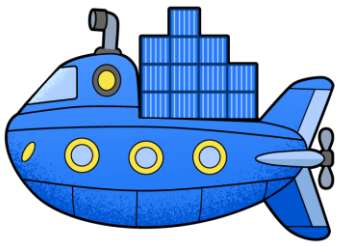


NEMO

Next Generation Meta Operating System

D2.1 Analysis Nemo Underlying Technology

Document Identification			
Status	Final	Due Date	31 /10/2023
Version	1.0	Submission Date	03/11/2023

Related WP	WP2	Document Reference	D2.1
Related Deliverable(s)	D1.1, D1.2, D3.1	Dissemination Level (*)	PU
Lead Participant	ENG	Lead Author	Antonello Corsi
Contributors	ENG, TSG, INTRA, TID, UC3M, WIND3, CMC, AEGIS, SU, UPM, STS, ATOS, SYN	Reviewers	AEGIS SU

Keywords:

Cybersecure Micro-services' Digital Twins, Cybersecure Federated Deep Reinforcement Learning, Federated Machine Learning, Federated meta Network Cluster ControllerDevice-to-Device

Document Information

List of Contributors	
Name	Partner
Antonello Corsi ; Andrea Sabatino	ENG
Victor Gabillon	TSG
SKIAS Dimitrios	INTRA
Luis M. Contreras, Alejandro Muñiz, Francisco J. Cano, Juan L. Ramos, Jesús Folgueira ; Hugo Ramonpascual	TID
Gianluca Rizzi	WIND3
Mika	Skarp, Jose
Spyridon Vantolas, Konstantinos Raftopoulos, Nikolaos Papadakis, Manos Karampinakis	AEGIS
Mohamed Legheraba	SU
Alberto del Rio, Javier Serrano, David Jimenez	UPM
G.Spanoudakis, Y.Papaefstathiou	STS
Aitor Alcázar-Fernández	ATOS
Borja Nogales, Iván Vidal, Francisco Valera	
Terpsi Velivassaki	SYN

Document History			
Version	Date	Change editors	Changes
0.1	10/07/2023	Antonello Corsi (ENG)	Intial ToC
0.2	17/07/2023	Alberto del Rio [UPM]	Inputs in 2.2.2 (Federated Learning & Reinforcement Learning) and in 2.3.2 (Resource Allocation)
0.3	01/09/2023	Andrea Sabatino (ENG)	Second revision of the ToC
0.4	08/09/2023	Antonello Corsi	ENG contribution
0.5	16/10/2023	A. Corsi (ENG), D. Skias (INTRA), G.Spanoudakis, Y.Papaefstathiou (STS)	Updated contributions
0.6	20/10/2023	Terpsi Velivassaki (SYN)	Updated contributions
0.7	24/10/2023	Terspi Velivassaki (SYN)	Updated contributions
0.8		Antonello Corsi (ENG) Andrea Sabatino (ENG)	Updated contributions
0.9	25/10/2023	Luis M. Contreras (TID), Alejandro Muñiz (TID), Francisco J.	Updated contributions

		Cano (TID), Juan L. Ramos (TID)	
0.10	27/10/2023	Spyridon Vantolas (AEGIS), Nikolaos Papadakis (AEGIS), Manos Karampinakis (AEGIS)	Peer review
1.0	3/11/2023	ATOS	Quality review and submitted to EC

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Antonello Corsi (ENG)	03/11/2023
Quality manager	Rosana Valle Soriano (ATOS)	03/11/2023
Project Coordinator	Enric Pages Montanera (ATOS)	03/11/2023
Technical Manager	Harry Skianis (SYN)	03/11/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AAA	Authentication, Authorization, and Accounting
AD	Architecture Description
AI	Artificial Intelligence
AIOTI	Alliance for IoT Innovation
API	Application Programming Interface
CFDRL	Cybersecure Federated Deep Reinforcement Learning
CI/CD	Continuous Integration & Continuous Delivery/Continuous Deployment
CMDT	Cybersecure Micro services Digital Twin
CN	Core Network
CNCF	Cloud-Native Computing Foundation
CNI	Container Network Interface
CPU	Central Processing Unit
DAST	Dynamic Application Security Testing
DLT	Distributed Ledger Technology
DRL	Deep Reinforcement Learning
DT	Digital Twin
Dx.y	Deliverable number y belonging to WP x
E2E	End-to-End
EC	European Commission
GDPR	General Data Protection Regulation
HLA	High-Level Architecture
HW	Hardware
IAM	Identity and Access Management
IAST	Interactive Application Security Testing
IMC	Intent-based Migration Controller
IoT	Internet of Things
IBMC	Intent-Based Migration Controller
K8s	Kubernetes
KPI	Key Performance Indicator
MANO	Management and Orchestration
MEC	Multi-access Edge Computing
meta-OS	Meta-Operating System
ML	Machine Learning
mNCC	meta–Network Cluster Controller
mRA	meta-Reference Architecture
MO	meta-Orchestrator
MOCA	Monetization and Consensus-based Accountability
NAC	NEMO Access Control

NFV	Network Function Virtualization
NGIoT	Next-Generation IoT
OS	Operating System
PA-LCM	Plugin & Apps Lifecycle Manager
PPEF	PRESS and Policies Framework
PRESS	Privacy, data pRotection, Ethics, Security & Societal
RAN	Radio Access Network
RASP	Run-time Application Security Protection
REST	Representational State Transfer
RBAC	Role-Based Access Control
RL	Reinforcement Learning
SDK	Software Development Kit
SDN	Software Defined Networking
SDO	Standard Development Organization
SD-WAN	Software-Defined Wide Area Network
SLA	Service Level Agreement
SLO	Service Level Objective
TEE	Trusted Execution Environment
V&V	Validation & Verification
VM	Virtual Machine
VNF	Virtual Network Function
WP	Work Package
YAML	*Yet Another Markup Language*

Executive Summary

This document's main aim is to delineate the work carried out in the various tasks of WP2, focusing on the distinct components that forms part of the skeleton of the meta-OS of NEMO.

The primary aim is to offer a comprehensive description, addressing the initial hurdles associated with the development of these components that start to coexist within the same ecosystem. Through this exposition, the reader will fully understand the advancements proposed by NEMO partners respect to the State of the Art of the used technologies.

The document underlines the so far achievements during the initial period, placing emphasis on the overall methodology and approaches utilized to develop and report a Proof of Concept (PoC) concerning the key assets of NEMO in WP2.

These assets include:

The extension of the Distributed Ledger Technology (DLT) meta-level DT concept into the micro-services domain for prompt and reliable instances indexing,

The re-shape of MLOps to a novel CF-DRL paradigm ensuring cybersecure, federated real-time DRL with a balance between efficiency and ML convergence performance.

The documentation regarding the AI-driven mNCC facilitating ad-hoc Device-to-Device (D2D) and Device-to-Edge (D2E) network clusters connectivity, alongside AI-based End-to-End (E2E) multi-tenant clusters ensuring guaranteed QoS over highly dynamic networks.

In the end the interested reader can here find technical details concerning the developmental steps of the main component blocks along with an initial working version showcased as a mock-up PoC.

1 Introduction

This deliverable follows an approach that describes the various components realized in WP2 which are underpinned by the foundational principles of the NEMO meta-architecture. The NEMO meta-architecture represents an extensive "Meta-Architecture Framework" (MAF) specifically designed for the meta-operating system (meta-OS) environment, as thoroughly detailed in Document D1.2 - NEMO meta-architecture, components and benchmarking and D3.1 - Introducing NEMO Kernel. With this aim the subsequent sections should be regarded as a continuation of the viewpoint's methodology introduced in D1.2, which is defined as a set of steering principles for developing an Architectural View, aimed at fulfilling a series of specified requirements. NEMO has outlined several critical Viewpoints for the meta-OS meta-architecture, including "Network", "User", "Logical", "Operational", "Functional", "Process", "Development", and "Physical". The following subsections cover the Development and Process Viewpoints.

In the NEMO meta-Operating System, the WP2 related technologies are essential to enhance "by design" Cybersecure Micro-services Digital Twins (CMDT), advanced Cybersecure Federated Deep Reinforcement Learning (CF-DRL) and realize reliable on-demand self-healing end-to-end cloud clusters, including dynamic management of 5G micro-slices/resources (mNCC). This document is the initial report of a series of three reporting on the development progress of these technologies.

1.1 Purpose of the document

This document delineates the advancements of the three tasks encompassed in WP2. Its objective is to give to readers the essential background to comprehend the individual tasks, alongside elucidating the concepts for the main outcomes of these tasks. Additionally, the interconnections among the components in WP2 are detailed, along with the interaction among the components and the remaining parts of the NEMO solution that came from other work packages. Given its rich background information, this document is geared towards readers possessing a foundational grasp of cloud computing, AI operations as well as micro service-oriented architectures.

1.2 Relation to other project work

D2.1- Analysing NEMO Underlying Technology is a significant achievement in the progress of the NEMO project. This document is impactful, especially when viewed in parallel with D3.1 - Introducing NEMO Kernel, in fact both documents collectively establish the initial version of the overall NEMO technologies. Primarily, D2.1 follows from what has been done in D1.1 Definition and analysis of use cases and GDPR compliance and D1.2 - NEMO meta-architecture, components and benchmarking. Initial version about software architecture, ethical, privacy, and GDPR compliance frameworks that will govern the Living Labs of NEMO. D2.1 builds upon this foundation by exploring the technical facets of the main WP2 Objective so to enhance the technical strategies surrounding the NEMO project.

1.3 Structure of the document

This document is a comprehensive exploration of the WP2 Enhancing NEMO Underlying Technology. The first chapter and related subsections are an introduction detailing the document's purpose, its relation to other projects, and its structure. Chapter 2 presents the CMDT detailing its architecture, chosen technologies, and development evolution the same approach is followed in Chapter 3 that delves into the of CF-DRL and in Chapter 4 that details the Federated meta Network Cluster Controller (mNCC) TID. Lastly the Chapter 5 presents the Time Sensitive Networking (TSN) methodologies. The conclusive Chapter 6 end the deliverable with a proof of concept demonstration for all the WP2 components.

2 Cybersecure Micro-services Digital Twins

2.1 Overview

One of the most important components of the NEMO meta-OS is the CMDT that it is a comprehensive solution to cope with the ever-evolving world of micro services applications ensuring that they are secure and flexible.
The CMDT is a mainly a microservice [1] registry that ensure that every microservice and workload, from creation to retirement, is meticulously tracked and reported. In essence, CMDT ensures that there's always a reliable source of truth about the status and location of every microservice within its registry.
Conceptually the main functional blocks that constitutes the CMDT are:

- **Micro-services Registry**: CMDT meticulously stores detailed information about the available Micro-services and workloads. This repository of data encompasses vital details about each microservice, such as its functionality, status, dependencies, and more, thereby facilitating efficient management and deployment of Micro-services across the NEMO projects.
- **Accessibility and Interaction**: Ensuring a harmonious interaction between various actors and microservices, CMDT makes the stored information readily available. It ensures that all the users in the NEMO ecosystem can effortlessly access and interact with the requisite microservices.
- **Dynamic Dashboard Display**: CMDT can be accessed via a user-friendly dashboard that not only displays a comprehensive view of the Micro-services but also provides insightful data about the field edge devices linked to them.
- **Self-Discovery and Continuous Updates**: With its self-discovery functionalities, CMDT ensures that it is perpetually updated about the state of the various Micro-services from "the crib to the cradle." This means that from the moment a microservice is created to the moment it is decommissioned or replaced, CMDT autonomously tracks and updates its status, dependencies, and other relevant information, ensuring that the registry is always current and accurate.
- **Integration with Field Edge Devices**: CMDT goes beyond the microservices, extending its capabilities to manage and display information related to the field edge devices upon which the Micro-services are built. This ensures that the health, status, and performance of the underlying hardware are also monitored and managed effectively, ensuring optimal performance and reliability of the Micro-services deployed.

To ensure the seamless operation of all these functionalities, we planned an information model that meticulously tracks the data constituting the CMDT. This data is encapsulated in a YAML file, referred to as the CMDT descriptor within the NEMO project and that it is managed in the overall way following the architecture in Figure 1.

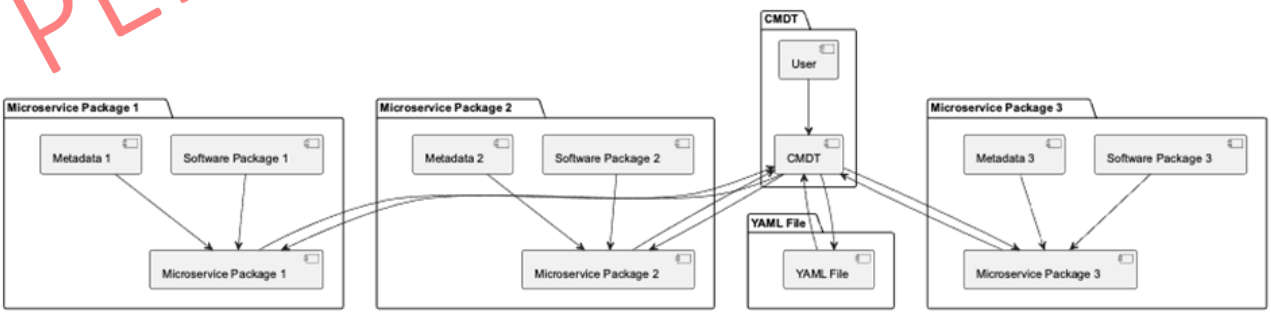


Figure 1: YAML FILE architecture

2.2 State of the Art

Just before to go deep in the architecture and then in the development details of the CMDT it is useful to provide a very concise map of all the product that realize in the research and in the market the main functionalities that are provided by the CMDT. None of them cover all the CMDT functionalities so in this sense the CMDT is a new solution in the ecosystem.

The main ingredient of a micro service architecture is Containerization and Orchestration for which we have:

- Docker[1]: A platform used to containerize applications, ensuring they have all necessary dependencies to run in any environment.
- Kubernetes[2]: A container orchestration platform, facilitating the deployment, scaling, and management of containerized applications.

One of the main aspects is the capability to registry the micro services once they are live so to this aim we have the Service Mesh:

- Istio[3]: Enhances the security, observability, and traffic management of microservices.
- Linkerd[4]: Another service mesh option providing similar features as Istio but with a focus on simplicity and performance.

For the telemetry aspects is useful cite the product related to monitoring and Logging:

- Prometheus[5]: An open-source monitoring solution that integrates well with microservices.
- ELK Stack[6] (Elasticsearch, Logstash, Kibana): A popular stack for logging, processing, and visualizing data in real-time.

The interaction with the external word is guarantee by API Gateways:

- Kong[7]: An API gateway that manages traffic between microservices.
- Apigee[8]: Offers insights into API performance and enhances security.

On top of all these solutions there is the choice to provide something that guarantee a certain level of security and also make possible for different components in a complex architecture to operate independently, making it easier the maintenance and the scale of the different applications.

This is done via following Message Brokers which are used to send messages between microservices securely:

- RabbitMQ[9]: An open-source message broker that supports multiple messaging protocols. It's known for its robustness, ease of use, and support for a variety of developer platforms.
- Apache Kafka[10]: Originally developed by LinkedIn and now an open-source project under the Apache Software Foundation, Kafka is a distributed streaming platform capable of handling trillions of events a day. It is suitable for event sourcing, stream processing, and real-time analytics.

[1] https://www.docker.com/
[2] https://kubernetes.io/
[3] https://istio.io/
[4] https://linkerd.io/
[5] https://prometheus.io/
[6] https://www.elastic.co/elastic-stack
[7] https://konghq.com/products/kong-gateway
[8] https://docs.apigee.com
[9] https://www.rabbitmq.com/
[10] https://kafka.apache.org/

2.3 Architecture & Approach

Our approach for the CMDT architecture includes the Development and process view so to create a well-established path for the reader to follow the creation of this NEMO architectural part.

Development view

The potential of exploiting data from simple sensor devices with the new computational technologies is boundless. In fact, when these devices become interconnected and numerous services are built upon them, the complexity of managing and extracting valuable information grows exponentially.

To address this challenge, NEMO proposes the concept of CMDT. In this section we will explore the essence of CMDT and its ability to facilitate the extraction of essential information from basic IoT sensors, while accommodating the intricate web of services layered on top.

At the core of this approach, we will use a YAML file as the core artifact that will allow seamless retrieval of data through REST interfaces, abstracting away the underlying complexity of the devices.

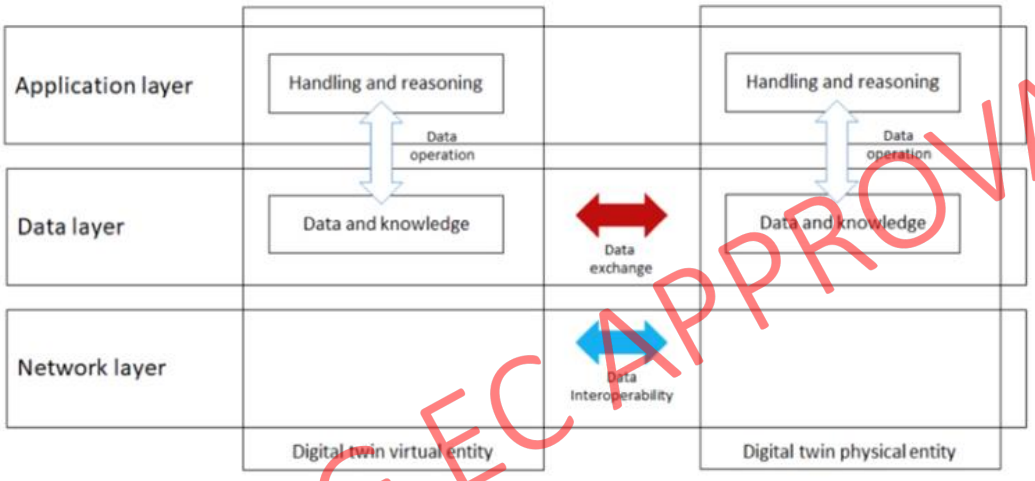


Figure 2: *HLA representation of digital twin example https://aioti.eu/wp-content/uploads/2022/09/AIOTI-Guidance-for-IoT-Integration-in-Data-Spaces-Final.pdf

Here based on the suggested architecture from Alliance for IoT and Edge Computing Innovation (AIOTI)[11] we can provide a first modelling of a digital twin that also incorporates information on the services built on top of the physical device that constitutes it. (See the high-level architecture in Figure 2)

In fact, building upon the sensor data, the CMDT incorporates services that leverage this data for enhanced functionality and value generation (Table1).

#	Description
Services	This section describes the monitored or described services. Each service has a name, description, a list of properties (represented as key-value pairs), and a list of Micro-services (µServices) associated with it. The overall service can be imagined as a list of chained µServices
Analytics	This section represents the analytics services that provide insights about the status and health of the monitored services.

[11] Alliance for IoT and Edge Computing Innovation (AIOTI) - https://aioti.eu/

Entities	This section represents the devices or entities associated with the digital twin. Each entity has a name, description, and a list of services it senses.
Data Streams	This section represents the input/output streams to the services and observable metrics per service. Each data stream has a name, description, and a list of properties associated with it.

Table 1: CMDT architecture description

The information model in Table1 is the first source to infer a YAML file named the descriptor.

Draft of the example "**Descriptor**" to be used in the interaction between CMDT and the other components is presented in Table 2

CMDT_NEMO.yaml

```
# Simple Proposed structure for CMDT
CMDT:
  name: CMDT_structur
  description: Proposed structure for CMDT #Write your description
here
  services: # List of services included in the CMDT
    - Name: service1 #1° service ----------------------------------
-----------------------------------------------
      Description: This is the first service that can be mapped to a
LL
      #package: charts/service1-0.0.0.tgz
      Properties:  #List of [Property, value] (e.g. a value could be
a PRESS policy)
       - Property 1: Value 1
       - Property 2: Value 2
      Microservices: #List of µServices
      - Name: µService 1
        Description: Microservice 1 description on how to access it
        Endpoints:
         - http://service1/microservice1
      - Name: µService 2
        Description: Microservice 2 description on how to access it
        Endpoints:
        - http://service1/microservice2
  Analytics: #List of Analytics_service (providing insights about
Services' status/health)
    - Name: Analytics Service 1
      Description: Provides insights about Service 1's status and
health.
      Endpoints: #that the service is accessible
       - http://analytics-service-1.com
  Entities: #Devices
    - Name: Device 1
      Description: This is the first device that creates data
      Services: #List of Services (sensed properties)
      - Name: Service 1
        Description: Properties of Service 1 built on top of the
device
        Endpoints: #that the sensed property is accessible
```

```
          - http://device-1/service-1
  DataStreams: #e.g. input/output streams to the Services, observable
metrics per service)
    - Name: Data Stream 1
      Description: Input/output stream for Service 1.
      Properties: #List of [Property, value]
       - Property 1: Value 1
       - Property 2: Value 2
```

Table 2: CMDT information model

The main architecture is defined in Figure 3 and in the following pages we are going to detail the different parts of it.

