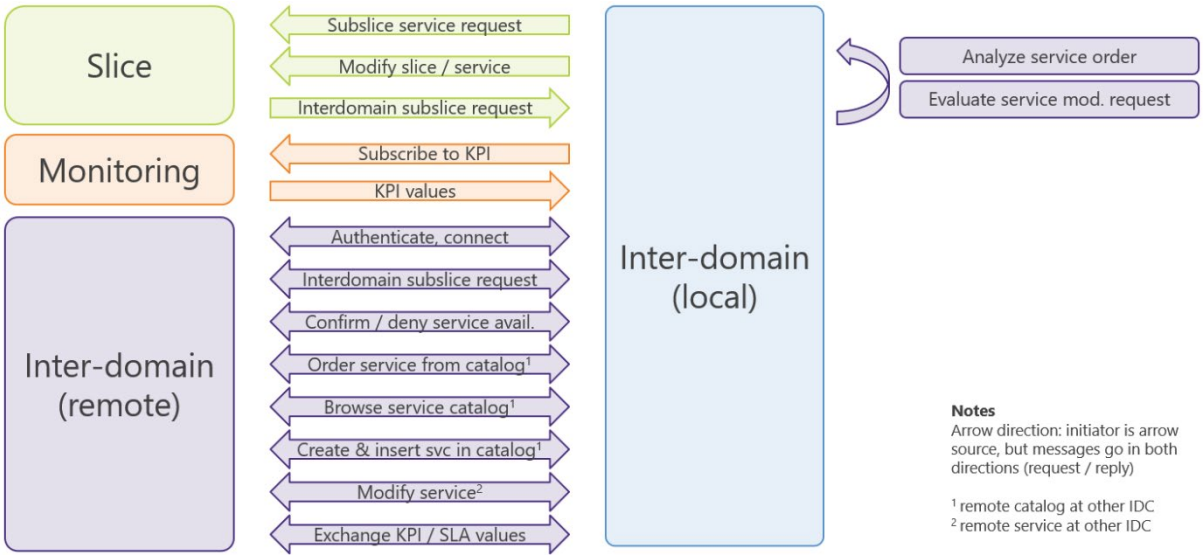

Figure 24: Main interfaces of the Inter-domain component

Given the design of the IDC and the identified communication partners, Figure 24 illustrates the interactions, and hence required interfaces, of the IDC towards other TeraFlow components. As outlined previously, the IDC primarily interacts with the Slice component, the Monitoring component, and other Inter-domain components. Each interaction is usually comprised of a request and reply, and is represented by an arrow in the graphic. In this context, the direction of the arrow indicates that the source of the arrow is the initiator of the communication. In the following, we briefly explain the functionality of the corresponding interfaces in their order of appearance in the graphic. Items marked with *[in]*, *[out]*, and *[in/out]* correspond to interactions that terminate at the IDC, originate at the IDC, and are exchanged between IDC instances, respectively.

- [out] Subslice service request: After receiving an inter-domain subslice request from a peer IDC, the IDC analyses the order and – if the order can be fulfilled – uses the corresponding interface of the Slice component to request a subslice.
- [out] Modify slice / service: Similarly, when a modification of an inter-domain E2E service is requested, the IDC uses the corresponding interface of the Slice component in the same domain to propagate the modification request.
- [in] Inter-domain subslice request: Since service orders are dispatched from the customer to the Slice component, the latter sends an inter-domain subslice request to the IDC if the requested service cannot be fulfilled within the original domain.
- [out] Subscribe to KPI: Using interfaces of the Monitoring component, the IDC subscribes to relevant service-level KPIs to enable E2E SLA assurance.
- [in] Send KPI values: Based on the aforementioned subscriptions, the Monitoring component periodically sends KPI values to the IDC, which stores them locally for the purpose of SLA assurance.
- [in/out] Authenticate, connect: Before exchanging any requests, two peer IDCs need to authenticate each other. By storing the authentication state and details of external IDCs in the state DB, authentication only needs to be performed once (on initial connectivity).
- [in/out] Inter-domain subslice request: If a service request spans multiple domains, the Slice component involves the IDC by sending it an inter-domain subslice request. In addition to sending an inter-domain subslice request to the local Slice component, the IDC also forwards

the inter-domain subslice request to the IDCs of other domains that are involved in the E2E service.