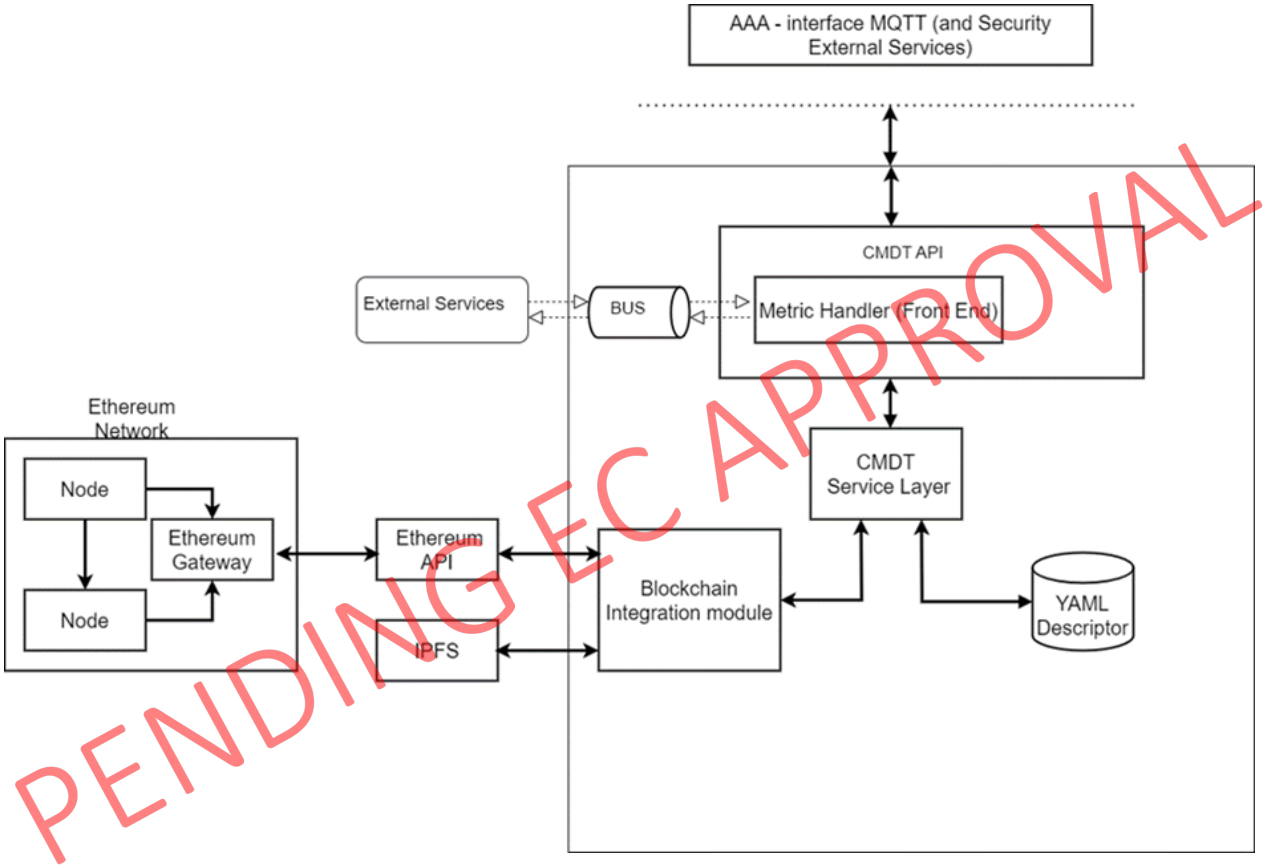


Figure 3: CMDT Architecture

Process view

In the broader scopes of CMDT, we have two main interactions foreseen. Those are meticulously orchestrated to ensure efficient data management and service registration. Keep in mind that each of these macro - functionalities can be further breakdown and that they will be specified and fine tunes across the time life of the project.

- **Service registration**: When a service is up it initiates a series of authentications and data exchange processes that culminate in the registration of the service, with the system gathering and processing metrics and updating all relevant components, from local databases to blockchain storage. This registration is just the beginning of the entire management flow for the Micro-services in the CMDT.

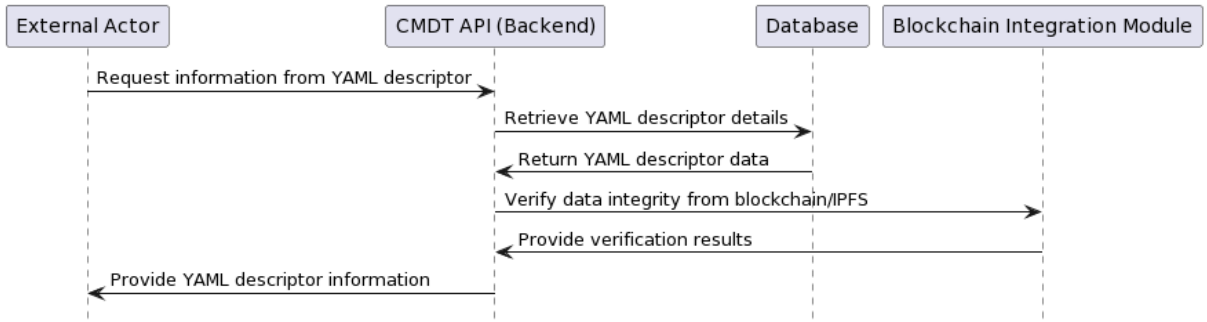


Figure 4: Sequence Diagram for Registration

- **External actors ask information**: As external actors that want to interact with the information in the descriptors CMDT's trigger a backend to data retrieval and verification.

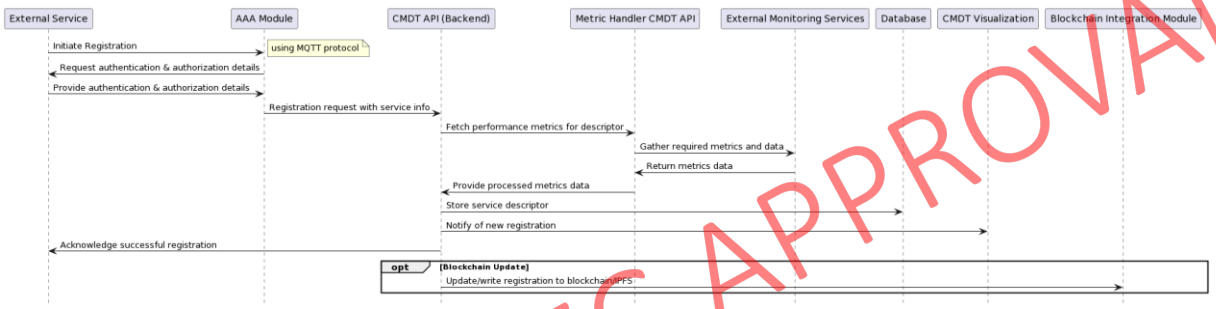


Figure 5: Sequence Diagram for external actors

2.3.1 Description of subcomponents

In the following sub-sections, we will create a detailed breakdown of the CMDT's subcomponents. Each subsection description provides a snapshot of the current state of the subcomponents. As the project continues to evolve over the coming months, we anticipate updates to these descriptions in subsequent deliverables.

2.3.1.1 Authentication, authorization and Accountability (AAA)

Description

The AAA module consists in an interface that allows the MQTT protocol to be enabled. This external module will also take care of enabling authentication, authorization and accountability for the external component.

Example of Technology Enablers

Broker MQTT[12]: (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) lightweight publish-subscribe messaging protocol placed on top of TCP/IP. It has been designed for situations where low energy impact is required and where bandwidth is limited. The publish-subscribe model requires a messaging broker. The broker is responsible for distributing messages to the recipient clients.

[12] https://en.wikipedia.org/wiki/MQTT

keycloak[13]: Keycloak, an open-source identity and access management solution, provides centralized authentication and authorization services. Its features include single sign-on (SSO), user management and role-based access control (RBAC), keycloak is an open source software product to allow single sign-on with identity and access management aimed at modern applications and services

Technology Chosen

After finalizing the design, we will evaluate the optimal technologies for deployment and the related implementation.

Interfaces

The interfaces among the different subcomponents are in early stage and they will be defined in the next deliverables.

Licensing

This subcomponent/module is released under an open-source license, ensuring accessibility and flexibility for users Please note that the decision regarding the selection of an appropriate license for the AAA service will be finalized in the next months, depending on the various technical choices that are yet to be made.

2.3.1.2 Metric Handler CMDT API

Description

The Metric Handler is the component responsible for managing the dynamic aspects of the CMDT descriptor document. It is capable of working with general-purpose message buses such as Kafka or RabbitMQ to supply dynamic information for the descriptor and enable retrieval if needed. The Metric Handler serves as a data provider for the front-end, supplying the necessary information for visualization purposes i.e. analytics information from the descriptor.

A transformation function is needed to take logs as input and provide the processed data ready to be used by upper layers as output. For example, the visualization component will demand processed data, representing the targeted KPIs, instead of raw data.

So, in summary we think of this Metric Handler CMDT component like this:

A specific microservice contains certain data or information. The CMDT requires performance details about this microservice. This information can be sourced using the Metric Handler. For instance, by connecting to Prometheus instances related to a particular microservice, the Metric Handler can extract and collect the necessary data.

The "descriptor", which is essentially a microservice descriptor encapsulating all pertinent details, can be created or updated. This information is stored through an interface, allowing the CMDT to retrieve data via the Metric Handler. When other components undergo modifications, they communicate these changes, resulting in updates to the descriptor through another interface.

Suppose a component that update the "property" CPU used by the microservice. This change occurs due to communication from the given component. Queues become valuable in these scenarios as they generate an event leading to property updates.

The information gathered to fill the "descriptor" is added to a "registry". Given the record in this registry, and based on specific use cases and requirements, the descriptor might be stored in a Mongo DB format or on an IPFS, with an option to save it on a blockchain as an identifier

[13] https://en.wikipedia.org/wiki/Keycloak

Example of Technology Enablers

Kafka: Given its capability to handle high-throughput, real-time streams, Kafka can be used to gather, process, and disseminate large volumes of event data, this makes it a good candidate for managing dynamic aspects of the CMDT descriptor document.

RabbitMQ: As an open-source message broker software, RabbitMQ can be used for sending data across applications and services asynchronously. Its versatility in supporting multiple messaging protocols makes it suitable for collecting data from varied sources.

Prometheus: This open-source systems monitoring and alerting toolkit is particularly tailored for reliability and scalability. With the Metric Handler's requirement to connect to Prometheus instances to extract data, this becomes a central technology.

Technology Chosen

After finalizing the design, we will evaluate the optimal technologies for deployment.

Interfaces

The interfaces among the different subcomponents are in early stage and they will be defined in the next deliverables.

Licensing

This subcomponent/module is released under an open-source license, ensuring accessibility and flexibility for users.

2.3.1.3 CMDT Visualization

Description

CMDT Visualisation component is the interface between the end-user and CMDT services. It allows users to access and utilize the services provided by CMDT offering either real-time or historical data representations. The visualisation component takes as input log data from the Metric Handler component to provide both real-time monitoring of raw data and processed data representing targeted KPIs.

The CMDT Visualisation component builds upon Advanced Visualisation Toolkit (AVT[14]), which is a sophisticated set of visualisation tools, capable of connecting with state-of-the-art streaming platforms and databases, while it can be tuned to consume different kinds and formats of data, providing essential, interactive and user-friendly visualisations. The AVT is provided as a framework that consists of a web application, accompanied by a dedicated server based on Node.js[15] to handle all the external connections, real time messaging and user management. Both components are provided as docker containers with two separate docker images. The AVT front-end is based on the Angular.io[16] framework in conjunction with a set of visualisation libraries that are interchangeably used to cover specific needs and particularities of the analysed data. These include interactive timeline representations for time-dependent data, graph-based visualisations, chord diagrams, geospatial representations, etc.

[14] https://aegisresearch.eu/solutions/advanced-visualization-toolkit/

[15] https://nodejs.org/en/

[16] https://angular.io

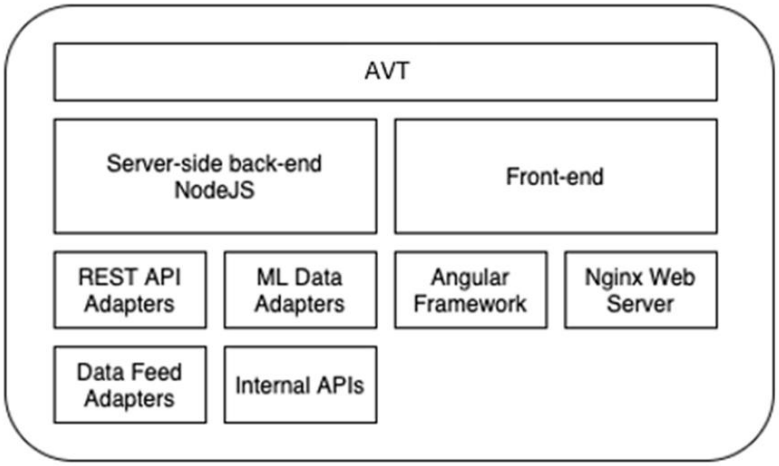


Figure 6: AVT internal architecture

The internal architecture of a deployed AVT solution is as follows (Figure 6): The front-end container contains the main User Interface (UI) of AVT and the complementary web server that serves it. The interface's internal proxy configurations are handled inside this container. The back-end container contains the AVT's server-side middleware, which is responsible for handling all the necessary external communications in the deployed environment and the processing activities of the toolkit. Several adapters are integrated in this core component in order to support real-time data streams (e.g., FIWARE, Kafka, etc), database connections (e.g., MongoDB, MySQL), different kinds of data formats (e.g., JSON files), REST APIs and more. The implementation of the required data queries and the needed pre-processing activities are taking place within this component. Continuous communication and data transmission between these components coordinate the AVT's completed flow.

Example of Technology Enablers

Angular.io or React: For building dynamic, efficient, and robust User Interfaces.
Node.js: A server-side runtime environment suitable for building scalable and efficient server-side applications. It's especially useful when dealing with real-time data and handling numerous I/O operations, making it apt for your middleware.

FIWARE: for context information management and to get an integrated view of data from various sources.
Kafka: A distributed streaming platform suitable for building real-time data pipelines and streaming applications.

Technology Chosen

After finalizing the design, we will evaluate the optimal technologies for deployment.

Interfaces

The interfaces among the different subcomponents are in early stage and they will be defined in the next deliverables.

Licensing

This subcomponent/module is released under an open-source license, ensuring accessibility and flexibility for users.

2.3.1.4 CMDT API - Backend

Description

The CMDT Service Layer will parse YAML files from the database, and it will also interact with the Blockchain Integration Module to get data from the blockchain. Similarly, if requested, it will send data to the Blockchain Integration Module when a YAML file needs to be updated on the blockchain.

The backend will also be in charge to manage all the flow of data among the different subcomponents.

Example of Technology Enablers

Express.js: One of the most popular web application frameworks for Node.js that is fine tuned for the process of building robust APIs and at the same time providing a minimal and flexible enabler that offers a variety of web application features.

Nest.js: Is a Node.js framework for building scalable and maintainable server-side applications. Equipped by design with TypeScript by default (but also supports plain JavaScript)I t's built around the concept of decorators, modules, and has its own ecosystem that provides tools like the CLI and integrations with many popular JS libraries.

Technology Chosen

Once we already have the complete design, we will study the best technologies to realize the deployment, and therefore select the ones chosen.

Interfaces

The CMDT service layer, on the backend side, will interface with the database (Mongo DB) and will also interact with the Blockchain integration module to obtain data from the blockchain.

Licensing

The CMDT service layer module is released under an open-source license, ensuring accessibility and flexibility for users.

2.3.1.5 Blockchain integration module

Description

The Blockchain integration module will interact with the CMDT service layer and service representation. It will also communicate with the Blockchain client to perform the actual reading and/or writing to the blockchain and/or with IPFS. Blockchain Integration Module is basically a manager that, depending on the use case, decides which part of the Digital Twin to store on the blockchain and/or on IPFS.

Example of Technology Enablers

Ethereum: It is a blockchain platform that enables the execution of smart contracts and decentralised applications, making it versatile and suitable for a wide range of use cases.

Ethereum APIs: We are evaluating some different Ethereum APIs to connect client applications to the blockchain. The main ones are: Alchemy, BlockCypher, Infura or py.ethereum [17].

JSON-RPC: It is a communication protocol that uses the JSON format to allow applications to send requests and receive responses when calling methods or functions on remote servers. The main advantages of this protocol are that it is versatile, easy to implement and widely used in applications that

[17] https://ethereum.org/sw/developers/docs/apis/backend/

require interaction between client-server systems. This protocol is needed if we finally decided to work with the Ethereum API commented above.

Technology Chosen

The technology to realize the deployment of this component is still under study as long as it needs to have a clear understanding of the blockchain implementation and the different potential clients, being a pending task for the next stage.

Interfaces

As it is represented in the architecture design, this module will interact mainly with three components:

- CMDT API Backend: This interface will request the Blockchain/IPFS for data integrity verifications and it will be module in charge to update the registries that will be saved. For these two tasks, we are evaluating two possible interfaces: First one, a single interface that could work as a synchronous interface in one case and as an asynchronous interface in the other one. The second possibility is to count with two separated interfaces: one bi-directional oriented to realize requests and the other directional oriented to insert new registers.
- Ethereum API and IPFS: These two components will be acceded using existing APIs to communicate with each of the modules, so from this side of the communication we will adapt our implementation at the other side.

Licensing

The Blockchain integration module is released under an open-source license, ensuring accessibility and flexibility for users.

2.3.2 Interaction with other NEMO components

The CMDT interacts with other NEMO components, as described below and with this aim we provide a general outline, as previously mentioned in D1.2, of the anticipated connections with the NEMO components once CMDT is fully operational.

2.3.2.1 Meta-Orchestrator

The CMDT and the meta-Orchestrator (MO) are intricately linked, characterized by a dynamic input and output relationship. The CMDT acts as a primary information source, furnishing the MO with critical security insights concerning microservices, aiding in risk assessment and security measure implementation. Conversely, the MO serves as a pivotal transmitter of data to the CMDT, facilitating security configurations, monitoring, and enforcement, thereby enabling the CMDT to disseminate this information to other components within the system.

2.3.2.2 Intent-based Migration Controller

The CMDT closely collaborates with the Intent-Based Migration Controller (IBMC) through the MO. The CMDT supplies crucial service descriptions and security insights to the IBMC, aiding in risk assessment and supporting secure service deployment. Simultaneously, the IBMC provides intent-based network paths and deployment instructions to the CMDT via the MO, ensuring that the CMDT possesses the necessary data for security configurations, monitoring, and enforcement during the deployment process.

2.3.2.3 PRESS & Policy Enforcement Framework

The PRESS & Policy Enforcement Framework communicates with the CMDT to ensure policy compliance through multifaceted policies that address different aspects of the application lifecycle (security, privacy, cost, environmental impact, etc.), channelling "violations" as input.

2.3.2.4 Plugin & Applications Lifecycle Manager

The type of interaction associated with this component is categorized as input/output and the Plug-ins and Application Lifecycle Manager it is designed to be a flexible plug-in and application lifecycle manager, facilitating timely deployment of necessary plug-ins. In its interaction with the CMDT, it supplies "descriptors" as input, while retrieving "lifecycle info" as output from the CMDT.

2.3.2.5 Monetization and Consensus-based Accountability

The Monetization and Consent-Based Accountability," interfaces with the CMDT. This interaction facilitates the exchange of information essential for writing data to, and retrieving data from, the blockchain.

2.4 Technology Chosen

To realize the CMDT software architecture, a carefully chosen technology stack is essential. This section outlines the various technologies that can be employed for each component of the CMDT system. The following technologies are recommended for achieving the desired functionality while considering ease of deployment and integration.

Programming Language

Python is selected as the programming language for developing the CMDT software due to its versatility, extensive library support, and ease of use. Python enables efficient handling of YAML files, zip file manipulation, and blockchain integration. It offers a rich ecosystem of libraries and frameworks, facilitating rapid development and prototyping.

Database

MongoDB, a powerful NoSQL document database, is chosen as the preferred option for CMDT. MongoDB's flexible document model allows for easy storage and retrieval of YAML files and zip files. It provides scalability, performance, and a simplified data model, making it an excellent fit for CMDT's requirements.

YAML Processing

PyYAML, a widely-used YAML parser and emitter library for Python, is the recommended choice for YAML processing in CMDT. PyYAML allows effortless parsing, manipulation, and serialization of YAML data, enabling seamless integration with the MongoDB database.

Zip File Handling

Python's built-in zipfile module is selected for managing zip files within CMDT. The zipfile module offers a comprehensive set of functionalities for creating, extracting, and modifying zip archives. Its intuitive API simplifies working with zip files and aligns with the Python programming language choice.

Blockchain Integration

To integrate the CMDT software with the Ethereum blockchain, an Infura node will be utilized. Infura is a popular infrastructure provider that offers a reliable and scalable gateway to interact with the Ethereum network without the need to run a local Ethereum node.

Web3.py, a Python library for interacting with Ethereum, will be employed to communicate with the Infura node. Web3.py provides a high-level API that simplifies the interaction with Ethereum smart contracts and facilitates the storage of a portion of the YAML data in the blockchain ledger.

By utilizing the Infura node and Web3.py library, the CMDT software can leverage the benefits of the Ethereum blockchain, such as decentralized storage, immutability, and tamper resistance, while eliminating the need to maintain and manage a local Ethereum node.

It's worth noting that Infura requires an API key to access its services. Therefore, proper authentication and security measures should be implemented to protect the API key and ensure secure communication between the CMDT software and the Infura node.

Authentication and Authorization

To enhance the security of CMDT, a Keycloak security layer is incorporated. Keycloak, an open-source identity and access management solution, provides centralized authentication and authorization services. Its features include single sign-on (SSO), user management, and role-based access control (RBAC), ensuring secure access to CMDT and its functionalities.

By leveraging this technology stack, CMDT can be developed with a robust and efficient architecture, allowing for seamless management of YAML files, zip files, blockchain integration, and authentication and authorization.

2.5 Conclusion, Roadmap & Outlook

The CMDT is an evolving component that aligns with the roadmap established for other components within the Meta OS framework. An initial version, showcasing a blueprint of the component, is foreseen for release by the end of M14. Following this, a functional Proof of Concept (PoC) is scheduled to the end of M16. Upon establishing the basic functionalities through this PoC, the component will be involved in the final stages of the functional development. The progress in these phases will be documented and verified in D2.2 and D2.3.

3 Cybersecure Federated Deep Reinforcement Learning

3.1 Overview

The CFDRL component provides capacity of learning decision-making models in a collaborative and distributed way between communicating nodes/entities. It comes with features that strengthens the cyber-security of the learning process by detecting external attacks and deploying appropriate countermeasures. The learning procedure combines two complementary learning paradigms: Federated Learning (FL) and Reinforcement Learning (RL).

- Federated Learning: Federated Learning is a distributed machine learning approach that enables training models on decentralized data sources while maintaining data privacy and security. It allows multiple participating entities, such as edge devices or local servers, to collaboratively train a shared model without sharing their raw data. Instead, only model updates or gradients are exchanged between the central server and the participating entities. This decentralized approach helps protect sensitive data, as it remains within the respective entities control.
- Reinforcement Learning is a sub-field of machine learning that focuses on training agents to make sequential decisions in an environment to maximize cumulative rewards. In our system, Reinforcement Learning algorithms are employed to enable autonomous decision-making and policy optimization. Agents within the federated network interact with the environment and learn through trial and error, aiming to optimize a defined reward function.

The RL and FL paradigm can be combined in different manners. On one end of the spectrum, in Horizontal FRL all the entities solve the exact same copy of a RL problem in parallel, without any interaction between the entities, but learn from different local interaction data the exact same model by sharing and copying local models between nodes/entities. At the other end of the spectrum, in Vertical FRL, all entities, as in a multi-agent setting, live in the same problem but have different views of it, different actions, different model and can interact. However, they learn collaboratively through the possibility to share their model and thus coordinate. In our first investigation of FRL we will implement the Horizontal version of FRL.

We add cybersecurity features to our CF-DRL component in order to make the component robust to attack that want either to steel the model or perturb the learning.

In that respect we start modelling several attacks and developing specific countermeasures to them. Within the 1st year of the project, we have implemented 1 ML-targeted attacks (weight poisoning attack) and have developed one countermeasure (differential privacy). In the next phase of the project, we will implement at least four more ML-targeted attacks and at least one more countermeasure Additionally, certain features provided by the Security Modules developed in Task 3.3 and described in Section e.3 of deliverable D3.1 can also be utilized by the CF-DRL component so as to be more secure (e.g. encrypted communication between the clients and the server).

Within the NEMO project, there are two types of applications that are foreseen:

- Living Labs: The living labs of NEMO propose decision-making problem that could be formalized in a RL formalism. Moreover, some of them would fit the Federated Learning framework and thus fill their need for collaborative and privacy preserving learning.
- Meta-Orchestrator: Part of NEMO is to implement an intelligent open source meta-Orchestrator able to explore the decentralization and distribution of computing workflow over the IoT to edge to cloud continuum. NEMO meta-Orchestrator will utilise intelligence based on CF-DRL to assess strategies for orchestration of different kind of containerized Micro-services or unikernels workflows enabling migration, placement and scaling. Decision will be mainly based on three parameters: a) migration time, b) downtime, and c) overhead time, while additional

functional parameters, such as network and resource availability, *non-functional requirements*, such as policies, energy efficiency, CO2 footprint and *FinOps requirements*, like networking and hosting cost will also be analysed and evaluated.

3.2 State of the Art

CF-DRL is a combination of two well-established fields, Federated Learning and Reinforcement Learning respectively. We will first review the SOTA in Federated learning then Reinforcement Learning and finally will focus on the works that study their combination.

3.2.1 Federated Learning

Federated Learning is a powerful machine learning approach that allows multiple parties to collaboratively train a model without the need to share their data. For a specific review of federated Learning see [2]

3.2.1.1 Architecture (centralized or peer to peer)

When Federated Learning was first introduced[3], the architecture was centralized. As illustrated in Figure 7 we distinguish 3 types of architecture:

1. **A central server** is responsible for coordinating clients of the network, aggregating the results and sending the final model to the clients. The presented algorithm (FederatedAverage) works as follows. First, the server sends the machine learning model the clients. Thus, each client performs one or several steps of training on his local dataset. Then, each client sends back the local parameters to the server, who compute the average of all received parameters and send the new version of the global model to the clients (and so on).
2. **Decentralized Approaches**: Aggregation can also be performed on a decentralized network (peer to peer). The first example of a peer-to-peer federated learning model was introduced by [4]. Decentralized approaches to federated learning are being explored to address the security and privacy concerns of centralized federated learning systems. In a fully decentralized approach, there is no longer a central server that acts as an initiator, coordinator, and model aggregator. Communication with the central server is replaced by peer-to-peer communication. In this type of topology, network agents learn personalized models, while communication with other members is essential in order to increase the quality of their model. For this reason, a number of protocols have been proposed in the literature such as the Gossip [5], Dis-PFL [6]and Soft-DSGD [7].
3. In a **semi-decentralized architecture**, a new aggregator is chosen at each round, using byzantine algorithms (or a blockchain consensus algorithm). The algorithm then works as in the centralized case. Some architectures also use a smart contract as an aggregator.

For a more complete view on decentralized architectures see [8].

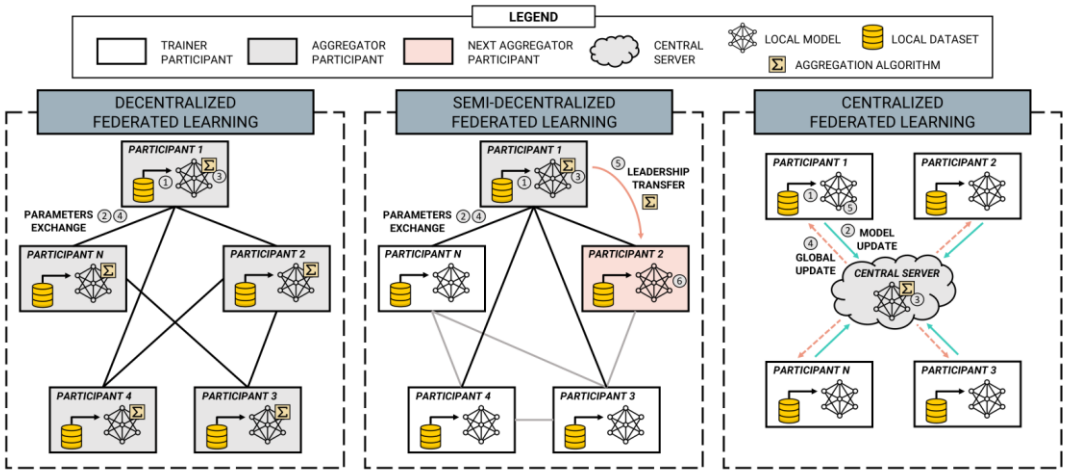


Fig. 4: Common approaches based on DFL architecture decentralization.

Figure 7: Common approaches based on DFL architecture decentralization from [8]

3.2.1.2 Security & Privacy

As explained in the introductory paper on the concept of federated learning [4]one of the goals of federated learning is to allow everyone to keep their data private. As a result, an attacker will aim at breaking this objective and succeed in accessing the data of the network participants. This guarantee of data privacy also implies that the aggregator node does not have access to the data but only to the gradient values, so it is much more difficult to prevent an attack on the model, as explained in the surveys on the subject of security in federated learning models [9] and [10]. In addition, the non-iid nature of the data makes detection of attackers more complex, as it is difficult to differentiate between an attacker who sends misleading data and a participant who would just have very different data from other network users.

Privacy-preserving federated learning is a novel approach to train machine learning models across decentralized data sources without the need to transfer data to a central location. This paradigm has become increasingly popular due to the numerous benefits it offers, including maintaining the privacy of user data, reducing communication costs, and enhancing scalability. In this state-of-the-art review, we will discuss several methods of privacy-preserving federated learning.

Homomorphic encryption is a cryptographic technique that enables computations to be performed on encrypted data. It can be used to preserve privacy in federated learning by encrypting data before sending it to the central aggregator. Several works have shown the effectiveness of homomorphic encryption in privacy-preserving federated learning. For example, in [11] authors proposed a novel algorithm that used homomorphic encryption to protect the privacy of sensitive data in federated learning.

Differential privacy is a privacy-preserving technique that adds random noise to data to make it difficult to identify individual data points. It has been used in federated learning to protect the privacy of users' data. In [12]the authors proposed a framework for privacy-preserving federated learning that used differential privacy. Their approach ensures that the central aggregator does not learn any information about individual user data.

Secure aggregation is a technique used in privacy-preserving federated learning to protect the privacy of user data by performing the aggregation of model updates in a secure and private way. In[13], the authors proposed a framework for privacy-preserving federated learning that used secure aggregation. Their approach ensured that the model updates were protected during transmission and that the central aggregator could not learn anything about the individual data sources.

Federated transfer learning is a variant of federated learning that aims to leverage the knowledge gained from multiple data sources to improve model performance. It has been used in privacy-preserving federated learning to improve model performance without compromising the privacy of individual user

data. In[14], the authors proposed a federated transfer learning approach that used secure aggregation to preserve the privacy of user data.

Multi-party computation is a cryptographic technique that allows multiple parties to jointly compute a function on their private inputs without revealing their data to each other. It has been used in privacy-preserving federated learning to protect the privacy of user data. In [15], the authors proposed a novel algorithm for privacy-preserving federated learning that used multi-party computation. Their approach ensured that the central aggregator could not learn any information about individual user data.

Privacy-preserving neural networks are a type of neural network designed to protect the privacy of user data. They have been used in federated learning to ensure that individual user data remains private while training the model. In[16], the authors proposed a privacy-preserving neural network that used differential privacy to protect user data during training.

Cryptographic primitives such as hash functions and digital signatures have been used in privacy-preserving federated learning to protect user data. In[17], the authors proposed a framework for privacy-preserving federated learning that used cryptographic primitives to ensure the privacy and security of user data during training.

In conclusion, privacy-preserving federated learning is a promising approach to train machine learning models across decentralized data sources while preserving the privacy of individual user data. In this state-of-the-art review, we have discussed several methods of privacy-preserving federated learning, including homomorphic encryption, differential privacy, secure aggregation, federated transfer learning, multi-party computation, privacy-preserving neural networks, and cryptographic primitives. These methods have shown promising results in protecting the privacy of user data in federated learning.

3.2.1.3 Attacks

According to [10], an attack can occur at 3 different stages: **before the training** (called the data and behavior auditing phase**), during the training**, and during the **prediction phase**.

Before the training, the participants of the network, and the aggregator can be attacked by a malicious individual, who would exploit a hardware or software failure. Once the attacker has taken control of the machine, he can modify the local data or change the behavior of the machine. According to [10], one of the ways to avoid attacks during the next phases (training and prediction), is to audit the network nodes and watch their behavior to see if they behave like normal or malicious nodes.

The **training phase** is the one that has undergone the most research in terms of security studies, indeed, as explained by [9], it is during this phase that attacks can do the most damage. The literature - as in [9]and] [10] - divides the attacks taking place during the training phase into 2 categories:

1. attacks on **data privacy**,
2. and attacks on the learning model (also called **poisoning attacks**)
3.

[9] categorizes 3 types of attacks on **data privacy**: membership inference attacks, property inference attacks, and data reconstruction attacks.

1. **Membership inference** attacks try to determine whether a specific sample was used to train the model.
2. **Property inference** attacks try to infer the characteristics of a dataset of an individual in the network.
3. **Data reconstruction** attacks attempts to reconstruct the data and/or the labels used during the training.

[9] categorizes **poisoning attacks** according to 2 axes: an untargeted/targeted axis, and a data poisoning/model poisoning axis. Untargeted attacks are byzantine attacks, where an attacker just wants to bring down the network or the model. Targeted attacks are attacks that try to set up a backdoor in the model, i.e. to train another (hidden) model in parallel to the global model. Data poisoning attacks try to poison the data, i.e. adding noise or flipping labels. Model poisoning attacks aim to manipulate the model updates, by changing their local objective function. It should be noted that, according to [18],

poisoning attacks in a reinforcement learning context consist of modifying reward values, or to modify the learning environment. Now we detail several attacks during the training phase.

1. ***Data poisoning attacks:*** Federated Learning is vulnerable to data poisoning attacks, such as label flipping, which can significantly impact the model's performance. In label flipping attacks, a malicious node intentionally changes the labels of some instances, either randomly or in a targeted manner, with the goal of reducing the accuracy of the global model.
   Tolpegin et al.[19], [20]nvestigated the effects of such attacks on image classification tasks using MNIST and CIFAR-10 datasets. They found that a higher percentage of malicious nodes led to more significant drops in model performance. Targeted label flipping was shown to be harder to detect and had a minor impact on the remaining classes. Attack timing and availability also played a crucial role, as the model could converge if given enough time to train with malicious participants. The attack described is known as "dirt-label," where the adversary can change the labels of the dataset. A similar attack is "clean label," where the adversary injects synthetic records to the dataset.
   Chen et al. [20], [21] investigated the clean label attack, where the attacker has no access to the original dataset but can inject synthetic records. They demonstrated the effectiveness of single key injection, where a single synthetic image with a fake label can achieve a 100% attack success rate. They also showed that blending injection strategies incorporating a desired pattern to the model could be applied with very few synthetic samples.
   Xie et al. [21] proposed a distributed backdoor attack on federated learning systems. Unlike the centralized approach used in previous studies, each adversary chooses a local trigger, which is used to form global triggers in the inference stage. Through experiments, they demonstrated that this kind of attack is more persistent than centralized scenarios.
2. ***Model poisoning attacks:*** Model poisoning attacks are implemented by an agent who modifies model updates before returning the local model to the server. This approach is different from the data poisoning task, where a malicious participant modifies the existing data or injects new data points. Regarding this case, Xingchen Zhou et al.[22] introducing an optimization-based model poisoning attack. In their work, a malicious client has 2 datasets Db and Dp of type benign, poisoning respectively. The core idea is that for the benign task when the model converges, only a specific portion of the model's neurons contribute to the prediction tasks. With this knowledge after training the model using Db these neurons can be found. After finding these neurons, constraints can be applied to them, and the model can subsequently be trained using Dp. In such a manner, the knowledge of Dp will be injected to the neurons that are not used for the main task introducing stealth into the process. Results show that this approach improves backdoor attacks significantly.
   Minghong Fang et al. [23] introduced some attack methods tailored to specific state-of-the-art aggregation algorithms with the goal of global accuracy degradation. In their work, the attacker controls c devices and knows the local training set, the code and the local models. Some also well-established defense mechanisms were tested against these attacks. Results show that the existing mechanisms are unable to defend against these attacks and the global models worsens regarding accuracy.
   Arjun Nitin Bhagoji et al. [24] construct an auxiliary dataset of size S with the following procedure: First, they sample s points from the test distribution. They then flip the label to one of the labels that is not the ground truth. The objective of the attacker is to maximize the accuracy of the trained model on the auxiliary dataset (attack accuracy), typically while ensuring that the model performance on the remaining data does not degrade significantly. The attacker is present throughout the course of training. In their work, they show that this attack is effective and can degrade the quality of the federated process as well as that modern defense mechanism need enchantment to be effective against these types of attacks.
3. **Model Stealing Attacks**: In these attacks, an attacker tries to steal the trained machine learning model by intercepting the model updates being transmitted between the devices and the central server. Model stealing attacks aim to build a model that is functionally equivalent to the victim

model. Theoretically, if the adversary can train the substitute model on all the samples in the input space, the substitute model can achieve the same performance as the victim model [25].

4. **Gradient Masking Attacks**: In these attacks, an attacker manipulates the gradients being transmitted between the devices and the central server to mask malicious data. Specifically, Gradient Masking is the practice of altering a model to hide its original gradients from an attacker. In other words, the defence methods mask the gradients of the model's output with respect to its inputs [26].
5. **Sybil Attacks**: In these attacks, an attacker creates multiple fake devices to participate in the federated learning process and manipulate the model being trained. Specifically, a system that allows clients to join and leave is susceptible to sybil attacks, in which an adversary gains influence by joining a system under multiple colluding aliases. Sybil attacks are especially prevalent in IoT sensor networks, an emerging use case for Federated Learning [27].

Finally, attacks at the level of **the prediction phase** are classified into 2 categories by [10]: evasion attacks (which try to circumvent the prediction of the model by adding some noise to a sample) and attacks on privacy (as during the training phase). For instance, **Model Inversion Attacks**: In these attacks, an attacker tries to recover sensitive information from a trained machine learning model by reversing the training process. Model Inversion Attack was first proposed by Fredrikson et al. [28] which is able to use black-box access to prediction models in order to estimate aspects of someone's genotype. Their attack works for any setting in which the sensitive feature being inferred is drawn from a small set using a trained classifier in order to extract representations of the training data. Subsequently, they develop new model inversion algorithms that can be used to infer sensitive features from decision trees hosted on ML services [29] .

In summary, while federated learning has many benefits, it is crucial to be aware of the potential security risks that arise in this collaborative framework. Data poisoning attacks, such as label flipping and clean label attacks, can significantly degrade the model's model's performance, and distributed backdoor attacks are also a cause for concern. Therefore, it is essential to develop robust defenses against these attacks to ensure the integrity and accuracy of federated learning models.

3.2.1.4 Defense

Privacy-preserving federated learning is a novel approach to train machine learning models across decentralized data sources without the need to transfer data to a central location. This paradigm has become increasingly popular due to the numerous benefits it offers, including maintaining the privacy of user data, reducing communication costs, and enhancing scalability. In this state-of-the-art review, we will discuss several methods of privacy-preserving federated learning.

To counter these attacks, presented in the previous subsection several defense methods exist. [9]presents a taxonomy of defense techniques and classifies them in 3 categories: defenses at the **aggregator server level**, defenses **at the client node level** and defenses at the **communication channel level.**

Server level defenses are:

1. **Secure aggregation** is a technique used in privacy-preserving federated learning to protect the privacy of user data by performing the aggregation of model updates in a secure and private way. In [13], Google proposed a secure aggregation algorithm that securely aggregates the clients' updates by using SMC to compute the weighted averages of received updates. Upon receiving a sufficient number of clients' updates, the FL server can decrypt the average update. This is possible because client updates are transferred via additive secret sharing. As a consequence, the clients' private data are protected. In the same direction, Xu et al. proposed a privacy-preserving approach [30], called VerifyNet, that aim to realize secure gradient aggregation and verification. This approach employs a double-masking method (based on Shamir's secret sharing and homomorphic hash function) making it difficult for malicious adversaries to infer training data.
2. **Monitoring and Auditing**, **Anomaly detection:** Anomaly detection tries to detect malicious nodes by looking at the model updates and excluding those that appear anomalous (those that deviate from the distribution). Regular monitoring and auditing of federated

learning systems is necessary to detect and prevent any security or privacy breaches. As most of the defense strategies are based on malicious worker detection, there is an urgency to prepare the federated learning process for monitoring and eventually behavioral pattern analysis. In fact, without timely and dynamic monitoring methods, the coordinator cannot detect and remove the malicious or unreliable workers from the system. In the literature there are some works [31] [32]for monitoring and auditing the federated learning systems; the first one visualizes the policy sequence of model in order to understand the temporal patterns of the attacks, while in second one a monitoring process are developed during the early stages of training in FL that are based on behavioral pattern analysis to eliminate potential malicious workers from the FL process.

3. **Access control**: Proper access control mechanisms should be put in place to ensure that only authorized parties can participate in federated learning. Scanning all sources of vulnerabilities and tightening defenses are thus mandatory steps needed to build up a federated learning-based system while ensuring the security and privacy of the data form curious/malicious attackers [33] .
4. Modification of the **learning rate**, by modifying at each aggregation round the learning rate,
5. **« less is more »,** which is a family of compression techniques, in which we will remove some redundant weights from the neural network.
6. **Model protection**: The machine learning models created during federated learning should be protected from tampering or reverse engineering to prevent malicious actors from using them for malicious purposes. In order to keep attackers from tampering with local models, Kalapaaking et al. proposed a blockchain-based Federated Learning framework [34] with an Intel Software Guard Extension (SGX)-based Trusted Execution Environment to securely aggregate local models in Industrial Internet of Things.

The concerns and techniques at the **client node** are:

1. **Data privacy:** To prevent sensitive information from being leaked, the data used in federated learning should be anonymized, sanitized or differential privacy techniques should be used. FL emerges a new approach to tackle data privacy challenges in ML-based applications by decoupling of data storage and processing (i.e., local model training) at end-users' devices (i.e., local nodes) and the aggregation of a global ML model at a service provider's server (i.e., a coordination server). So, only locally trained model parameters, which contain the essential amount of information required to update the global model, are shared with a coordination server. Nevertheless, such model parameters still enclose some sensitive features that can be exploited to reconstruct or to infer related personal information [35].
2. **Differential Privacy**: Differential privacy techniques are widely used to protect the privacy of individuals in federated learning systems. The main concept is to preserve privacy by adding noise to sensitive personal attributes. In FL, differential privacy is introduced to add noise to clients' uploaded parameters in order to avoid inverse data retrievals. For instance, the DPFedAvgGAN framework [36]uses DP to make GAN-based attacks inefficient in training data inference of other users for FL-specific environments. Additionally, Ghazi et al. improve privacy guarantees of the FL model by combining the shuffling technique with differential privacy and masking user data with an invisibility cloak algorithm [37]. Moreover, Hema Priya et el. develop an Advanced Encryption Standard (AES) algorithm with Differential [38] privacy giving 97.3% accuracy. In [12], the authors proposed a framework for privacy-preserving federated learning that used differential privacy. Their approach ensures that the central aggregator does not learn any information about individual user data. Furthermore, the Private Aggregation of Teacher Ensembles (PATE) [39]suggest a distributed training framework with strong privacy guarantees, based on DP. In PATE, an ensemble of "teacher" models are trained individually on their private datasets and then

their knowledge is transferred to a "student" model by noisy aggregation of teachers' answer to public data input. In this setup, DP aims to to limit the effect of any single sensitive data item on the student's learning.

Finally, we have the **communication channel** defenses, which are classified into two categories: those based on **Encryption**, and those based on secure multi-party computation (SMPC).

1. **Encryption:** All data used in federated learning should be encrypted to protect against unauthorized access or eavesdropping. A number of defensive approaches for Federated Learning security have been presented in the literature [40], each of which is proposed to address different security/privacy and performance/accuracy objectives. For example, if the encryption level in Secure Multi-Party Computation (SMC-based) mechanisms or the quantity of noise in DP-based mechanisms is not enough, the clients that participate to the FL process still suffer from the risk of privacy leakage. On the other hand, if the encryption level is too high or too much noise added to the exchanged updates, the FL model severally suffers from low accuracy. Research is ongoing to develop advanced encryption techniques that can be used to secure federated learning systems. Homomorphic encryption is a cryptographic technique that enables computations to be performed on encrypted data. It can be used to preserve privacy in federated learning by encrypting data before sending it to the central aggregator. Several works have shown the effectiveness of homomorphic encryption in privacy-preserving federated learning. For example, in [11], the authors proposed a novel algorithm that used homomorphic encryption to protect the privacy of sensitive data in federated learning. Also, Febrianti Wibawa et al. [41]propose a privacy-preserving federated learning algorithm for medical data using homomorphic encryption. The proposed algorithm uses a secure multi-party computation protocol to protect the deep learning model from the adversaries. In addition, Jaehyoung Park et al. employ a homomorphic encryption (HE) scheme [42]that can directly perform arithmetic operations on ciphertexts without decryption to protect the model parameters. Using the HE scheme, the proposed privacy-preserving federated learning (PPFL) algorithm enables the centralized server to aggregate encrypted local model parameters without decryption.
2. **Multi-party computation** is a cryptographic technique that allows multiple parties to jointly compute a function on their private inputs without revealing their data to each other. It has been used in privacy-preserving federated learning to protect the privacy of user data. In [15] , the authors proposed a novel algorithm for privacy-preserving federated learning that used multi-party computation. Their approach ensured that the central aggregator could not learn any information about individual user data.