- [in/out] Interaction with the service catalogue and inventory: In the context of workflows related to the preparation and activation of an inter-domain service as well as the modification of services, interactions with the service catalogue (available service templates) and the service inventory (existing service instances) are necessary. These correspond to CRUD (create, read, update, delete) operations on the corresponding parts of the state DB which are exposed via interfaces as well as propagating the requests to the responsible components.
    - Confirm / deny service availability.
    - Order service from catalogue.
    - Browse service catalogue.
    - Create and insert service in catalogue.
    - Modify service.
- [in/out] Exchange KPI / SLA values: To make sure that service SLAs are met in an E2E fashion, the IDCs that are involved in providing such a service exchange KPIs with each other. Using this information, the IDC that is located in the domain from which the service request originated can trigger actions to mitigate potential SLA violations, e.g., by modifying the amount of resource that is allocated to the corresponding slices.
- Analyse service order, evaluate service modification request: These actions are handled internally by the IDC, but are listed for the sake of completeness.

5.2.3 Requirements Towards other TeraFlow Components

Based on the design and interfaces of the Inter-domain component, the following requirements towards other TeraFlow components have been identified. In addition to listing the requirements, we provide a brief discussion regarding their context and justification.

- Slice management.
    - The Slice Management component has a function to check whether a service request is intra- or inter-domain. Since the Slice component is the recipient of customer-initiated service requests, it needs to determine whether the IDC needs to be involved in serving a given request.
- Context, slice management, and service.
    - While the Context component has function for retrieving instantiated services (ListServices), the IDC will additionally obtain information about available slice and service templates (catalogue) as well as instantiated slices and services (inventory) along with their SLA settings from the Slice and Service components, respectively. This is in line with the general idea of separation of concerns and discovery not being a core responsibility of the IDC.
- Monitoring and policy.
    - The Monitoring component provides subscription to information on slice, service, and device KPIs.
    - The Monitoring component only provides metrics and alarms: it does not provide SLA violation events. An option would be for the Policy component to subscribe to appropriate metrics and trigger SLA violation events based on alarms and / or thresholds. While this is not strictly required by the IDC and could be achieved

internally by comparing received KPI values against SLA information in the state DB, the event-based approach could also benefit other components.

In addition, there are several concerns to be investigated in the next steps for WP4:

1. Which components are responsible for creating, storing, and maintaining (updating) the service catalogue and service inventory (parts of which will be exposed to the other IDC)? In general, one basic communication task across IDC instances is the exchange and browsing of the service catalogue and service inventory so one IDC can order a service from another IDC. The question is whether the IDC should create its own service catalogue and inventory or just retrieve the catalogue and inventory data when needed (e.g., when the other IDC instance requests service catalogue / inventory information). Considering the service catalogue and inventory are useful for other components (e.g., Slice, Service, and Context components), it may not be necessary for the IDC to create a DB to store the service catalogue and inventory. However, retrieving the service catalogue / inventory information may take extra time and require extra inter-component communications. The final decision should depend on several factors, e.g., how big is the service catalogue and inventory, how often the service catalogue / inventory will be exposed to other IDCs, and how often the catalogue / inventory will be updated.
2. Should the IDC store KPIs in its DB? In general, one IDC instance should not expose KPIs to the other IDC instances for two reasons. First, the KPIs collected for one service may be of large volume. Transmitting all KPIs between two IDCs would consume too much communication resource (on the control plane) and storage resource (in the IDC DB). Given that KPIs would have been stored in a common DB, it is not necessary for IDC to create a DB and store KPIs locally. Second, service KPIs are detailed and too specific to an operator's domain and should not be exposed to another operator's domain via another IDC instance. The information exchanged between IDCs should be kept as abstract and high-level as possible, without involving detailed operations, management, and behaviour.
3. What is the service model between IDCs? Following Question 2, there is a need to define a service model with a clear specification of what and how information should be communicated between two IDCs.
4. Whether and how is the service modification realized? Run-time service modification relies on service assurance or SLA assurance, which requires proper monitoring and analysis capabilities. The IDC does not possess these monitoring and analysis capabilities and relies on other components to provide/support such capabilities if Service Modification is expected.
5. What are the delay- and frequency-related considerations?: Delay for adding elements to and retrieving elements from the catalogue, as well as update frequency.