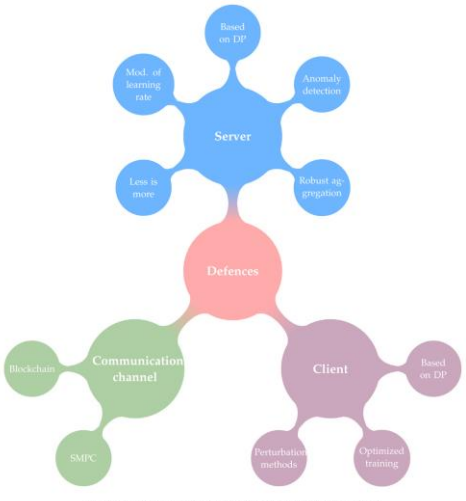


Fig. 9. Representation of the taxonomy of defences against adversarial attacks.

Figure 8: Representation of the taxonomy of defences against adversarial attacks from [8]

As we can see in figure 8 below, taken from the survey [9], not all defenses allow to protect against all attacks, some defenses are more adapted to some types of attacks. Generally speaking, defenses at the server level seek to prevent poison attacks while attacks at the client node seek to prevent privacy attacks. According to [9], differential privacy techniques are effective against privacy attacks and also provide some protection against poisoning attacks.

In conclusion, privacy-preserving federated learning is a promising approach to train machine learning models across decentralized data sources while preserving the privacy of individual user data.

3.2.1.5 Other developments

1. **Federated Distillation**: Federated Distillation is a recent development in federated learning that allows for the transfer of knowledge from a teacher model to multiple student models in a federated setting. In this direction, Li and Wang [43] proposed an algorithm of federated distillation called FedMD. In this algorithm, the authors seek to transfer knowledge from a fully trained model to a smaller model. Typically, the knowledge is a pre-trained model's logit, transferred to a small model for compression. The knowledge can also be a collection of other small models' logits, in that the collection of forecasts is often more accurate than individual predictions. Finally, Eunjeong Jeong et al.[44] present a federated distillation based on exchanging only the local model outputs whose dimensions are commonly much smaller than the model sizes minimizing its own loss function.
2. **Federated Transfer Learning**: This is a recent development in federated learning that allows multiple parties to train models on their own datasets while sharing the knowledge gained from the models with other parties. Federated Transfer Learning takes inspiration from transfer learning, a paradigm already popular in image analysis [45] [46].In this setting, machine learning models trained on a large dataset for one problem/domain is applied to a different but related problem/domain [47] [45]The performance of transfer learning is strongly dependent on interrelation between different domains. While talking about federated learning, stakeholders in the same data federation are usually organizations from the same industry. Thus, it is suitable to apply transfer learning in the federated learning framework [48].
3. **Privacy-preserving neural networks** are a type of neural network designed to protect the privacy of user data. They have been used in federated learning to ensure that individual user data remains private while training the model. In [16], the authors proposed a privacy-preserving neural network that used differential privacy to protect user data during training.
4. **Cryptographic primitives** such as hash functions and digital signatures have been used in privacy-preserving federated learning to protect user data. In [17], the authors proposed a framework for privacy-preserving federated learning that used cryptographic primitives to ensure the privacy and security of user data during training.

3.2.2 Reinforcement Learning

Many approaches exist to address the RL problem, which results in different classes of algorithms.

Model-based RL aims at exploiting a representation of the model of the dynamics and the reward to help predicting the next states and rewards and planning the best actions. This model is either known in advance or learned. Optimal control can be seen as an extreme case of model-based RL where the learning part is nonexistent. Dynamic Programming (DP) is a technique usually used when the state and action spaces are low dimensional. Otherwise, planning heuristics such as Monte-Carlo Tree Search (MCTS) are exploited to assess long-term rewards (AlphaGo, AlphaZero). One may also simultaneously learn an initially unknown model of the environment with trial-and-error and exploit this model to plan the optimal actions (MuZero, Dreamer).

On the other side, **model-free RL** accommodates with not representing the environment model. It learns to select actions that optimize expected future rewards from state-action-reward transitions alone. Model-free methods exploit trial-and-error to gradually learn how to predict the expected return, as opposed to model-based methods that exploit the model for this aspect. Model-free methods have less

pre-requisites than model-based methods but need usually more interactions with the environment to converge. Classical methods involve TD3 [49] , PPO [50].

The standard RL framework involves one agent interacting with the environment. When multiple agents share the same environment in which they potentially collaborate or compete, we talk about **Multi-agent RL (MARL)** [51]. Each agent typically possesses some kind of independence with respect to the other agents (otherwise, it would simply be classical single agent RL). For instance, each agent could only possess a limited observation of the complete environment. In addition, the actions of each agent are decided locally in a decentralized fashion.

In terms of learning in the MARL setting, while the action decision/execution in MARL is by definition decentralized, the learning can be itself centralized or decentralized. When the learning is centralized, during the training phase, all agents can communicate freely and access the full state. However, the learned polices have to meet the constraint of partial observability and decentralized execution. Centralizing the learning makes the problem simpler and corresponds to many use cases. The alternative learning scenario is the decentralized one where all agents learn under partial observability and decentralized execution conditions. In that scenario, the baseline is to have each agent consider the rest of the agents as part of the environment, which is called independent learning.

The multi-agent setting is motivated by problems where the real world imposes some communication or sensory constraints that force the agents to act in a decentralized way once deployed in the world. However, in some cases, even if there are no such constraints in the world inherently, the decentralized execution and partial observation constraints can be imposed artificially in order to force a meaningful decomposition of the problem with the hope of making the learning problem tractable this way.

The extension of single to many agents comes with new challenges. First, the complexity of the problem increases with the number of agents. Moreover, at the agent level, the problem becomes non-stationary as the environment can now evolve based on, not only its actions but also other agents' actions. If the environment is cooperative, the agents need to understand how their actions can benefit the overall achievement of the global task, if the agent competes then on top of RL techniques agent should be designed with tools from the game theory literature.

3.2.3 Federated Reinforcement Learning

The main reference for a review of the literature on Federated Reinforcement Learning is [18]. It lists many views of the problem and applications. We will reuse some of its structures in this document.

3.2.3.1 Methodology

As in [18],we can distinguish two categories of CF-DRL, described in the following subsections.

3.2.3.1.1 Horizontal Federated Reinforcement learning (HFRL)

HFRL is at one hand of the spectrum where one entity in the federation possesses a dataset of experiences with their environment (interaction history/replay-buffer) that is independent of the datasets of the other entities in the federation. However, do all the entities interact with exact copy of the same environment?

An example of HFRL is the autonomous driving system. A fleet of vehicles drives on roads in various conditions and trains train their models locally. All vehicles have the same type of observation, actions and global environment but cannot gather alone enough data to anticipate all situations, so they want to learn collectively. They can use HFRL to share their experience without breaking the privacy of their sensitive data and learn together a shared model.

In [52], a HFRL architecture is proposed, with an additional layer of clustering some of the entities into clusters if they need to solve similar task. Therefore, they can further personalize the policy of each entity to a specific task. Experiments are done in Atari game Pong. In [53] in the context of reinforcement learning for robot navigation, transfer methods are used to share the experience of the robots so as to learn a shared model (potentially with a new NN architecture) from annotated examples

generated by the entities along with a confidence score. In [54] suggest the use of FRL architecture between edge nodes and IoT devices for computation offloading tasks with a simple model aggregator.

Note that this setting is related to distributed RL, which looks at a similar setting without having the privacy preserving constraint. In the field of distributed RL reference work are A3C [55], Impala [56], Ape-X [57] and Gorila [58].

3.2.3.1.2 Vertical Federated Reinforcement Learning (VFRL)

In VFRL the entities all live in the same environment but have different views of it their dataset of experiences is dependent on each other.

An example of VFRL extracted from [18], is a microgrid system that includes household users, the power company, and the photovoltaic (PV) management company as the entities. All the entities observe the same environment. The global state of the microgrid system generally consists of several dimensions/features, i.e., State-of-Charge (SOC) of the batteries, load consumption of the household users, power generation from PV, etc. The household agents can obtain the SOC of their own batteries and their own load consumption, the power company can know the load consumption of all the users, and PV management company can know the power generation of PV. In order to learn the optimal policies, these agents need to communicate with each other to share their observations without breaking the privacy.

There are fewer works in VFRL. In [59] [60]the agents build a shared value network. However, the setting is limited to two agents and the second one cannot build its own policies and rewards. It is applied in two different games, such as Grid-World and Text2Action.

Note that this setting is related to multi agent RL, which looks at a similar setting without having the privacy preserving constraint. In the field of multi-agent RL reference work is for instance QMIX [59] . To allow multi-agent RL methods to be brought in VFRL, one needs to make sure that these methods are designed for the partially observable setting as it is the case in VFRL as each entity only has a limited view of the environment.

3.2.3.2 ML/FL software and tools

3.2.3.2.1 CFDRL code

Pysyft performs numpy-like analysis on data that remains in someone else's server. It is bases on torch and has a **RL** tutorial.

Byzantine-Federeated-RL GitHub – flint-xf-fan/Byzantine-Federeated-RL: Code for paper [61]

FeReD GitHub – sotostzam/FeReD: (in C) Code for the paper [62]

3.2.3.2.2 Other FL frameworks

Fed-BioMed: Fed-BioMed is an open source project focused on empowering biomedical research using non-centralized approaches for statistical analysis and machine learning. The project is currently based on Python, PyTorch and Scikit-learn, and enables developing and deploying federated learning analysis in real-world machine learning applications. Fedbiomed.gitlabpages.inria.fr

EasyFL GitHub – EasyFL-AI/EasyFL: An easy-to-use federated learning platform

- FATE (Federated AI Technology Enabler): FATE (Federated AI Technology Enabler) is the first production-oriented platform developed by Webanks AI Department. Its goal is to support a collaborative and distributed AI ecosystem with cross-silo data applications while meeting compliance and security requirements.
- TensorFlow Federated: is an open-source framework for machine learning and other computations on decentralized data.
- PaddleFL is an open source federated learning framework based on PaddlePaddle.
- NVIDIA FLARE (NVIDIA Federated Learning Application Runtime Environment) is a domain-agnostic, open-source, and extensible SDK for Federated Learning.

- FLOWER: A unified approach to **federated learning**, analytics, and evaluation. Federate any workload, any ML framework, and any programming language.
- PATE /FED-PATE Differentially Private Federated Learning with Knowledge Transfer

3.2.3.2.3 Thales internal software

SAFERLEARN: Secure Federated model framework - Enterprise Edition (close source). Limitations for implementation: * Granted on a royalty-free basis, * Access rights to the source code of this software needed for implementation only shall be detailed in a separate license agreement. Limitations for exploitation: * Granted on Fair and Reasonable conditions.

DAD: The Distributed Anomaly Detection (DAD) is a tool responsible for detecting anomalies in network/SCADA communications. No access rights are granted for the source code.

3.2.3.2.4 ML platforms

ML Flow: MLflow is an open source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

KubeFlow: Kubeflow is an open-source platform for machine learning and MLOps on Kubernetes introduced by Google.

3.2.4 GAN dataset generation for FL poisoning attacks

Generative Adversarial Networks (GANs) have been widely used for image and data generation tasks. GANs have been proposed by Ian Goodfellow [63]as a generative model estimation procedure, which allows the generation of synthetic data, which have the same characteristics with the data used in the training process. GANs are composed of two competing neural networks. One of them is called the "Generator" which creates synthetic data through random noise, while the second one is the "Discriminator" which receives as input the generated data and classifies them as real or fake. The Discriminator's output is fed back to the Generator, so that the Generator's weights can be updated though backpropagation over the whole system. Eventually, the generated output becomes realistic enough to deceive the Discriminator, which means that the Generator has been trained successfully.

GANs span wide spectrum of application. In particular, GANs address the deficiency of training datasets, allowing expanding them through realistic synthetic ones. Moreover, the authors in [64]use GANs in order to be able to have different model architecture among FL participants. In addition, an FL framework for GAN is proposed in [65] which allows collaborative training for the generator across cloud and edge resources, making it more appropriate for GAN applications in a distributed meta-OS environment.

However, GANs can also be used for malicious purposes, such as realizing membership or poisoning attacks in federated learning (FL). Indicatively, GANs can be used to generate malicious, yet realistic, data, with altered labels. Then, an attack may be realized by injecting them into the local real datasets of participating clients in a Federated Learning system, thus poisoning the global model. This type of attack, known as GAN-based data poisoning attack, can cause severe damage to the performance and accuracy of the eventual global model produced during federated learning training. Indicatively, the authors in [66] consider a Federated Learning system, which suffers from a GAN based data poising attack. One of the FL participants initially acts as benign, while stealthily training a GAN to mimic prototypical samples of its peers' training set, which the attacker has no access to. Then, these generated samples are used to generate the poisoning updates, and, thus, compromise the global model by uploading the scaled poisoning updates to the server. Moreover, the accuracy of a jointly trained model via FL is jeopardized via synthetic samples, leveraging the poisoning attack algorithm Fed-MIFGSM proposed in [67]

Therefore, it is of paramount importance for NEMO to study and develop defences against such sophisticated generative-based attacks in FL systems.

3.2.5 Decentralized learning approach and security

Gossip Learning

In gossip learning, similar privacy and poisoning concerns arise. Ensuring that malicious nodes do not compromise data privacy or inject poisoned information into the network's shared updates is more complex as the network is totally distributed, without a central actor.

Gossip learning [68] in contrast to traditional federated learning, is a decentralized machine learning paradigm that emphasizes peer-to-peer communication among nodes within a network. In federated learning, a central server orchestrates the model updates from various distributed devices or nodes, whereas gossip learning relies on nodes periodically sharing model updates and information directly with a subset of their peers.

Advantages of Gossip Learning:

1. Decentralization: Gossip learning eliminates the need for a central server, which can enhance system robustness and reduce the risk of single points of failure.
2. Scalability: Gossip learning can be highly scalable as nodes communicate directly with one another, which is particularly useful in large-scale networks
3. Privacy Preservation: Gossip learning often enables better privacy preservation since data never leaves the local node, reducing the risk of data leakage.
4. Low Communication Overhead: Communication between nodes typically involves a smaller subset of peers, reducing the overall communication overhead compared to federated learning.

Disadvantages of Gossip Learning:

1. Slower Convergence: Gossip learning may converge more slowly compared to federated learning, as information propagates in a more decentralized and probabilistic manner.
2. Synchronization Challenges: Ensuring consistent model updates and synchronization among nodes can be challenging, particularly in networks with varying degrees of connectivity.
3. Complexity: Implementing and managing gossip learning can be more complex than traditional federated learning due to decentralized control and potential synchronization issues.

Our goal is to design an architecture that can at the same time be secure against privacy and poisoning attacks, and at the same time with an efficient learning process and a quick convergence time. In the pursuit of heightened security, cryptographic techniques, Byzantine fault tolerance mechanisms, and outlier detection algorithms are often integrated, which can add computational overhead and latency. These extra computational demands can slow down the learning process, potentially impeding the timely convergence of models and the overall system performance.

Moreover, the stringent privacy-preserving measures required to thwart privacy and poisoning attacks can restrict the depth of information shared between nodes, potentially limiting the quality of model updates. Striking the right balance between privacy and utility is a delicate task, as overly stringent privacy measures may hinder the model's ability to extract meaningful insights from distributed data sources.

For the simulations, we will use the **Gossipy** simulator[18].

[18] https://github.com/makgyver/gossipy

3.3 Architecture & Approach

Logical view

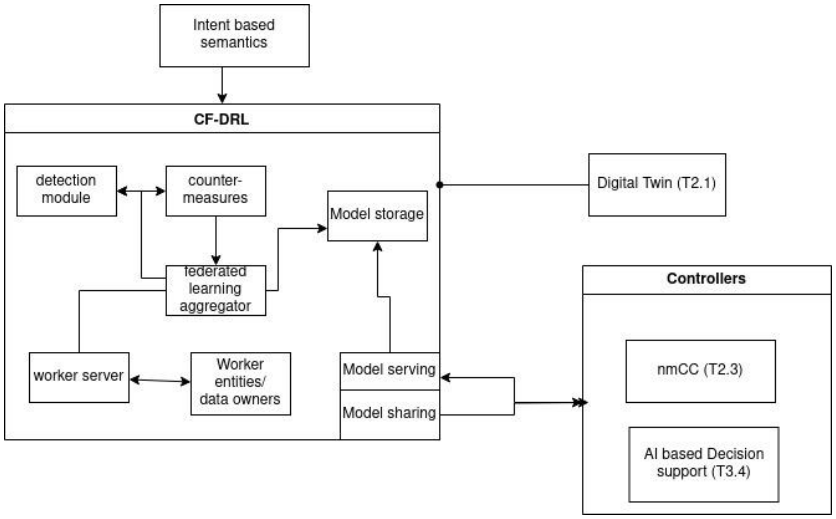


Figure 9: Logical view of CFDRL

We decompose the CFDRL into the following sub-components as illustrated on Figure 9:

- Worker Entity/Data Owners: These are the entities that are each of them owing data and learning from them. Their main features are their ability to learn from their data, send parameters of their model to the worker server and to receive new aggregated model from the worker server.
- Federated learning aggregator: It aggregates the model sent by the entities and send the aggregated model back to the data owners/Worker entities. It communicates with the detection module to detect attacks and might implement countermeasure proposed in case of detected attacks.
- Worker server: The worker server manages the registration of entities and communications between the aggregator and the entities/data owners.
- Detection modules: They scan the models exchanged and raise alerts in case of detected attacks. These alerts are sent to the counter measure module.
- The Counter Measure module: In case of raised alert the counter measure model, the counter measure model proposes counter measures.
- Model storage: This module stores the learned models.
- Model serving model sharing: The models can either be shared completely or queried for a specific action in a given state.

Process view

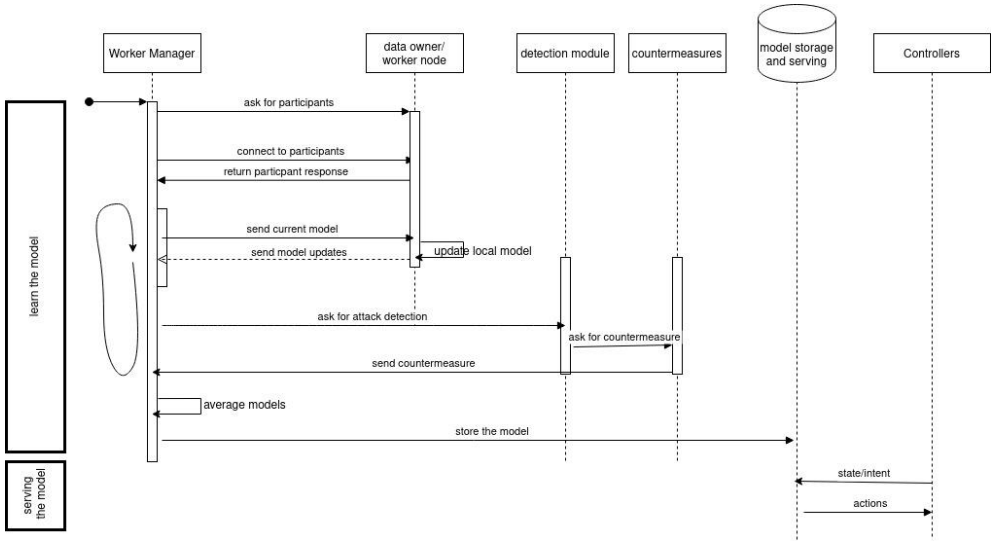


Figure 10: Process View of CFDRL

We envision that the CFDRL would process in the following way as illustrated in Figure 10. The process is divided into two phases. The learning phase during which the model is learned and the serving phase where the model is served.

During the learning phase, first the worker nodes can ask and then be registered as entities in the learning process. The main loop sees the worker nodes send their model parameters to the worker manager that, in turns aggregates the model and sends them back to all the entities. The worker manager communicates with the detection module to detect the attacks. In case of attacks the countermeasure are proposed by the countermeasure component.

In the serving phase, the model is either served or shared.

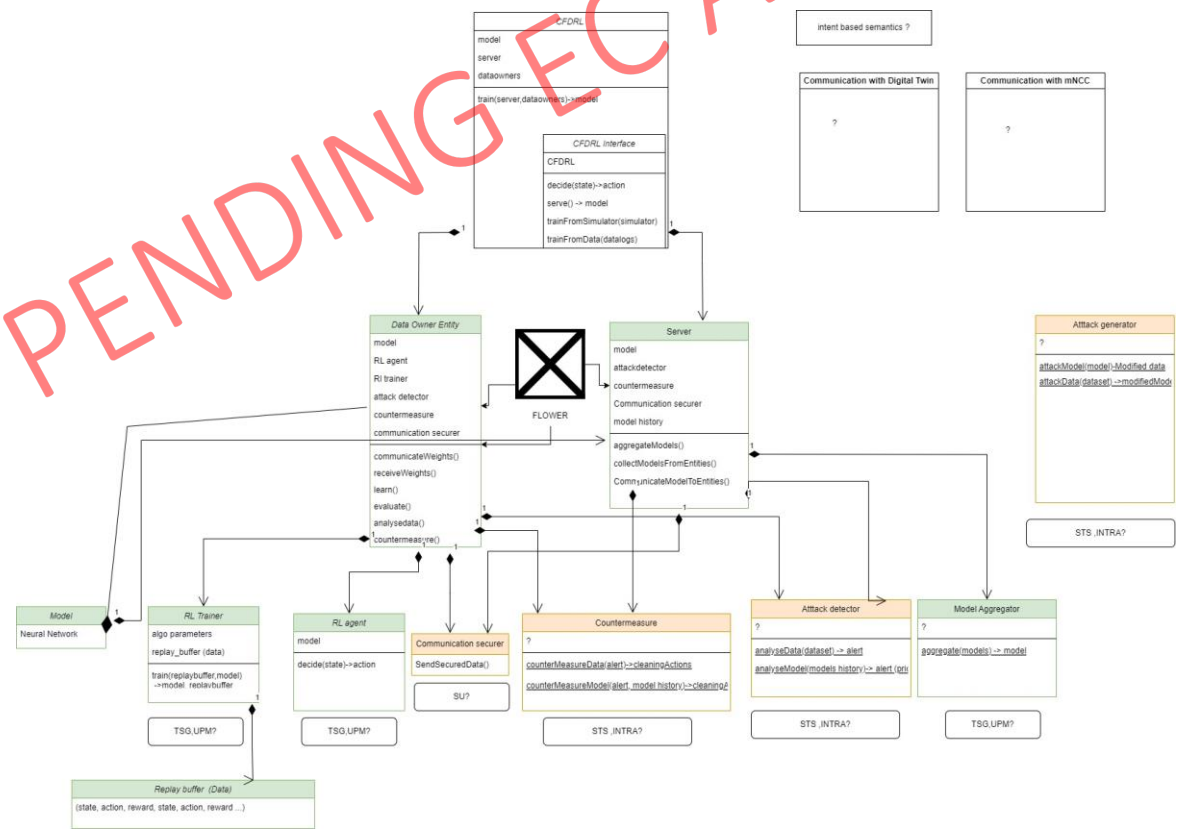


Figure 11: Development view of CFDRL

In Figure 11, we give a more detailed development view of the component.

We have investigated the use of several federated learning paradigms. Our current implementation is using Flower (https://flower.dev/)

In Table 3, we list the functionalities of the blocks in the component.

Component	Functionalities
Server	• aggregateModel() • collectModelsFromEntities() communicateModelToEntities()
Worker Node / Data Owners	• communicateWeights() • receiveWeights() • learn() • evaluate() • analyseData() countermeasure()
RL Trainer	train(rl_agent, training_steps)
RL Agent	decide()
Model Aggregator	aggregate()
Attack Generator	attack_model()
Attack Detector	• analyseData() analyseModel()
Communication securer	sendSecureData()

Table 3: CFDRL functionalities

3.3.1 Description of sub-components

3.3.1.1 RL Trainer

- The RL Trainer is in charge of training the RL agent (that contains the neural network model).
- It is coded in Python and based on the code provided by Thales in the TheRLib library that contains algorithm such as DQN TD3.
- The RL trainer takes as input a configuration file (JSON file) with hyper-parameter values for the learning algorithms. Its main method is the train method that takes as input the rl agent and the number of learning steps to run. It then updates the model of the RL_agent by running the input number of steps of the algorithm.
- It is open-source.

3.3.1.2 Flower Client /Worker node

- The worker node is in charge of managing a node that is learning a model based on its own data and models shared and communicated between nodes through the server nodes. This component needs to manager both the communication with the server and to trains its model. This component is therefore composed of a RL Trainer and a RL agent.
- There are various frameworks for orchestrating this type of federated collaborative learning through a server. The most famous in Python are Nvidia Flare, Pysyft and Flower. After testing them we have decided to Flower for its simplicity and effectiveness.
- The Interface of the client is composed of three main functions which are to send the parameters of the current model to the server, to receive the model from the server and update its current model and finally to update its current model based on its learning procedure (Rl agent and RL Trainer).

3.3.2 Interaction with other NEMO components

While the discussion on the applications to the Living labs is in early stages we investigated the link to the meta-orchestrator.

3.3.2.1 Meta orchestrator

We are currently investigating the use and utility of the CFDRL with the Meta-orchestrator in Task 3.4 of NEMO.

RL for the Meta-Orchestrator (MO): RL solves control task on dynamic environment over time. The Meta Orchestrator is a dynamic environment that would fit the formalism of the task that RL can solve. More precisely, we need to decompose the MO into states (observation of the agent), actions (of the agent), and reward. One this point, the state would be characteristics describing the micro-services, actions would be linked to migration, placement and scaling and reward would be links to migration time, downtime and overhead time. This description will be made more precise as the T3.4 unrolls. The question of whether the MO is single agent or multi agent is not decided. In addition, we started to a do SOTA research about this FRL-MO combination) among which we list some interesting reference below:

- DRS: A Deep Reinforcement Learning enhanced Kubernetes Scheduler for Microservice-based System Zhaolong Jian et al, Nankai University College of Computer Science January 4, 2023
- Deep reinforcement learning-based microservice selection in mobile edge computing, Feiyan Guo et al 2022, Cluster Computing.
- Online Microservice Orchestration for IoT via Multi-objective Deep Reinforcement Learning, Yu et al, IEEE Internet of Things Journal, 2022.

Federated Learning for the Meta-Orchestrator: The question of whether Federated Learning is applicable to MO remains open. The presence of local orchestrators in the formulation of the MO hints that a connection might be possible.

Data to be exchanged between CFDRL and MO: Typically, the state action, next state and reward information are supposed to be vector of scalar. Can MO will investigate how to provide this type of data. For RL to work, we need data of that are sequence of tuples (state, action next state, reward). T3.4 will investigate the availability of such data. T3.4 might be able to provide a simulator of the MO so that the RL algorithm could automatically generates its own data. NN model are good candidates for MO.

API: Need to decide the framework such as Rest API/ GRCP /MTUPT.

The CFRDL expects either:

- Data of the form of sequence of tuples (state, action next state, reward).
- A simulator.

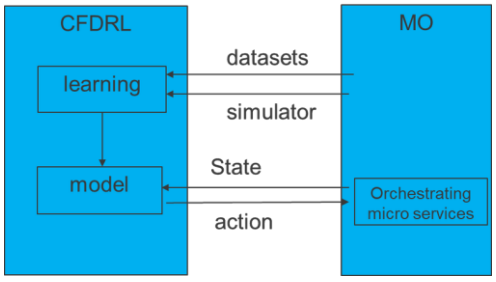


Figure 12: Interface between the CFDRL and the Meta orchestrator

The interface between the CFDRL and the Meta-orchestrator will be either of model sharing or model serving. As illustrated in Figure 12 once the model is learned by the CFDRL, the model can be queried by the MO with states as a request and orchestration actions will be served by the CFDRL.

3.4 Technology used and advanced

3.4.1 Federated Reinforcement Learning

We have implemented a proof of concept of federated reinforcement learning.

As defined above we chose to implement Horizontal RL, which means that all the distributed entities learn the same model for the same single-agent RL task.

We used the framework Flower[19] which provides a unified approach to federated learning in Python. Flower provides us with a core implementation of the Server and the Client/Data owner sides and runs automatically the interaction loop between the server and the clients. In our case we especially modified the client side to add RL training functionalities. The Flower client is decomposed into four main functions: *get_parameters* is the function that send the parameters of the local model to the server, *set_parameters* is the function that collect the updated parameters from the server and copy them in the local model, *fit* is the function that train and update the local model with the RL algorithm (see below for more detail), and finally evaluate is the function that evaluates the model and provide metrics on its performances.

The RL task that was used is the acrobat task from the Gym environment, which is a Classic Control environment. The system has two connected segments with the end point of one of them being locked. The goal is to apply torques on the actuated joint to swing the free end of the chain above a given height while starting from the initial state of hanging downwards. This problem has a discrete action space of size 3 (right torque, no torque, left torque) and observation space of 6 dimensions being the angles and their velocity.

The Model used is Multi-Layer Perceptron (fully connected) with 3 layers where the input is a concatenation of the state and the action, and the output is an estimation of the value of performing this action in that state followed by the estimated optimal policy.

The base RL algorithm is DQN (Playing Atari with Deep Reinforcement Learning, V Mnih et. al., 2013) an algorithm that learns a neural network model based on a replay buffer of interactions with the environment.

The federated learning aggregation is the Vanilla federated averaging (McMahan et al., 2017).

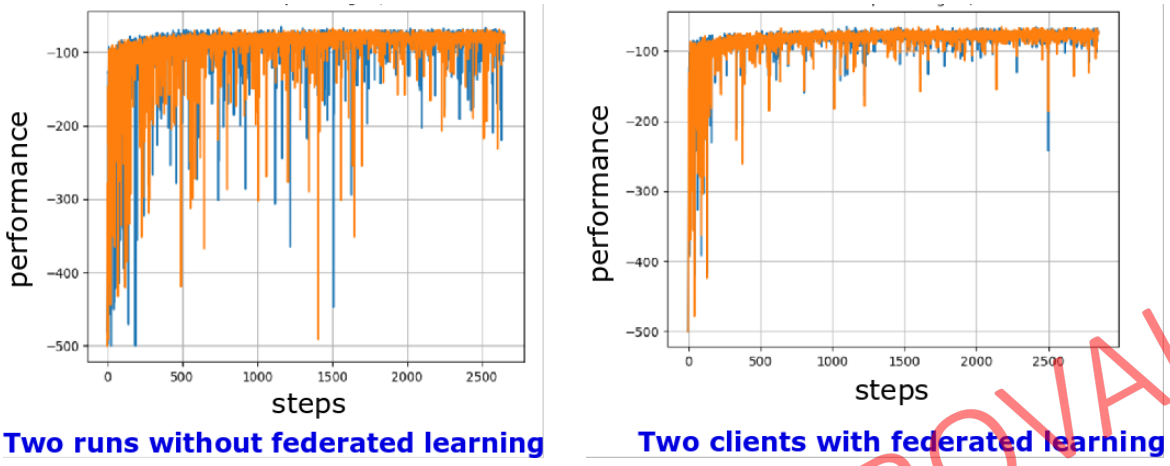
Regarding more recently algorithms beyond DQN, there were notable advancements in DRL. They include algorithms like Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic (A3C) (Asynchronous Methods for Deep Reinforcement Learning, Mnih et. al., 2016), and Soft Actor-Critic (SAC) (Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et. al., 2018). A2C and A3C integrate value estimation and policy optimization into a unified framework, with A3C leveraging parallelism for faster training. In contrast, SAC addresses exploration challenges in continuous action spaces through soft value functions and entropy regularization. These algorithms provide enhanced sample efficiency, stability, and adaptability compared to traditional DQN, potentially improving our RL model's performance in complex environments.

[19] (Flower: A Friendly Federated Learning Framework)

Our current results are the comparison of the run of the RL learning with or without model sharing through the server. We use 2 clients and compare the speed and stability of learning between the two following setting:

- the two clients do not aggregate their model (Figure 13 (a))
- the two clients aggregate their model (Figure 13 (b))

Figure 13 shows that in the acrobot environment the learning speed is not impacted but the model sharing but model sharing brings more stability of the learning as in the right figure we do not see massive performance drop as in the case without federated learning.



(a) (b)

Figura 13 : Comparing learning curves

3.4.2 Privacy Preserving Federated Learning

Despite its intricate aim for keeping data privacy, Federated Learning does not inherently support privacy preservation features which would indeed guarantee that data privacy is respected during the whole lifecycle of training within FL systems. In addition, knowledge aggregation and transfer remain big burdens in distributed diverse environments of cloud, edge and IoT devices, especially in the context of a multi-cluster setup found in the meta-OS. In NEMO, we suggest privacy preserving enhancements for knowledge aggregation and transfer beyond FL, based on the technological options for CF-DRL.

The core FL framework selected for CFDRL is Flower, an open-source FL framework addressing FL node diversity in the meta-OS infrastructure. Indeed, Flower supports real FL deployment across a number of clients, which may span from a handful of IoT devices and mobile phones to numerous clients. It has been engineered with specific design objectives in mind, including compatibility with leading machine learning frameworks like PyTorch, TensorFlow, and Keras, as well as interoperability across various operating systems, programming languages, and hardware types (including IoT devices, mobile phones, and servers) among the clients in the federated learning system. It is also designed to seamlessly scale to handle many clients. However, it is worth noting that Flower does not provide inherent support for privacy preservation, but rather supports Differential Privacy (DP) only in experimental mode[20]

In order to address privacy preservation challenges in knowledge aggregation and transfer, and, subsequently, in CF-DRL, NEMO introduces FREDY (Federated Resilience Enhanced with Differential Privacy) which integrates Flower with PATE to bolster privacy features. Much like the PATE framework, FREDY implements a teacher-student scheme within its architecture. The teachers are trained within a federated learning framework, and upon completion of the federated learning process, each teacher's model is employed for making predictions on the student's public data. These predictions,

[20] https://flower.dev/docs/framework/explanation-differential-privacy.html

generated from the student's publicly available data, are collectively aggregated and subjected to the addition of noise to facilitate the training of the ultimate federated learning framework. Our system is composed of a Flower Server and multiple Flower Clients, each possessing its own local and private datasets. In the context of the federated learning training process, both the teachers and the student function as clients. The Flower Server plays a pivotal role in training the teacher models and subsequently takes responsibility for the noisy aggregation of the outputs from these teachers, enabling the labelling of the student's public, yet unlabelled, dataset.

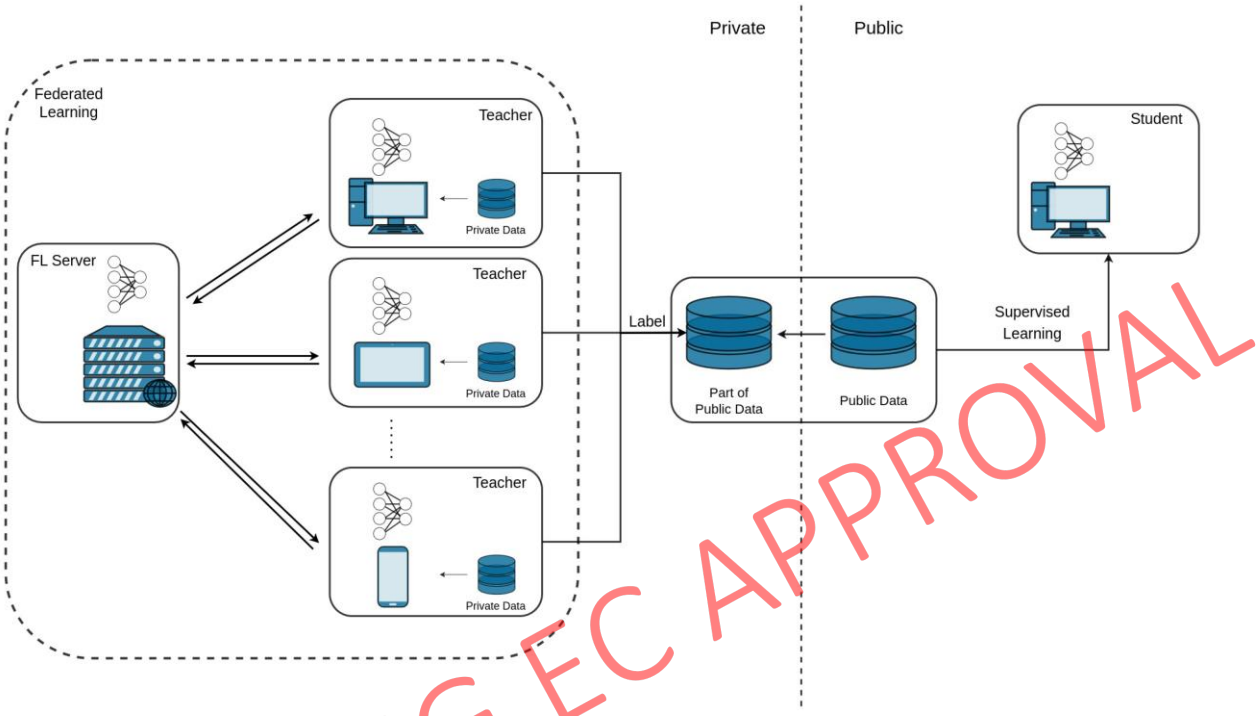


Figure 14: FREDY (Federated Resilience Enhanced with Differential Privacy) framework for privacy preserving

An overview of our approach can be viewed on Figure 14. Here, we see that the teachers play the role of the federated learning clients and are trained using local data. After they are trained, they are used to create labels for the student which has public unlabelled data. Using the teachers votes the student trains a model with the labelled data which is resilient to inference attacks due to the noise added to the teacher votes.

3.4.3 Generative Adversarial Network attack dataset generation

The first step towards the cybersecurity protection of a Federated Learning system adheres to the formulation of a simulation attack scenario where some clients are malicious, and some are benign. Having this FL setup as a starting point then we can proceed identifying common attack scenarios within the FL framework and engineer appropriate defenses or evaluate relevant metrics (such as malicious attacker capacity of the network etc.). Consequently, we create synthetic attacks within a FL environment which resemble real world applications. The framework that is used to setup our paradigm is Flower in accordance with CF-DRL options. Flower provides a comprehensive solution where we can manipulate low lever FL parameters such as model weights and aggregation methods. Regarding the synthetic attacks, we want to simulate a scenario where a federated learning client can generate and manipulate network level logs and assign a label to them arbitrarily. This is intended to work as a label flipping attack confusing the global model and degrading its accuracy.

An overview of our method can be viewed in Figure 15. Here, we observe a federated learning system with three clients. Among these clients, two are benign (green) and one is malicious (red). The benign

clients possess their local log data using which the local model is trained and transmit the result back to the server. Concerning the malicious client, the local data are used to train a Generative Adversarial Network (GAN) which can subsequently output plausible log data. A machine learning model known as a "generative adversarial network" (GAN) consists of two neural networks: a generator and a discriminator. The goal of GANs is to produce new data that closely resembles a given training dataset. They are utilized in a number of applications, such as data augmentation, text synthesis, and image production.

The generated data are then label maliciously to formulate the label flipping situation. For this attack, the local model is trained using both flipped and original (local) data and thus deviates from the correct behavior. This simulated environment allows us to explore the challenges and vulnerabilities associated with federated learning in the presence of adversarial clients. It provides valuable insights into the system's resilience and the need for robust defenses against malicious actors within the federated learning framework.

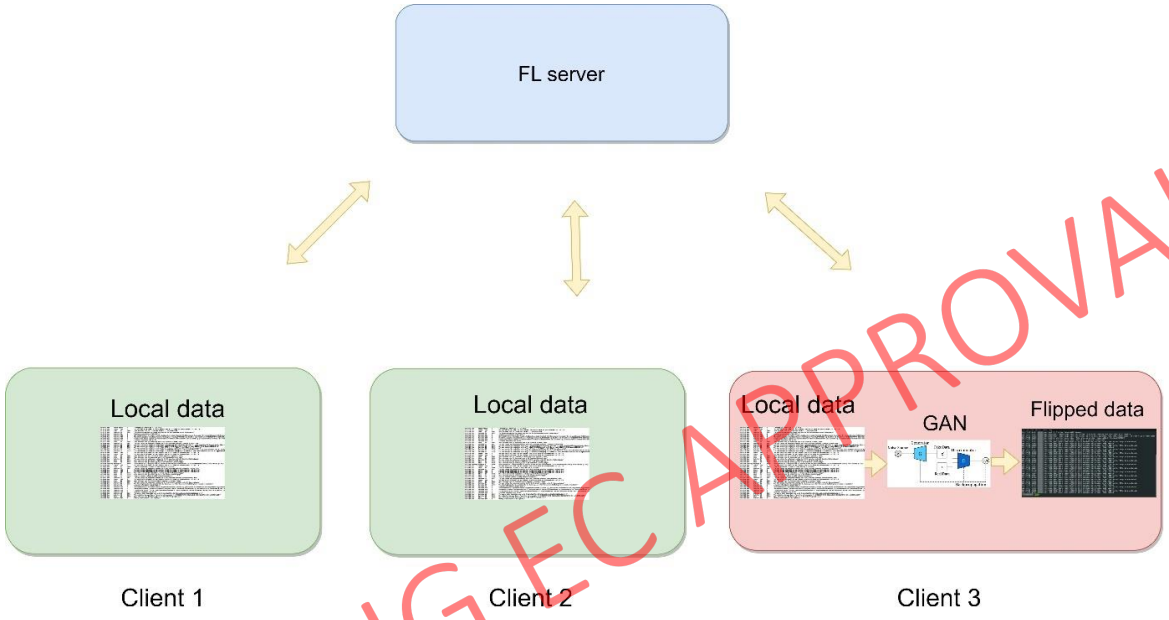


Figure 15: Synthetic attack generation using GAN

Experimentation

To demonstrate the robustness of attacks generated by GANs we experiment with a scenario like Figure 15. The local data we employ is the openly accessible UNSW-NB15 dataset [2] Using this dataset, we demonstrate synthetic generated samples produced by the GAN and subsequently use these to formulate an attack from a malicious client within a federated learning system.

Dataset

This dataset is a widely recognized dataset used for Intrusion Detection System (IDS) research and evaluation. It contains network traffic data and is particularly designed for testing and training IDS models. Researchers and cybersecurity professionals often use this dataset to develop and assess the performance of intrusion detection algorithms and systems. An example of some (standardized) samples of the dataset can be seen on Table 4.

dur	proto	service	state	spkts	...	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	attack_cat
-0.21372	7 0.410563	- 0.674406	0.932695	- 0.124455	...	-0.640033	-0.644190	-0.10607	0.707105
- 0.213728	0.430563	- 0.674336	0.832211	- 0.101091	...	-0.640033	-0.734107	-0.10607	0.707105

Table 4: Scaled examples of the UNSW-NB15 Dataset

Data generated by GAN

In this section, we show how a malicious client can process local data to train a GAN model and consequently produce a model degradation attack. Firstly, the data are standardized, and the label is discarded from the local dataset. Then, a GAN is trained using these data until convergence. After the training process has been completed, the GAN can be arbitrarily used to produce synthetic data. Examples of such data can be viewed on Tabe 5. We can see that the data generated are plausible and resemble the original dataset.

dur	proto	service	state	spkts	...	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	attack_cat
-0.317240	0.461433	-0.681220	0.781153	-0.223455	...	-0.545822	-0.655661	-0.074459	0.559396
-0.307742	0.461189	-0.663046	0.757519	-0.238520	...	-0.548030	-0.628103	-0.071399	0.540178

Table 5: Synthetic IDS data generated by GAN

Comparison of original/generated data

In this section, we compare the generated data with the original to measure the quality of the dataset produced by the GAN. To do this, we use the Kolmogorov-Smirnov (KS) test. This test quantifies the similarity between two datasets using the KS statistic (D). The KS statistic is calculated by finding the maximum absolute difference between their empirical distribution functions (ECDFs):

D = max(sup_x |F1(x)-F2(x)|, sup_x |F2(x)-F1(x)|)

Where:

D is the KS statistic.

sup_x is the supremum (least upper bound).