5.2.4 Preliminary Results

With the use of Blockchain, the inter-domain management may experience a new management model (i.e., distributed) in which all the transport domains collaborate with each other to deploy a set of (Domain) CSs that all together comprise an End-to-End (E2E) CS. To do this, as illustrated in Figure 25, each transport domain Optical Transport SDN Controller makes use of a module called the Blockchain Controller to become part of the peer-to-peer (P2P) network. Blockchain transactions are distributed among Blockchain Controllers with the necessary information to deploy the specific Domain CSs.

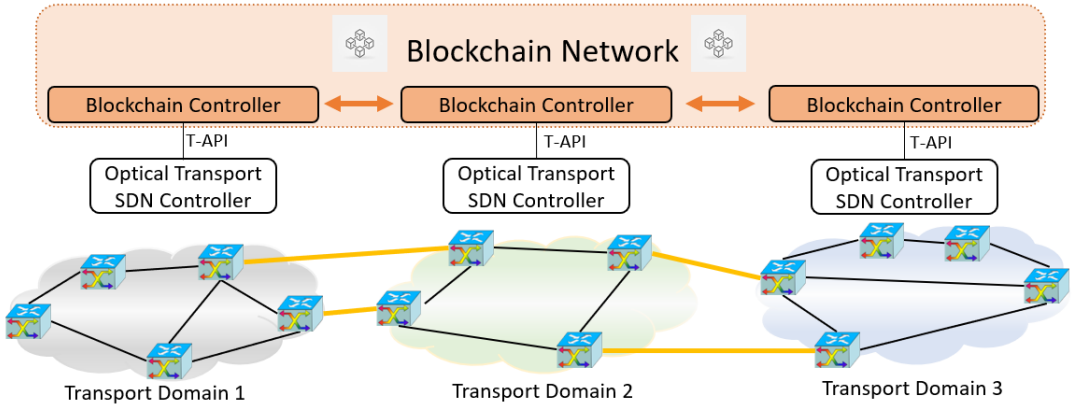

Figure 25: Multi-domain Blockchain-based architecture with intra-domain black) and inter-domain (yellow)

With the new module and with no central element taking care of the whole management, the different domains must exchange their SDN context information (i.e., their edge points and internal topology). Since the amount of data within an SDN context may vary, we made use of abstraction models [15]. Using abstraction models on SDN control infrastructures allows reduction of the amount of information that a control plane element sends to elements in remote domains. With the received abstract information, the receiver has an overview of its domain transport resources and retains the capability to request Domain CSs to the (Optical) Transport SDN Controller without the need to know every single detail of the physical infrastructure.

To experiment with different abstraction models, we selected the following three well-known options: transparent, Virtual Node (VNode), and Virtual Link (VLink) and a visual example of the use case implemented is presented in Figure 26.