F1(x) and F2(x) are the ECDFs of the two datasets.

Smaller values of D indicate greater similarity between the datasets, supporting the null hypothesis that they come from the same distribution. Larger D values imply dissimilarity.

Results for this test can be viewed on Table 3.

Table 3: Results for the Kolmogorov-Smirnov test between the two datasets.

KS Statistic 0.04819318181818184

P-Value 0.09

The results of the test indicate a relatively small difference between the two datasets' distributions, with a KS statistic of 0.0482, suggesting greater similarity. The p-value of 0.09 implies that the observed difference is not statistically significant at a typical significance level (e.g., 0.05), thereby failing to reject the null hypothesis. In this context, it suggests that the two datasets are reasonably similar in terms of their distributions, but the interpretation should consider the specific significance level and context of the analysis.

Attack generation

In this section, we show the outcome of the GAN-based attack described above. The malicious client assigns random labels to the generated data and combines the generated dataset with the original local one. In our setup, we have two benign clients and one malicious. The training federated rounds are fifty while each client trains the model locally for two rounds. The results are demonstrated in Figure 16. In this scenario, it's evident that the model's performance is severely compromised as a direct consequence

of the attack, resulting in an overall accuracy that is approximately 20% lower than when no attack is present. This significant decrease underscores the attack's ability to undermine the system's robustness, underscoring the pressing need for effective defense mechanisms to safeguard against such vulnerabilities.

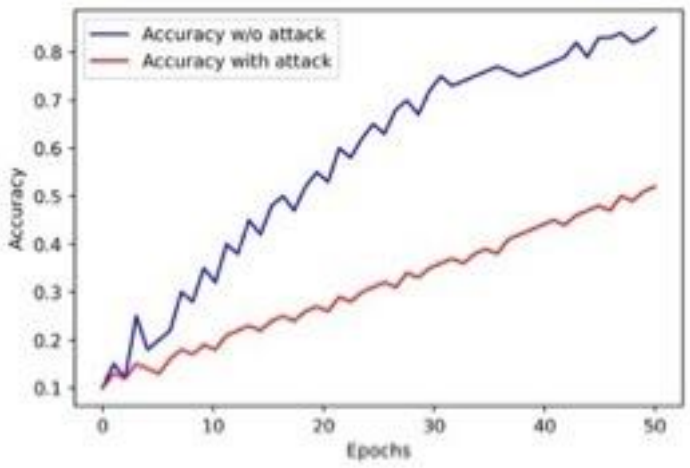


Figure 16: Accuracy of the federated learning model with/without an attack

3.4.4 Attack detection

3.4.4.1 FL attack detection

The scenario described in Section 3.4.2 is constructed as a straightforward federated learning example, aimed at illustrating the vulnerabilities to attacks. This basic setup lacks the capability to defend against malicious clients and is susceptible to attacks that could significantly impact critical aspects, like model accuracy and accurate label classifications. To address these issues, we introduce the FL Attack Detector (FLAD), a server-side component that incorporates a dual-layer security approach, encompassing both proactive and reactive measures. The approach can be viewed in Figure 17.

In this image, we focus in the federated learning server's architecture and study how our security approach can be integrated with this component. On the bottom of the image, we see the local models which are sent from the clients. In a compromised setup these models may be benign or malicious and thus need to be somehow filtered. Our first approach (part one of FLAD) is to filter these updates using a **secure aggregation technique**. A distributed system approach called secure aggregation is used to safely combine data from several sources while maintaining confidentiality and anonymity. Without disclosing specific data points to unauthorized parties, it makes it possible to combine sensitive information.

This component tries to filter out malicious clients by viewing the model updates sent from the clients. The basic assumption is that the model updates generated by malicious clients differ significantly compared to the benign ones. This difference can be investigated in the parameter space, and one can measure the distance between model parameters separately or between whole models (the sum of model parameters). To measure this difference, common distances are the Euclidean distance, cosine distance, etc. Thus, we employee the defences of:

- Median aggregation
- Trimmed mean aggregation (with known number of malicious participants)
- Krum aggregation

The first two defences operate at the parameter level and treat each parameter separately. Thus, they can result in a partially poisoned global model where some parameters are aggregated using only benign updates and other parameters using both benign and malicious updates. On the other hand, Krum treats each model separately, and thus, in a federated round, it is possible for a completely poisoned local

model to be selected as the global one. The choice between each security measure is a configurable parameter and can be adjusted accordingly to the task at hand.

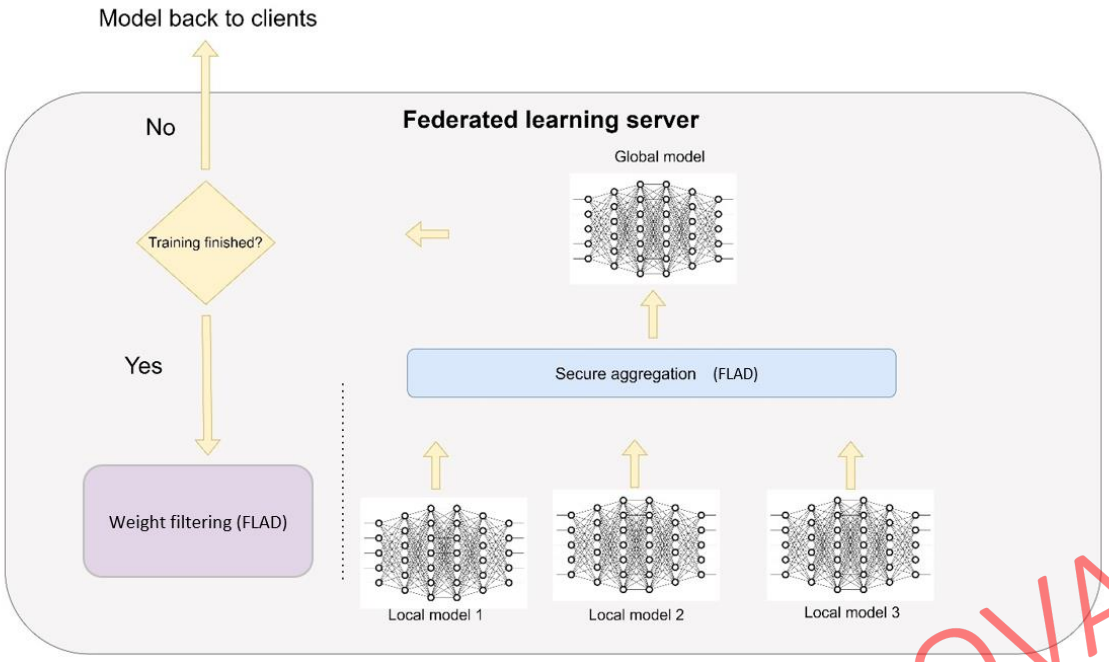


Figure 17: Federated Learning Attack Detector (FLAD

After the secure aggregation, we end up with the global model that has accumulated knowledge from all the clients. If the training procedure is in progress, this global model is sent back to the clients. However, if the previous federated round was the last one, then the second module of FLAD is used, namely the **weight filtering**. Weight filtering includes deleting or setting specified weights to zero depending on predetermined standards or limits. In order to enhance speed, minimize memory requirements, and maybe improve generalization, weight filtering is used to simplify and condense the model.

This method relies on the assumption that the server possesses a limited dataset consisting of pristine, labelled images, and can continue to train the model for a few more rounds after the client training phase. This technique serves to mitigate any malicious activity and rectify the model's performance. To elaborate further on the implementation details, we continue training the global model for an additional $R = 5$ rounds on the server following the conclusion of the federated learning process. This extended training period allows the server to refine the model's performance further, thereby minimising the impact of any potential attacks that may have occurred during the federated learning phase.

Experimentation

In this section, we experiment with our defence mechanism (FLAD) and prove its robustness against the attack mentioned in section 3.4.3. We recreate the attack scenario, but this time, rather than employing a basic aggregation method, we implement Krum. Krum is a secure aggregation mechanism designed to mitigate such attacks effectively. Additionally, after concluding the federated training, we apply weight filtering on the server for 10 local rounds, with the aim of completely neutralizing the attack. Results can be viewed on Figure 18. In this Figure, we see that during the fifty rounds of training the application of a secure aggregation technique helps in mitigating the attack but not completely. Specifically, we see that the accuracy improves significantly but does not reach the levels of the no attack case. This happens because using this mechanism the weights are filtered but not completely since this method is applied on a parameter level. We thus end up with a partially poisoned model. However, we see that the weight filtering applied for 10 local rounds after the federated training is successful in

mitigating the attack completely since we can reach the accuracy obtained by the no attack case in such a short training time.

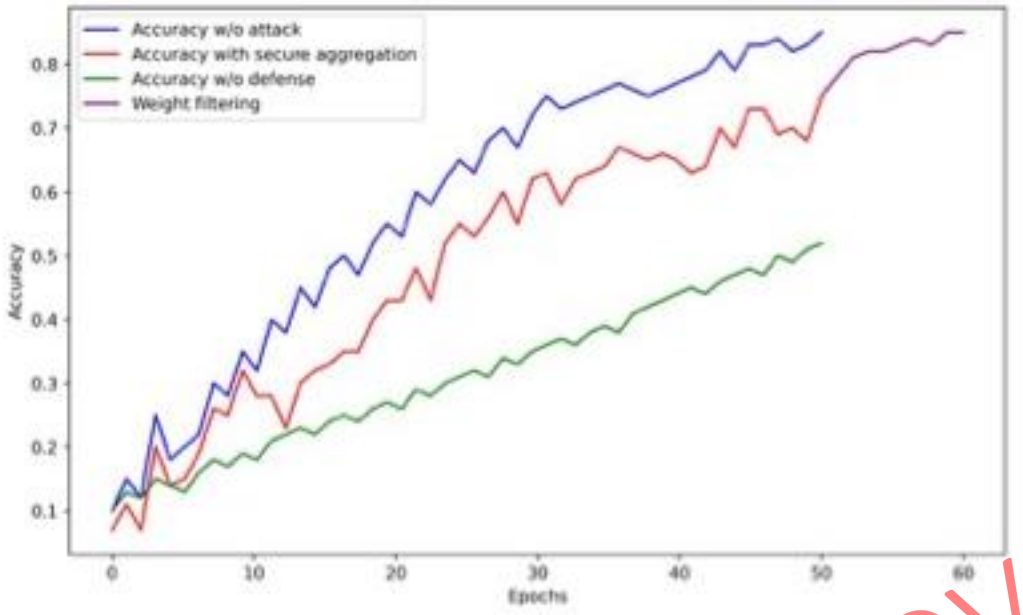


Figure 18: Accuracy results when FLAD is employed

3.4.4.2 Network Attack detection

EVEREST is α monitoring tool developed by STS which will be integrated within NEMO so as to allow for the detection of attacks to the NEMO devices/services. EVEREST is designed to address the evolving challenges of cybersecurity and data protection in contemporary cyber systems. In today's data-driven world, organizations collecting and handling sensitive information must adhere to stringent data protection regulations and security standards. EVEREST steps in to provide the necessary guarantees and measures to ensure data sustainability and protect against potential cyber-attacks by alerting for violations of cybersecurity space, in a continuous runtime manner.

The Key Features of EVEREST are the following:

1. Continuous Monitoring Assessments: EVEREST conducts thorough continuous runtime monitoring assessments to check for violations of security and dependability properties in a system. It is used to ensure that all security solutions in place are functioning correctly and efficiently.
2. Event Stream Analysis: The tool excels in complex event recognition and reasoning, tracking and analyzing streams of events to detect patterns of special significance. Event streams from various sources, including mainly Elasticsearch stack log aggregators, sensors, computer networks, and different log files are processed with exceptional accuracy.
3. Logical Reasoning System: EVEREST is based on Drools, a logical reasoning system that utilizes the Event Calculus formalism. It employs Business Rules Management Language for logical operations related to security policy assessment, ensuring accurate and effective event recognition.
5. Scalable and Distributed Architecture: The tool operates either through Docker Compose or on a Kubernetes clustered-based architecture, offering scalability and seamless distribution of event recognition processes. This design enables it to handle event streams with high velocity and volume, a challenge faced by conventional event processing systems.

The high-level architecture of the monitoring sub-system of NEMO is shown below in Figure 19, in which the interface with the corresponding block developed as part of WP3 (i.e. message broker) is also shown.

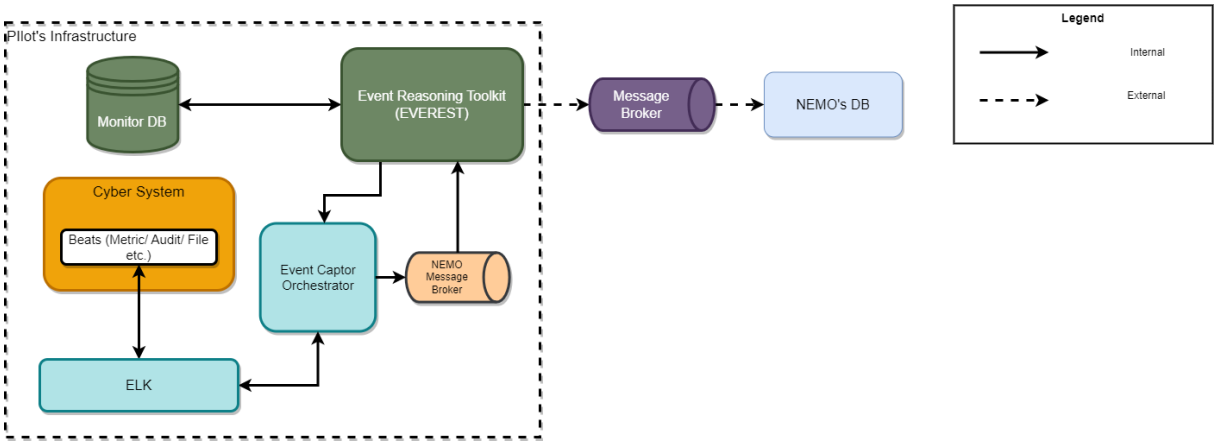


Figure 19: High level architecture of the monitoring sub-system of NEMO

3.4.5 Counter-measure

By effecting a transition in the information exchange flow among entities, shifting from the raw data itself to exclusively the parameters of machine learning models, Federated Learning, by itself, has attained an elevated level of privacy compared to conventional Machine Learning methodologies. FL has demonstrated pronounced utility in applications tailored for mobile platforms[21], capitalizing on the potential for a multitude of participants to partake in training while offering prospects to train on otherwise unattainable data. However, FL in isolation does not suffice to confer comprehensive privacy assurance, as it has been evidenced that models can exhibit memorization tendencies towards the provided training examples, as opposed to merely assimilating feature utilization for task accomplishment. This predicament gives rise to vulnerabilities such as membership inference attacks, which can be employed to discern and recover the original samples used for model training.

A prevalent approach for mitigating this issue, that is supported by the Security modules of NEMO through mainly the implemented message broker, which is described in Section 2.3 of Deliverable 3.1, is the encryption of a model's weights before their transmission to another entity, whether it be a central server or a peer participant. Nonetheless, utilizing the security modules can introduce augmented computational and communication overheads into the workflow. An alternative countermeasure, which is implemented already in NEMO, is the utilization of Differential Privacy (DP), a concept initially introduced by Dwork[69] in 2006 which hinges on the concept of adjacent databases, which are databases differing by at most a single element, and is formally articulated as follows:

Definition 2.4 (Differential Privacy). A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ε, δ)-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all $x, y \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|x - y\|_1 \leq 1$:

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \Pr[\mathcal{M}(y) \in \mathcal{S}] + \delta,$$

Where,

M: Randomized algorithm i.e query(db) + noise or query (db + noise).

S: All potential output of M that could be predicted.

x: Entries in the database. (i.e, N)

y: Entries in parallel database (i.e., N-1)

[21] https://doi.org/10.48550/arXiv.1811.03604

ε: The maximum distance between a query on database (x) and the same query on database (y).

δ: Probability of information accidentally being leaked

DP furnishes the assurance that when an element is either added or removed from the database, the response of our mechanism will not deviate by a specific amount, characterized by exp(ε) + δ. The standard approach to endow a mechanism with Differential Privacy entails the introduction of independent Gaussian noise, the variance of which is proportional to the mechanism's sensitivity. To assess the privacy budget expended by a randomized mechanism, an analysis must be conducted to monitor privacy loss, which encompasses the concept of Rényi Differential Privacy (RDP) [70] can be converted into (ε, δ)-DP. A salient advantage of DP is its resilience to post-processing, implying that the privacy guarantee remains unchanged, irrespective of subsequent manipulations applied to the obtained results.

Conceivably, one might opt to directly apply Differential Privacy to the raw data as a protective measure. However, the introduction of noise invariably incurs a trade-off in accuracy. Therefore, in NEMO our approach is to use the monitoring block to check whether a possible attack is under way and if so then we will apply Differential Privacy. Moreover, in order to be able to efficiently apply DP to the NEMO FL schemes we adopt a certain Differential Privacy version which utilized the Stochastic Gradient Descent (DP-SGD) optimizer, introduced by Abaddi et al.. This algorithm imparts noise not to the data or the model's parameters but directly to the gradient during each step of the gradient descent process. Furthermore, it incorporates clipping to restrict the influence of individual data elements over the training process.

3.5 Conclusion, Roadmap & Outlook

The CF-DRL is a component that develop reinforcement learning algorithm distributed over nodes that share privately their model but not their data. The process is made cyber-secure thanks to detection mechanism of various types of attacks including attacks by one of the nodes itself. Architecture based on a server-client is a standard used here but we are also simultaneously investigating a peer to peer architecture that does not require a central server.

Each partner has demonstrated their capacity to build a function of the final component. The next step is to combine the various function bricks and make them communicate. We are planning as a first step to release stable versions of the reinforcement learning component, the GAN-based attack generator and the ML-based cyber-attack detector (FLAD) on a simulation environment.

The second pillar of development is to support the decision making process of the meta-orchestrator in the management of micro-services across the continuum. This will be realized by training and serving a relevant machine learning model through the CFDRL component. Moreover, in the context of the task activities, further discussions regarding the use of CFDRL directly with the Living labs applications will be conducted.

4 Federated meta Network Cluster Controller (mNCC)

4.1 Overview

The meta–Network Cluster Controller (mNCC) module is the component of the Nemo system that is responsible for the management of network connectivity and the evaluation and export of network characteristics, thus being the link between the NEMO system and the physical topology that underlies this system.

The main functionalities it will provide are to generate a point-to-multipoint connectivity that allows to abstract the distributed network services from the network and how they are deployed. In addition, for efficient management of resources and requests, the mNCC must have the ability to read and display the network characteristics that are available at any given moment, allowing different services to update their view of the network without the need to connect to it.

Another functionality that the mNCC must have been the management of connectivity resource reservation requests. When the meta-Orchestrator identifies that the instantiation of new network resources is necessary, and that therefore a modification of the connectivity is needed in order to provide access to the different nodes that are going to be involved, it must request the mNCC to carry out such management. To do so, this module must have a request management API that allows abstracting low-level implementations to users, separating implementation from network desire.

In addition to request management, the mNCC should be able to expose a view of the network to the digital twin, to allow it to keep abreast of the context and assess the effectiveness of its predictions. On the other hand, these predictions will be sent to the CF-DRL module to perform an analysis of possible scenarios and thus foresee possible consequences. This is why the mNCC should be able to receive these requests, although in the initial design, we are evaluating whether it should pass directly to the mNCC or whether it should first pass through the meta-Orchestrator to perform a more intelligent management of the network.

4.2 State of the Art & Technology Chosen

A federated meta-Network Cluster Controller (mNCC) is a system that coordinates, manages and controls many network clusters that are dispersed across various geographical regions. Depending on how it is implemented, the Federated mNCC may serve as a hub for communication between the various clusters or as a dispersed network of systems that cooperate to manage certain clusters or subsets of the entire cluster.

4.2.1 Link-Layer Secure connectivity for Microservice platforms (L2S-M)

Link-Layer Secure connectivity for Microservice platforms (L2S-M) [71]is a networking solution for micro-services platforms based on Kubernetes (K8s) that complements the CNI (Container Network Interface) plugin approach to create and manage virtual link-layer networks. These virtual networks allow pods (workloads) to have isolated link-layer connectivity with other pods within a K8s cluster, regardless of the K8s node where they are actually deployed. L2S-M enables the creation/deletion of virtual networks on-demand, as well as attaching/detaching pods to those networks. This solution is seamlessly integrated within the K8s environment, through a K8s operator [72].

Figure 20 showcases the design of L2S-M, which achieves its virtual networking model through a set of programmable link-layer switches distributed across the platform. These switches form an overlay

network relying on IP tunnelling mechanisms, specifically, using virtual extensible LANs or VXLANs [73] , and serve as the basis for creating virtual networks.

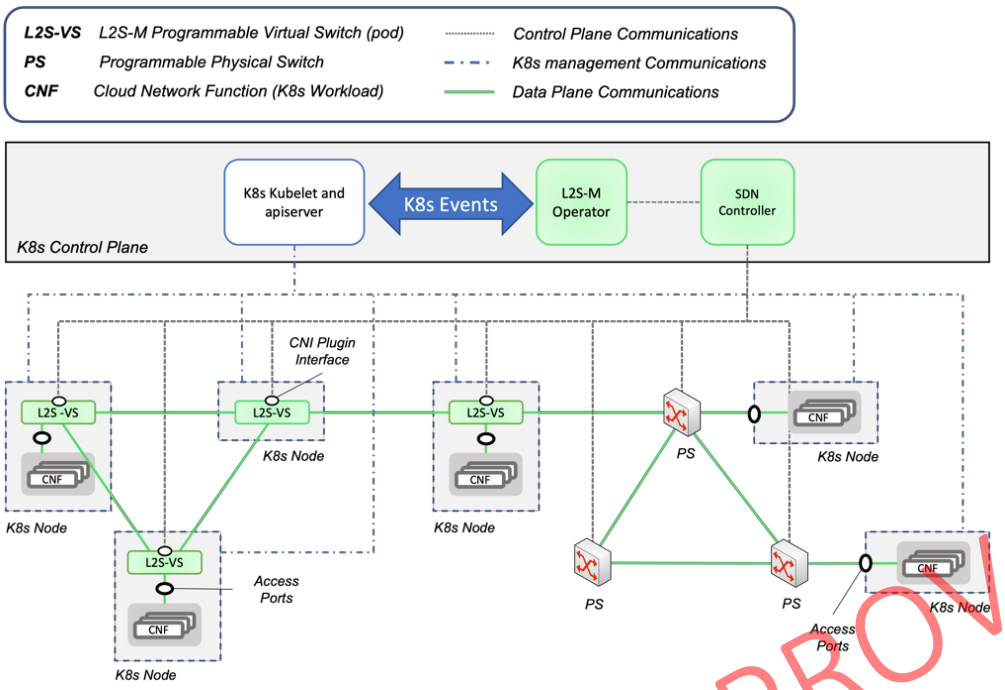


Figure 20: L2S-M Design (figure extracted from the official L2S-M repository [71])

L2S-M provides its intended functionalities using the programmable data-plane based on Software Defined Networking (SDN), which in turn provides a high degree of flexibility to dynamically incorporate new application and/or network configurations into K8s clusters. To support the full programmable aspect of the overlay, L2S-M uses an SDN controller to inject the traffic rules in each one of the switches, and to facilitate the implementation of distributed traffic engineering mechanisms across the programmable data plane. For instance, priority mechanisms could be implemented in certain services that are sensitive to latency constraints.