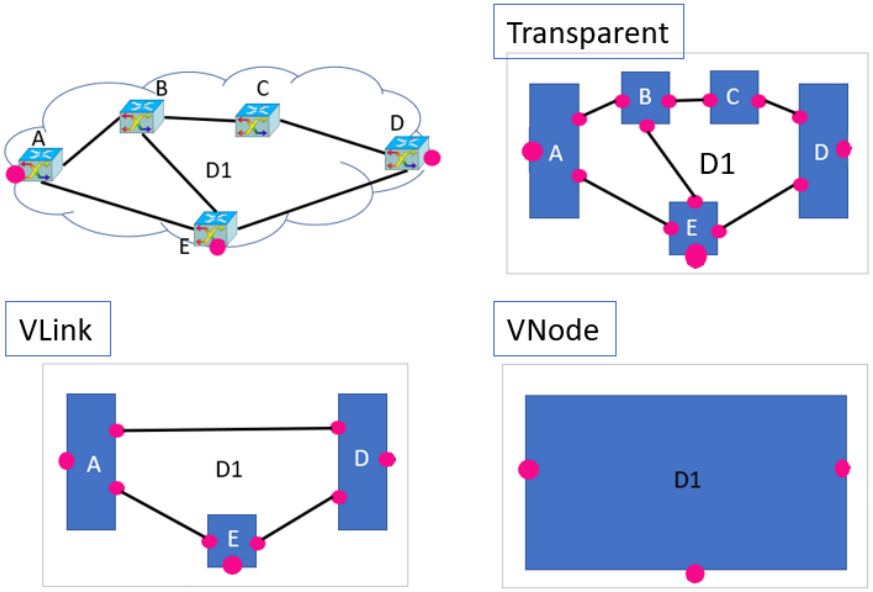

Figure 26: Example of a domain and its abstracted topologies.

The "transparent" model is an exact copy of the physical infrastructure. It contains all the details of all the domain nodes (i.e., its ports and bandwidth or spectrum resources) and intra-domain links. As presented in Figure 26 (top right), all the nodes (blue boxes) and their ports (pink circles), and the links between them (black lines) generate an exact copy of the original physical resources (Figure 26 top left).

While the transparent model is the most detailed of the three abstraction models selected, the VNode abstraction model is exactly the opposite. This is the simplest model in terms of the amount of abstracted information. In this model, the complete internal domain topology is omitted (physical nodes and intra-domain links) and only those node ports located in the edge domain are kept. The result is that each domain presents itself to the other domains as a single node as presented in Figure 26 (bottom right).

The basic idea behind the VLink model is to keep only a specific group from the real nodes and to interconnect the selected nodes in a virtual way. To do so, a set of "virtual" links are defined to interconnect the selected nodes among them. In our work, the condition implemented for the abstraction process was to select those nodes with one of their ports being an edge point of the domain. Using the example in Figure 26, the resulting abstraction (bottom left) had three nodes and three virtual links instead of the five nodes and six links of the original physical infrastructure. Due to the fact of defining the virtual links (which are not real, it is necessary to add a certain value, i.e., weight) to make the path computation equally valid in this abstraction model with respect to the path computation in the other two. All the virtual links defined had a weight assigned equal to the smallest number of hops between the nodes in the physical infrastructure. For example, checking the path between the nodes A and D in Figure 26, the virtual link between them (bottom left) has a weigh equal to 3, the number of hops in the original topology (top left).

To validate the presented Blockchain-based SDN architecture and the abstraction models implemented, Figure 27 presents the use case designed (top left) with the complete E2E multi-domain network ready after all the domains have distributed their abstracted network topologies to the other domains. In there, four optical domains (D, D2, D3 and D4) were designed with a set of inter-domain links interconnecting them and each domain had its own SDN Controller and the Blockchain Controller on top. The resulting abstractions are presented Figure 27 with the Transparent (top right), the VLink (bottom left), and VNode (bottom right) E2E abstracted topologies.