Moreover, L2S-M has been designed to flexibly accommodate various deployment options, ranging from small K8s clusters to those with a high number of distributed nodes. In this context, the main K8s interface of pods (provided by a CNI plugin) remains intact, retaining the compatibility with all the standard K8s elements (e.g., services, connectivity through the main interface, etc.).

On the other hand, L2S-M extends this approach of establishing virtual link-layer networks through the combined use of network-layer tunnelling and SDN technologies to a scenario where pods may require connectivity with others that run on separate Kubernetes clusters. We refer to this type of connectivity as inter-cluster communications. In this particular situation, it is necessary to introduce two new elements in the design of L2S-M:

- **Network Edge Devices (NEDs)** are programmable switching functions implemented using Open Virtual Switches (OvS). They forward traffic between virtualization domains (Kubernetes clusters). Each domain containing pods that require connectivity with other pods in different domains must have at least one NED. The NEDs are interconnected through protected point-to-point links (e.g., VXLAN/IPsec tunnels), thereby creating a NED overlay network that interconnects all the domains. Thus, NEDs provide a data-plane interface on Kubernetes clusters to support external and inter-domain communications (i.e., between cloud/edge Kubernetes-based infrastructures).
- The **Inter-Domain Connectivity Orchestrator (IDCO)** operates as an SDN controller, implemented as an internal application that runs within an instance of the Open Network

Operating System (ONOS)[74]. All the NEDs connect to the IDCO via the OpenFlow 1.3 protocol, supporting the creation and deletion inter-cluster virtual networks.

When two or more VNFs located in different domains require inter-domain connectivity, each of them is connected to an access port of a local NED (e.g., using L2S-M or the Multus CNI). Then, an inter-domain virtual network connecting those NED ports can be created by communicating it to the IDCO through the HTTP REST API.

Subsequently, the IDCO establishes (tunnel-id-based tunnels) point-to-point or multi-point virtual circuits on top of the NED overlay, through which the inter-domain communications for those VNFs are delivered. This is achieved using the Virtual Network Identifier (VNI) in the case of VXLAN. In our implementation, the virtual circuits created in this manner follow the least-cost path between domains considering the hop count as the cost metric.

4.2.1.1 Teraflow SDN

In light of the TeraFlow project, the ETSI group has decided to develop a secure, open-source, cloud-native SDN controller that aims to integrate current NFV and MEC frameworks and enable integration for traffic management and network devices. The main goals of the project during its current development stage are scalability and control resilience. TeraFlowSDN is also working to offer extensions for OpenConfig network elements and an extension of the ONF T-API for optical SDN controllers [75]

This controller is designed to provide capabilities and services expected for networks B5G. The main areas of focus for this solution are [76] :: (i) defining a cloud-native network operating system (NOS) for high-performance control plane operations, (ii) native support for IP, optical, and microwave traffic transport technologies, (iii) automated zero touch provisioning (ZTP) of network services and operations, and (iv) multi-tenant network slicing as a service and SLA requirements. Teraflow also includes ML modules for security. This project uses Netconf and Restconf as the protocols for communication between APIs, with YANG being the data model used.

Some of the features included in this project are a machine learning-based security module, automatic resource administration component, inter-domain manager, centralized and distributed attack detectors, etc. A more complete view of the features can be seen in the next figure.

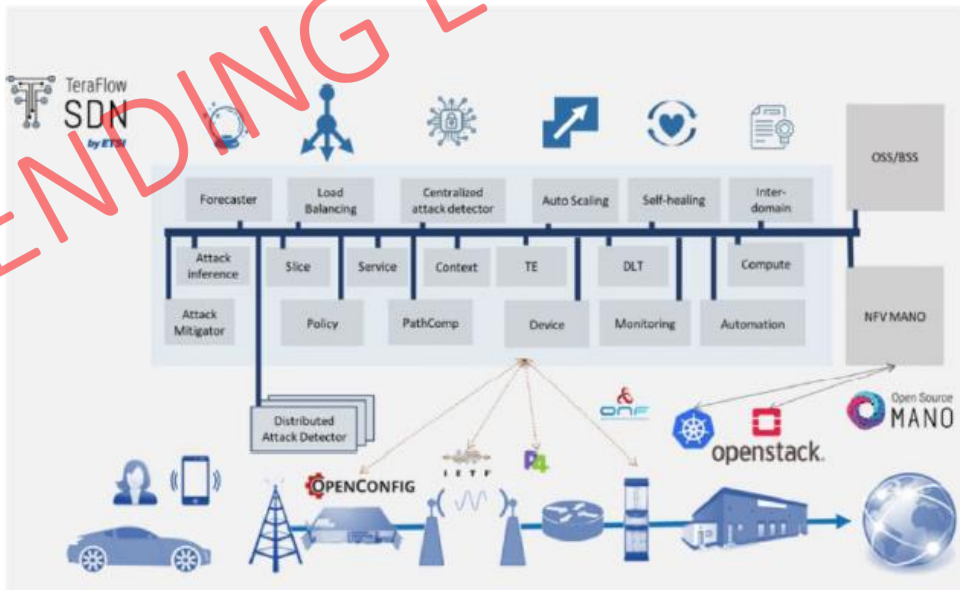


Figure 21: Teraflow SDN architecture. Source: [75]

4.2.2 Network Slice Controller

The proposed Network Slice Controller (NSC) acts as an intermediary sub-module, streamlining the IETF's Network Slice request, realization and management processes. Functioning as a bridge between

higher level systems and Network Controllers, its primary purpose is to receive client requests and determine the resources required to bring the specified network slice to life through its North Bound Interface (NBI). The NSC then interacts with the Network Controllers through its South Bound Interface (SBI) to bring the requested slice to life and oversee its ongoing management. In the domain of Intent-Based Networking (IBN), the NSC harmonizes both the customer's needs and the provider's operational perspectives, ensuring a seamless alignment between requirements and operational actions[77].

This technology also consists of two components: the Network Slice Mapper and the Network Slice Realizer. The Mapper maintains the links between customer requests and the delivered slices, while the Realizer builds filtered topologies based on the network information provided by the Network Controllers. In Figure 22 it is represented that scheme and the data models that are used in the original proposal. However, this module could be used including other data models.

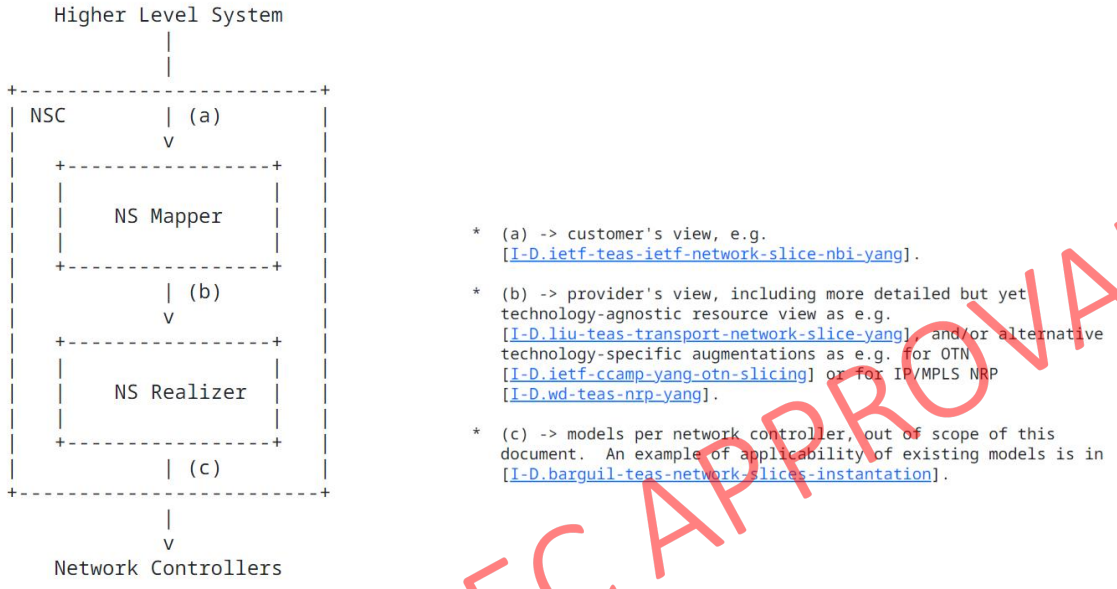


Figure 22: IETF Network Slice Controller structure and associated data models. Source: [[78]]

Ultimately, the NSC can play an important role in creating tailored network capabilities and optimizing resource allocation, being one technology already evaluated in others European projects in order to provide an abstraction layer between the network controller and the client view[77]. Therefore, this module could fit in the integration of the Intent-Based system's interface with other components, focusing on a modular processing of the intents received. The integration of this technology will be internal to the subcomponent indicated, being abstract its presence to the rest of NEMO's modules and, probably, to the rest of mNCC's submodules.

4.2.3 ALTO

Choosing the most suitable network resources can be a difficult task, and sometimes, topology information alone is not sufficient to gain a comprehensive understanding of the network. There may be nodes that provide useful services for some routes, and in other cases, there may be temporary requirements that can only be met by certain nodes. The Application-Layer Traffic Optimization (ALTO)[79]) service provides enriched network information on demand, including additional cost metrics, which can improve network performance by, for example, adapting to network resource consumption patterns. In other words, the main objective is to allow other SDN controllers to have a more accurate and dynamic view of the network, making it possible to predict the best network resources to use (e.g., routes) with higher accuracy and adapt when the context changes. [80]

One of the benefits of using this module in NEMO is the ability to share characteristics that would be particularly desirable in an edge-computing environment, such as nodes with high computational

capacity or URLLC services [81]. Additionally, the orchestrator would have more information about other neighbour networks, helping him to make the best decisions based on the characteristics required and the source and destiny asked for [82]

4.2.4 Apache Kafka

Apache Kafka is a distributed data transmission platform, which allows to publish, store, and receive data streams (referenced as events). This platform works as an asynchronous middleware that allows different technologies to communicate through a soft coupling [83]

This software has three core elements: producers, consumers, and queues. The producers are servers that generate the data and expose them periodically or after triggering events. The clients are the applications and Micro-services that receive this information asynchronously. These clients must subscribe to a topic in order to receive notifications from the queue in question. The third component of Apache kafka are the queues, which are identified by a topic, used as an ID for both publishing and subscribing. Each message that is included in a queue is known as an event, and the system that takes care of managing a set of queues is known as a broker.

One of the main advantages of this solution is the distributed management of the queues, which divides the events between the different brokers, providing advantages in terms of scalability, fault tolerance and availability. This feature makes it an interesting solution for exposing information in federated network contexts, with several network domains, or spread across several geographical regions.

4.2.5 5G networks

The references supplied describe the state of the art in 5G network technology, which includes several important components. To enable a wide range of use cases, including increased mobile broadband, large machine-type communications, and ultra-reliable low-latency communications, 5G must offer a high level of flexibility, scalability, and performance. This is a difficult endeavour since it necessitates the development of new network architecture, spectrum management, and radio access technologies to accommodate the anticipated growth in traffic and the diversity of services and devices. There are additional non-technical difficulties like deployment, standardization, and legal problems. [84]

Another key component of the technology is the architecture of the 5G network[85]. The utilization of a multi-tier network structure in this regard is highlighted by a survey of 5G network design and includes a central cloud-based core network, edge cloud nodes, and radio access network. The efficient distribution of media content and services, as well as the optimization of connectivity and processing resources, are the goals of this architecture.

Support for a range of transport technologies, such as IP, optical, and microwave, automated zero-touch provisioning (ZTP) of network services and operations, multi-tenant network slicing as a service, and SLA requirements are all crucial components of 5G. The sources highlight the usage of a machine learning-based security module, an automatic resource administration component, an inter-domain manager, and both centralized and distributed threat detectors because security is a crucial component of 5G networks.[86] [87]

Beyond fifth-generation networks are also anticipated to be supported by 5G networks. To provide high performance control plane operations and native support for IP, optical, and microwave traffic transport technologies, a cloud-native network operating system (NOS) must be created. [88]

In conclusion, there are several technical concerns with 5G network technology, such as radio access technologies, network design, and spectrum management, as well as standardization, implementation, and regulatory issues. The references also point out a few important components that are being developed to address these issues, such as security, multi-tier network architecture, support for a range of transport technologies, and automated provisioning of network services.

4.3 Architecture and Approach

Logical View

The scheme below represents the different process and interactions mentioned in this sub-section.

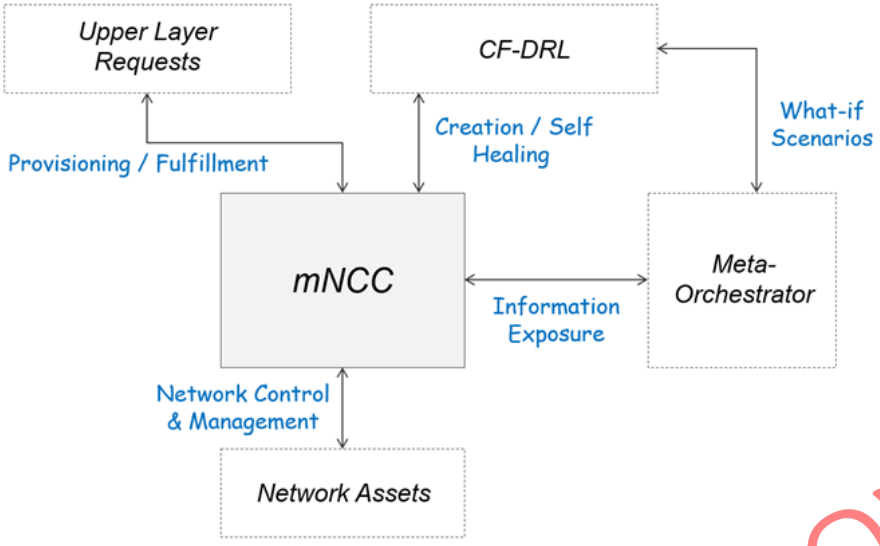


Figure 23: mNCC's Logical View

Development View

Internally, the mNCC must manage these functionalities through the development of several internal sub-modules. These sub-modules are represented in Figure 24.

On the one hand, we have the central sub-module of the mNCC: the connectivity controller. This sub-module will interact with the controllers of the network and compute domains to provide a connectivity overlay for incoming requests. On the other hand, the domain controllers must have access to the network resources, so they must have adapters that allow the connection with each of the different technologies that they must manage. These can also have a resource manager as a mechanism to identify in an agile and intelligent way the different nodes to be configured.

Continuing with connectivity management, the mNCC will also need to have a module that works as an NBI to receive service requests and that provides a level of abstraction to the modules that want to communicate with the mNCC to request connections.

Finally, the mNCC must have a module capable of integrating and exposing the different network metrics needed to establish service connections. These metrics will be obtained and managed internally and must be accessible on demand from outside the mNCC.

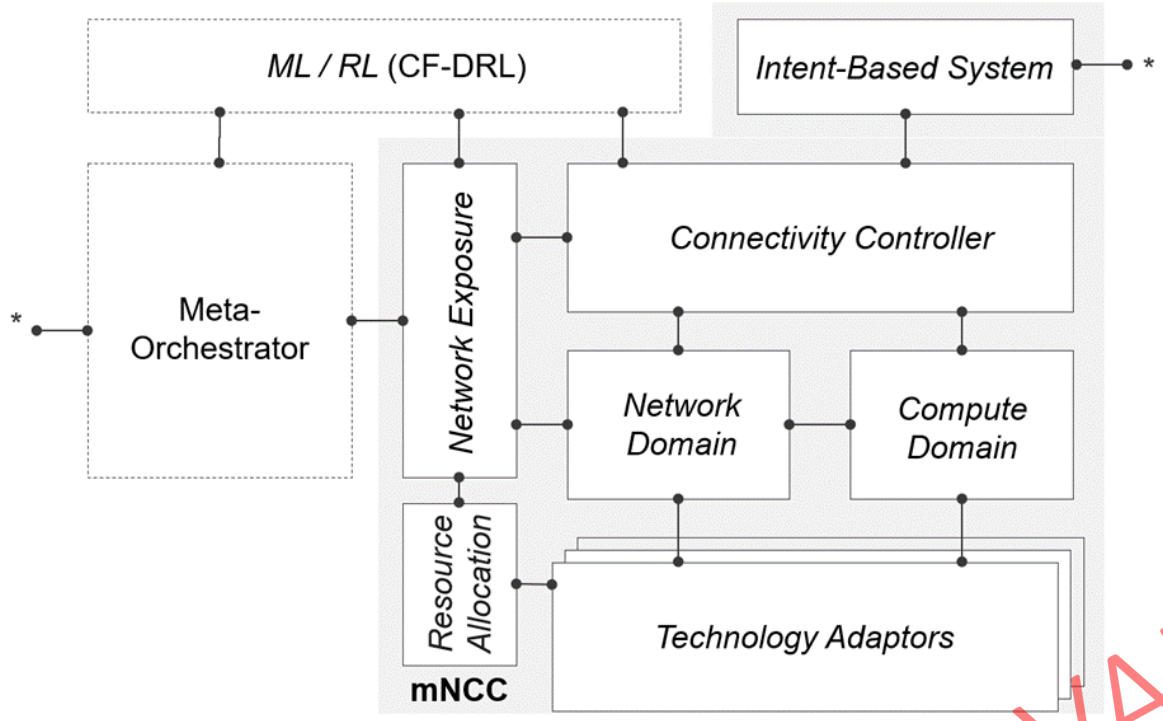


Figure 24: mNCC's Development View

Finally, all this will translate into the management of three main services: connectivity provisioning, capacity exposure and self-healing management. In the process diagram below we can see how the sub-modules discussed above (and described in more depth in the next sub-section) communicate with each other and with other modules of the NEMO system to carry out these functionalities.

It should be noted, as in the previous diagrams, that this is an initial approximation of the design. It is possible to incorporate minor modifications to this diagram if the context so requires.

Process View

The last diagram included is the process diagram (Figure 25). There, there are represented the three main processes to be deployed by mNCC: resource provisioning, information exposure and self-healing. The main goal in this view is to identify how each sub-component communicates which each other, in order to identify how many interfaces do we need, and which information should we exchange in each communication.

Should we indicate that in the first flow, the service request it is not associated to any component as the starting point. That is due to it can be generated by Meta-Orch (mainly), but we are still working with the possibility of letting the CF-DRL to start these communications too.

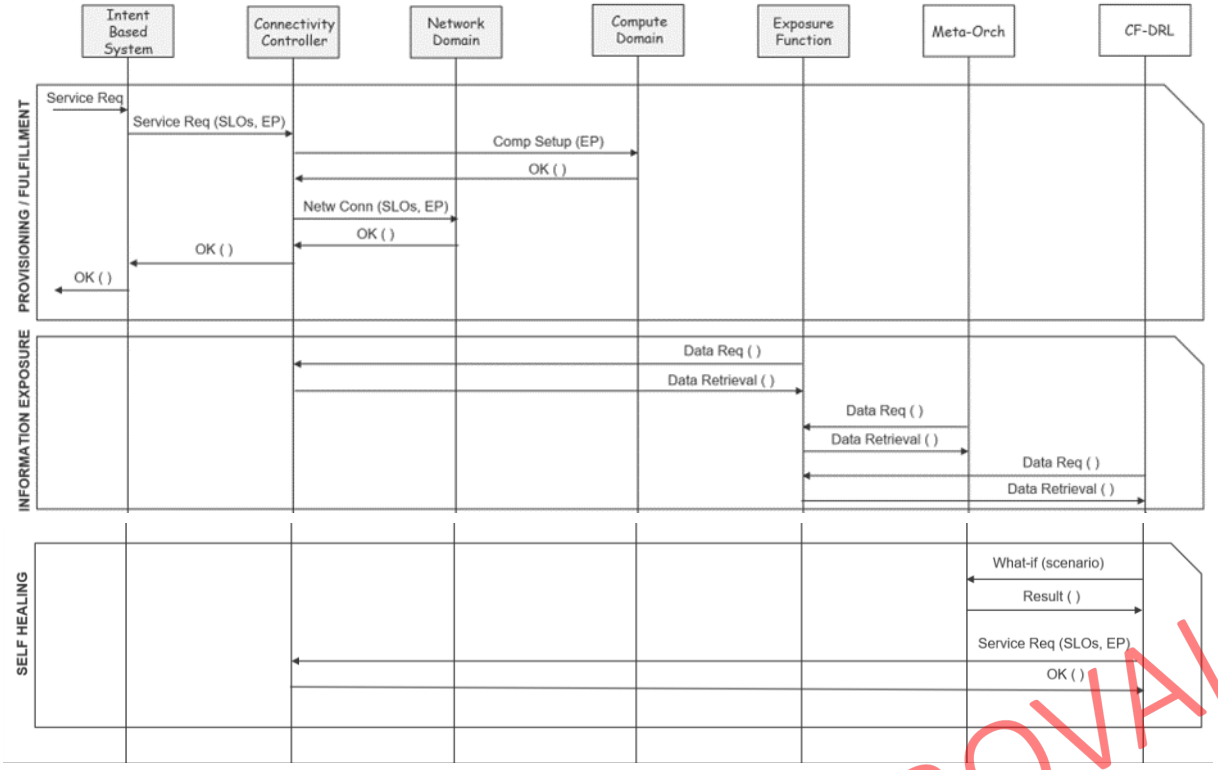


Figure 25: mNCC Process View

4.3.1 Description of subcomponents

4.3.1.1 Connectivity controller

As a component of the mNCC, the Connectivity Controller aims at providing a flexible, scalable and programmable connectivity service to enable the communications in a multi-cluster environment, where virtual functions may be executed over the premises of multiple edge and cloud platform providers. The primary objective of this module is to establish a seamless and dynamic network environment where these virtual functions can communicate efficiently and securely. To achieve this connectivity model, the design of this component is based on the slicing paradigm, and the programmable overlay network relying on IP tunneling technologies approach provided by L2S-M (see section 4.2.1).