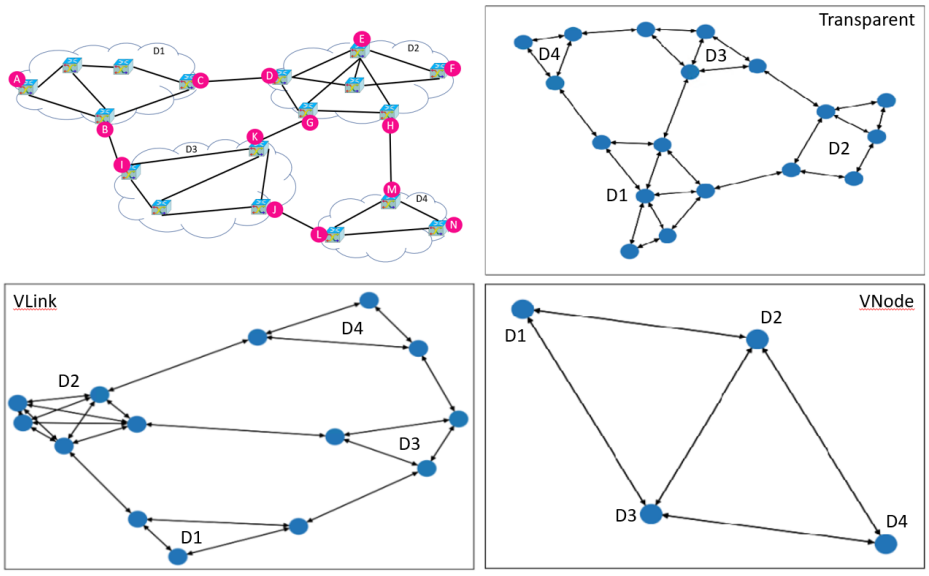

Figure 27: Original use case and its abstracted network topologies.

6 Conclusions and Next Steps

This deliverable (D4.1) concludes the preliminary evaluation of the TeraFlow security and B5G network integration. It describes the security components of the TeraFlow OS, provides the architecture and design, details the interfaces and preliminary results of each task and its associated components. This deliverable and its accompanying Milestone (MS4.1) form the basis for the work starting in the second phase of WP4 which will be documented in MS4.2 and D4.2. In the following paragraphs we provide more details on the next steps of each individual task in WP4.

As for the Centralized Cybersecurity component in Task 4.1 (see Section 3.1) there are several steps that are planned for the final deliverable D4.2. One of the steps is to continue improving and extending the set of machine learning models available for the Attack Inference module. Secondly, we will complete the security response loop by introducing attack mitigation strategies in the Attack Mitigator module. Thirdly, an integration of the Centralized and Distributed Cybersecurity modules will create a more complete security assessment of the network, both in the optical layer as well as in the IP layer. Finally, scalability and efficiency optimizations will be continuously pursued.

Regarding the Distributed Cybersecurity component (see Section 3.2) we plan to design and implement compact ML models to be deployed in the Distributed Attack Detector component to demonstrate that the scalability problems that could arise in a centralized solution can be alleviated using this distributed scheme. The current version of the Distributed Attack Detector deploys a feature extractor to decrease the amount of (i) information (e.g., packets) to be sent to the Centralized Attack Detector and (ii) computational resources required to process such fine-grained information at the SDN controller. Placing some of the ML processing on distributed components will definitely help to decrease the activity of the TeraFlow controller in this regard.

For the DLT component in Task 4.2 (see Section 4.1), the DLT module described in Section 4.1.3.1 will be the foundation to work on core Blockchain technologies and improve the scalability, privacy, and governance. In particular, we will enhance the underlying consensus algorithms using advanced architectures that distribute the transaction capacity without losing the core Blockchain properties. In the current deliverable we built initial smart contracts for accessing core functionalities of the Blockchain. These smart contracts will serve as the basis for more complex and domain-specific smart contracts used by other TeraFlow components. We will explore the technical applicability of smart contracts in evaluating network topologies to gain security insights and limit the attack surface of TeraFlow and its components. We will also explore novel use cases to enhance security in B5G networks, to enforce resource allocation and analysis of real time weaknesses of network applications

With respect to the Compute component in Task 4.3 (see Section 5.1) , the next steps will be focused on validating the complete lifecycle management of the network services handled by the NFV Orchestrator (i.e., OSM implementation) and deriving the implications for the TeraFlow OS. In other words, an update of a network service (e.g., increasing the network bandwidth requirements) imposes an interaction with the Compute component at the TeraFlow OS to automatically fulfil such a network service update. On the other hand, for the Inter-domain component as illustrated in section 5.2.3, there are still some key aspects that need to be defined and deployed. As an example, it is required to identify the elements needed to create, store, and maintain (update) the service catalogue and service inventory to be exposed between peer TeraFlow OSs. Additionally, it is also very relevant to determine how an inter-domain service is modelled (i.e., required information, attributes, requirements), etc. These and other aspects will need to be carefully explored and targeted.

References

[1] Open Source NFV Management and Orchestration (OSM), https://osm.etsi.org/.

[2] N. Skorin-Kapov, M. Furdek, S. Zsigmond and L. Wosinska, "Physical-layer security in evolving optical networks," in IEEE Communications Magazine, vol. 54, no. 8, pp. 110-117, August 2016, DOI: 10.1109/MCOM.2016.7537185.

[3] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X., "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN," in ACM Transactions on Database Systems (TODS), 42(3), 1-21, 2017.

[4] Carlos Natalino, Carlos Manso, Ricard Vilalta, Paolo Monti, Raul Muñoz, Marija Furdek, "Scalable Physical Layer Security Components for Microservice-Based Optical SDN Controllers," in Proc. of ECOC, 2021, paper We3E.2.

[5] Carlos Natalino, Carlos Manso, Lluis Gifre, Raul Muñoz, Ricard Vilalta, Marija Furdek, Paolo Monti, "Microservice-Based Unsupervised Anomaly Detection Loop for Optical Networks, " in Proc. of OFC, 2022, paper Th3D.4. To be presented.

[6] M. Furdek et al., "Machine learning for optical network security monitoring: A practical perspective", J. Lightw. Technol., vol. 38, no. 11, pp. 2860–2871, 2020. DOI: 10.1109/JLT.2020.2987032.

[7] S.S. Sachin et al., "Blockchain for Distributed Systems Security", Wiley-IEEE Computer Society, 2019.

[8] S. Boukria et al., "BCFR: Blockchain-based Controller Against False Flow Rule Injection in SDN," 2019 IEEE ISCC, 2019.

[9] S. Misra et al., "Blockchain-Based Controller Recovery in SDN," IEEE INFOCOM, 2020.

[10] P. Alemany, et al., "Peer-to-Peer Blockchain-based NFV Service Platform for End-to-End Network Slice Orchestration Across Multiple NFVI Domains," IEEE 5GWF, 2020.

[11] P. Alemany, et al., "Managing Network Slicing Resources Using Blockchain in a Multi-Domain Software Defined Optical Network Scenario," ECOC, 2020.

[12] P. Alemany, et al., "End-to-End Network Slice Stitching using Blockchain-based Peer-to-Peer Network Slice Managers and Transport SDN Controllers," OFC, 2021.

[13] V. Lopez et al., "Transport API: A Solution for SDN in Carriers Networks", ECOC, 2016

[14] R. Muñoz, et al., "The ADRENALINE Testbed: An SDN/NFV Packet/Optical Transport Network and Edge/Core Cloud Platform for End-to-End 5G and IoT Services," EUCNC, 2017.

[15] ConsenSys Software Inc. 2021, "https://www.trufflesuite.com/ganache"online. Accessed in November 3, 2021

[16] M. Fiorani, A. Rostami, L. Wosinska, and P. Monti, "Abstraction models for optical 5g transport networks," J. Opt. Commun. Netw.8, 656–665(2016)

[17] HTTP Response Codes: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