The slicing paradigm streamlines network segmentation, creating isolated segments known as slices. These segments enhance adaptability and scalability, supporting dynamic resource allocation for diverse applications and users. This scalability caters to increasing connectivity demands, enabling rapid provisioning of additional slices when required. On the other hand, the L2S-M overlay network approach provides the ideal substrate on which to deliver a programmable data-plane where applications and services can establish point-to-point or multi-point links on demand between each of their constituent virtual functions. To this purpose, L2S-M utilizes SDN, supporting the creation and management of virtual networks to which applications are able to attach and share the same broadcast domain.

Figure 26 showcases the conceptual approach outlined for the Connectivity Controller. As depicted in this figure, this approach encompasses diverse edge and cloud platforms offered by multiple providers. Within this ecosystem, various slices may be established through different Service Level Agreements (SLAs) instituted between these providers. These slices (represented in the figure with different colors such as purple or orange) individually constitute the substrate to deploy a self-contained and isolated overlay network. Building upon this foundation, this approach then aligns with the principles included in L2S-M, with a primary focus on facilitating the management of multiple inter-domain link-layer virtual networks. Thus, each slice supports the creation and management through SDN of several link-layer virtual networks over its corresponding overlay network.

An additional key aspect included in this approach entails the seamless, automated, and on-demand provisioning of each network slice and its corresponding overlay. To realize this, we aim to explore the capabilities offered by the MANO platform provided by ETSI OSM and integrate it as an element of the Connectivity Controller component.

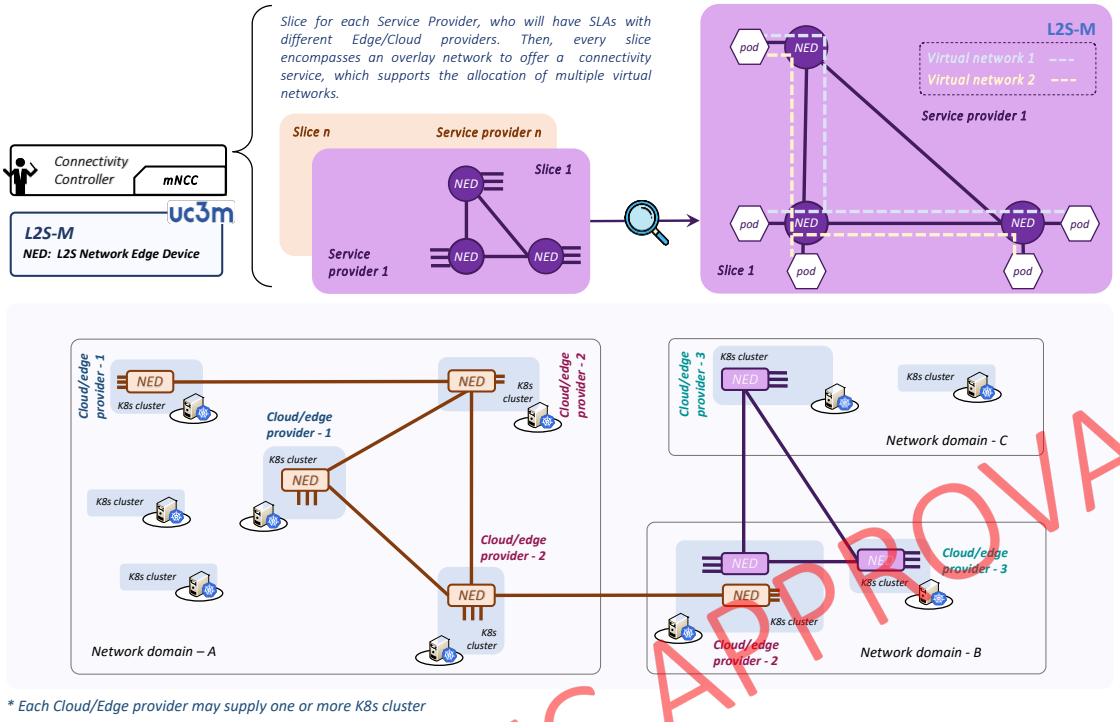


Figure 26: Overview design of Connectivity Controller

Although slices in this approach are characterized by offering isolated environments that facilitate the creation of virtual networks between different Cloud/Edge platforms, this fundamental aspect sets the stage for further exploration. For instance, future lines of research could focus on learning the performance of underlaying networks through the use of the ALTO technology and analyse the applicability of traffic engineering mechanisms to optimize the creation of isolated slices that support different performance and quality of services.

4.3.1.2 Network and Compute Domains

The ability to have complete visibility of our network is essential for effective network and computing resources management. Maintaining up-to-date knowledge of the state of the network is essential to adapt our resource management to the changing conditions and demands of the environment. The network is constantly being updated, and the ability to anticipate and respond to changes in real time becomes crucial. This monitoring of network and compute domains allows us to optimize performance and ensure efficiency in the use of our resources.

The information generated from this network visibility can be integrated and exported to other modules. These modules can leverage this data to make informed decisions about the placement of computing resources and services on the network. The ability to share this information effectively between different systems and applications is a key element in achieving comprehensive and efficient management of our infrastructure, and this ability will depend on effective management of network and data center visibility.

To realize this network and Computer monitoring we are integrating some network protocols as BGP to export network resource capabilities and the use of catalogues of compute infrastructure profiles to be aware of the compute domain status. One example of this Compute capabilities catalogue is the

definition of instances and flavors in which the CNTT (Common Network Function Virtualization Infrastructure Telecom Taskforce) is working in.

The monitoring components to be developed here will connect with the connectivity controller to provide them a vision of the network and with the network exposure module.

4.3.1.3 Network Exposure

In order to be able to make network management decisions, the NEMO system must have visibility of the network. This entails n the one hand, knowledge of the characteristics of the managed network and, on the other hand, knowledge of the computational capabilities of the network devices. To satisfy these knowledge demands, the mNCC module must be able to synthesize and expose this network information.

The information to be exposed needs to be extracted from network devices, being possible the use of network protocol speakers or the integration of recollection tools. Also, the network topology knowledge obtained from the Connectivity Controller will be useful in order to obtain a more complete view of the network status.

To realize the exposure, we are evaluating the use of a network exposure protocol as ALTO is. With this protocol, we could integrate the data extracted from the network and simplify it to be easier to digest for the meta-orchestrator and other modules that could require network status information. Also, we are evaluating two potential APIs for this data exportation: Using a HTTP API-Rest solution (easy to integrate and to deploy) or an Apache Kafka middleware (faster and more efficient). These decisions should be taken during the next steps of the implementation.

4.3.1.4 Intent-Based system

In addition to exporting the information, the mNCC module shall also be able to receive requests for connectivity and environment modifications via the meta-Orchestrator. These requests should be as standardized as possible, in order to avoid multiplying complexity.

That is why we will use an interface based on intents-based networking (IBN) for such communications. The IBN paradigm allows a client and a provider of a network request to communicate despite having two views of the network context that are far apart. The client requesting the service has a high-level view, where it asks for a what, without worrying about the how. In turn, the service provider is able to identify the "what" requested and turn it into a "how" transparently to the customer, without the customer knowing what is going on behind the scenes.

This sub-module will therefore serve as a connection between the Meta Orchestrator, which will request the mNCC (and more specifically the Connectivity Controller sub-module) an overlay connection between some given points and with some basic requested characteristics. After this, the Intents submodule will receive it, and from a predefined scheme will transform this request into a lower-level request to the Connectivity Controller, isolating both sides of the communication and integrating these requests in a simpler way.

As a starting point for this sub-module, our first option is to use the IETF Network-Slice Controller (NSC) component, adapting its features to the context in which we find ourselves: On the one hand, its mapper functionality will have to be evaluated to see if it is needed or if the Meta-Orchestrator could already function as such. Then, the Realizer part, instead of transforming the request into an IETF request, would convert it into one suitable for the connectivity controller module.

4.3.1.5 Resource allocation

Resource allocation plays a critical role in optimizing the performance and efficiency of the Federated Meta Network Cluster Controller. In our architecture, resource allocation refers to the intelligent distribution of computational resources, such as computing power, storage, and network bandwidth, among the federated nodes. To ensure effective resource allocation, we employ dynamic algorithms and techniques that take into account the varying demands of the federated learning tasks and the available resources across the network.

In an effort to enhance resource allocation's precision and effectiveness, the mNCC is integrating a network performance probe. This probe serves as a component for monitoring the real-time performance of network segments and devices within the federated network. By continuously assessing network conditions and performance metrics, the network performance probe provides insights into network health and utilization.

The probe gathers data on various network parameters, including latency, bandwidth utilization, packet loss, and congestion levels. This data is measured using sockets communications between machines. The real-time information is then made available to the mNCC for informed decision-making regarding resource allocation.

4.3.2 Interaction with other NEMO components

4.3.2.1 Cybersecure Federated Deep Reinforcement Learning (CF-DRL)

By integrating this comprehensive network data into the CF-DRL, this module gains the ability to thoroughly analyze network scenarios and predict potential outcomes. Such insights allow the CF-DRL module to not only meet the stringent Key Performance Indicators (KPIs) of constructing network clusters within 5 seconds and executing cluster self-healing or re-configuration in under 20 milliseconds but to also develop highly informed, intelligent strategies for network management.

The real-time actions derived from the inherent nature of reinforcement learning within the CF-DRL module ensures that the mNCC is always responsive to the dynamic, ever-evolving nature of the network.

4.3.2.2 Meta-Orchestrator

The meta-Orchestrator in the NEMO project is an AI-based module that dynamically configures the mOS setup across various nodes (IoT, Edge, Cloud, ad-hoc, or hybrid Clouds) in real-time. Its primary objective is to optimize the end-to-end federation, aligning with application Service Level Objectives (SLOs) and administrator-defined policies. To do so, it will need to communicate periodically with the mNCC module in order to obtain network metrics and ask for modifications in the network management.

As it is shown in Figure 27, mNCC module and the meta-Orchestrator will collaborate to create optimal, self-healing hosting clusters, across different administrative boundaries.

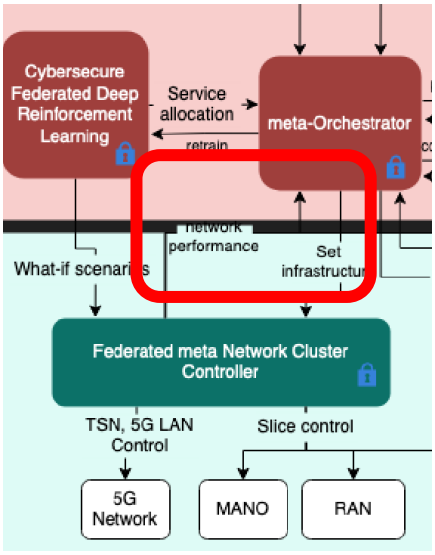


Figure 27: Interface between mNCC and meta-Orchestrator module

We propose implementing an Intent-Based communication mechanism between these two modules to abstract the implementation details from the user view. This abstraction layer will simplify integration and enable easy adaptation of both modules, connecting the meta-Orchestrator module with the Intent-Based sub-module proposed before. The interface will be designed with two connections within meta-Orchestrator:

1. An output interface from the network exposure sub-module to the meta-Orchestrator.
2. An input interface from the meta-Orchestrator to the Intent-Based System, conveying instructions for infrastructure modifications that are needed to be implemented.

Although both interfaces facilitate one-way information sharing, we advocate for making them bidirectional to enhance communication and enable state-feedback. We are also evaluating the potential inclusion of an asynchronous middleware to implement the first interface, making it one-directional.

In order to be able to encrypt the intercommunication of the federated learning framework the Message Broker and the Identity and Access Management sub-system of NEMO (as designed in WP3) will be utilized.

The Interface to and from the message broker consists of

- The meta-data including the tokens from the authentication mechanism and/or the encryption/decryption.
  The data to be encrypted in the transmitting node and the decrypted data in the reception node.

4.4 Conclusion, Roadmap & Outlook

Meta Network Cluster Controller (mNCC) is the Nemo component in charge to provide connectivity to the rest of the system, creating an abstracted view of the network managed and the micro-slices that will work underlay. To be able to provide these functionalities we will need to count with some components to be aware of the Network status. To do so, we are implementing three components: Network Domain, Compute Domain and Resource Allocation module. These components could be simplified and have a common integration. These modules will provide information to the Network Exposure function, that will export the information to the other Nemo's components and also to the Connectivity Controller (CC), module in charge to stablish the required connections, creating an overlay that abstracts the network below. Lastly, we need to receive these connectivity requests, being the Intent-Based System

the component in charge of receive the abstracted requests and adapt them to be interpretable internally for the CC component.

This module will be communicated with the meta-Orchestrator and also with the Cybersecure Federated Deep Reinforcement Learning module. The first one will request network status information and the creation of connections. Meanwhile, the second module will request for potential modifications in the connections already created.

The expected roadmap will be divided in 4 stages:

- First stage, yet completed, is the design of the module and the definition of its capabilities and scope. This includes the internal definition of components, which one needs to communicate with each other, the members responsible of each design and deployment and what we are doing in each of the next stages.
- The second stage, and the actual one, is an early deployment of the main functionalities of each module. During this stage, we also need to have periodical communications between the different work groups, and with the different NEMO's components in order to agree the interfaces that will be needed (number, types and data models). We include these communications in this stage as we have a clearer view of the components than in the stage before and also it is an agreement needed before the next stage.
- Third stage, integration. Once we have the functionality developed, the next task is to integrate all the isolated components in order to provide the big-picture functionalities. This will potentially require modifications on each component to adapt to the specifications and the whole functionality. This integration will count with two sub-stages: the first one is the internal integration, where we create the mNCC component itself, and the second one is the external integration, where we integrate this module with the rest of NEMO's modules.
- The fourth stage is the adaptation into a multi-cluster architecture. Once we have the functional requirements deployed, we need to make the module adaptable, scalable and resilience; and that includes multiclustering and the capability to work on federated environments.
- The fifth, and last, stage is the validation one, where we probe the module integrated in another context, in order to evaluate if the solution archived is good enough.

All these stages require continuous evaluations, being a flexible roadmap and not a hard one. The main idea is to have a clear idea about how to proceed, but in case unexpected problems appear, we need to be able to have enough flexibility to be able to solve them.

To complete this roadmap before the stablished deadlines, we expect to have the stable code of each subcomponent before the end of 2024 Q1. Therefore, we would count with almost 5 months to work in an early integration of the components, being able to submit the results in the first D2.2 review. In this stage we should have a firs stable version and being working on the correct provisioning of capabilities in D2D/D2E/E2E contexts. The first four stages should be completed before the end of 2025 Q1 (more specifically, before the beginning of March 2025) in order to have enough time to validate the results in the last stage (expected to be at least 6 months).

5 Time Sensitive Networking

5.1 Creation of slice and requesting data flow with needed QoS level

Network Slices can be created in the 5G core by using RESTful API assuming 5G RAN is supporting the characteristics of requested slice or slices. When creating slices we can define the maximum uplink and downlink capacity of the slice, type of the slice (EMBB, URLLC and MIOT), available QoS profiles and max number of users of the slice.

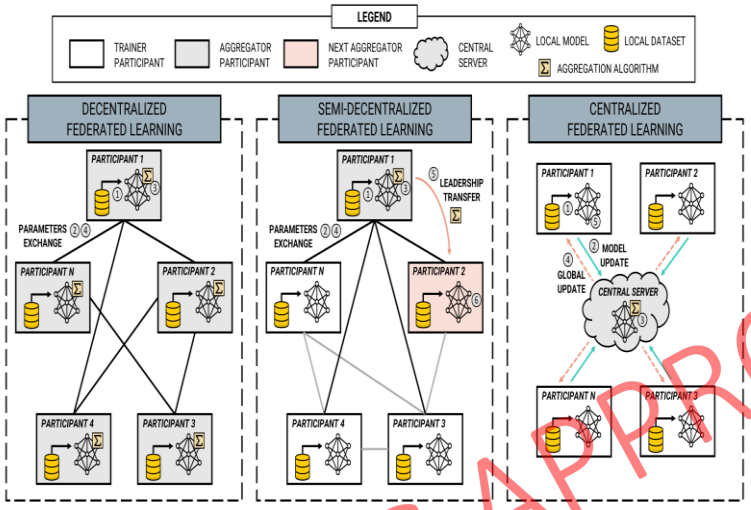


Fig. 4: Common approaches based on DFL architecture decentralization.

Figure 28: Network Slice management architecture

Data flows can be requested dynamically from outside of 5G SA network using RESTful API or through NEF. Dataflow request includes source and destination IP address, traffic flow template and QOS information.

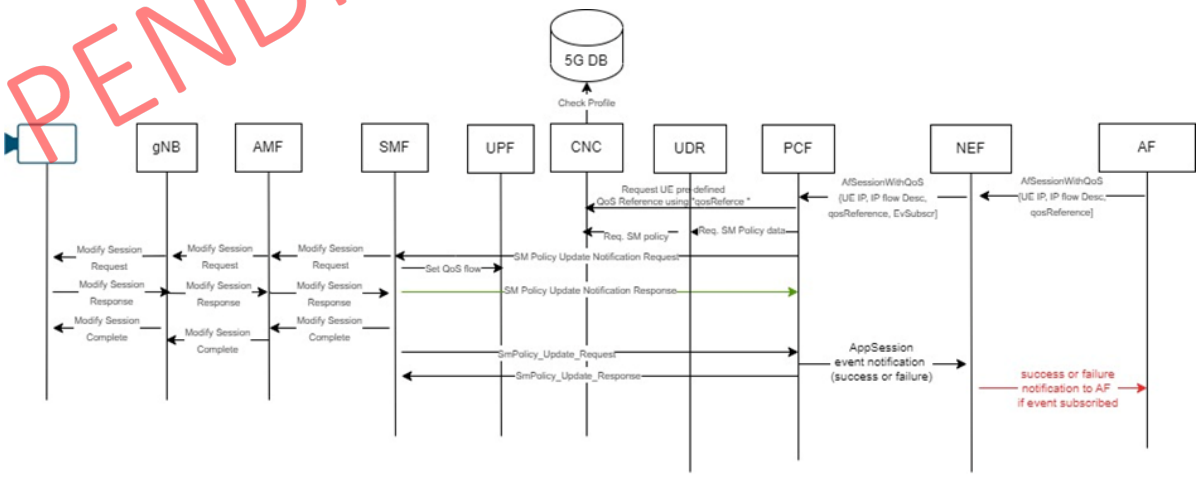


Figure 29: Data flow creation signalling flow

5.2 Overview

TSN features enable synchronizing mobile network UEs using mobile network. TSN can also be used to synchronize mobile network with existing IT infrastructure like automation systems. TSN traffic is organized in to own Network Slice to manage user group and to give TSN traffic high priority.
The industrial LAN may consist of TSN-enabled Ethernet bridges. The latest release of 5G specification supports the fully centralized TSN configuration model, where a central controller should be able to configure both Ethernet and 5GS bridges as a unified network. The 5GS supports the whole industrial network, both Medium Access Control (MAC) learning and flooding based forwarding as well as the static forwarding configured by the central controller need to be supported. 3GPP has defined that a 5GS can be modelled as one or more virtual TSN bridges.

5.3 State of the Art

3GPP Release 16 supports absolute time reference delivery as a service to UEs over the 5G System. The CNC is the entity in the TSN network that has complete knowledge of the network topology and is responsible for configuring the bridges to enable transmission of TSN streams from source to destination. The 5G control plane is interacting with CNC via the TSN Application Function (AF) which maps between the TSN parameters and the 5GS parameters. The TSN AF reports the 5GS bridge capabilities such as minimum and maximum delays between every port pair and per traffic class, including the residence time within the UE and DS-TT via TSN-AF to CNC.
Topology discovery information based on the widely adopted standard IEEE 802.1AB Link Layer Discovery Protocol (LLDP) is also exposed. The TSN AF also exposes its TSN capabilities like the support for scheduled traffic and *per-stream filtering and policing* (PSFP) as specified in IEEE 802.1Q-2018 in case that they are supported by all of the ports. The CNC obtains the 5GS bridge VLAN configuration from TSN AF according to IEEE Std 802.1Q
The TSN AF shall be pre-configured (e.g. via OAM) with a mapping table. The mapping table contains TSN traffic classes, pre-configured bridge delays (i.e. the preconfigured delay between UE and UPF/NW-TT) and priority levels. The CNC reads the capabilities of all bridges and calculates the traffic paths and schedules in the network. The CNC then provides the bridge configuration to the 5GS through the TSN AF, which contains, e.g., scheduled traffic, PSFP, and traffic forwarding information. In order to support QoS for Ethernet and TSN traffic, the traffic flows are mapped to 5G QoS flows. The CNC configures the traffic handling in the 5GS bridge for the different traffic classes according to the capabilities that have previously been reported by the 5GS bridge. The 5GS maps the Ethernet/TSC traffic classes or TSN traffic streams to the corresponding 5G QoS flows.

5.4 Technology Chosen

3GPP has defined 5G VN groups consisting of a set of UEs using private communication for 5G-LAN type services. A 5G VN group can be utilized for IP or Ethernet based services. A specific data network, identified by a *data network name* (DNN), is one of the possibilities to realize a 5G VN group, where the VN group can be either provided by Operation and Management (O&M) or by an TSN-AF. 5G VN group where the SMF has full control of the Ethernet network topology among the 5G VN group members (by control of forwarding decisions on all Ethernet PDU sessions from different UEs).

5.5 Architecture & Approach

For a centrally managed Ethernet network, it is required that the NMS/CNC can configure the VLAN handling for all bridges and all ports, including the 5GS bridge as shown in the Figure 30, as specified in IEEE 802.1Q.

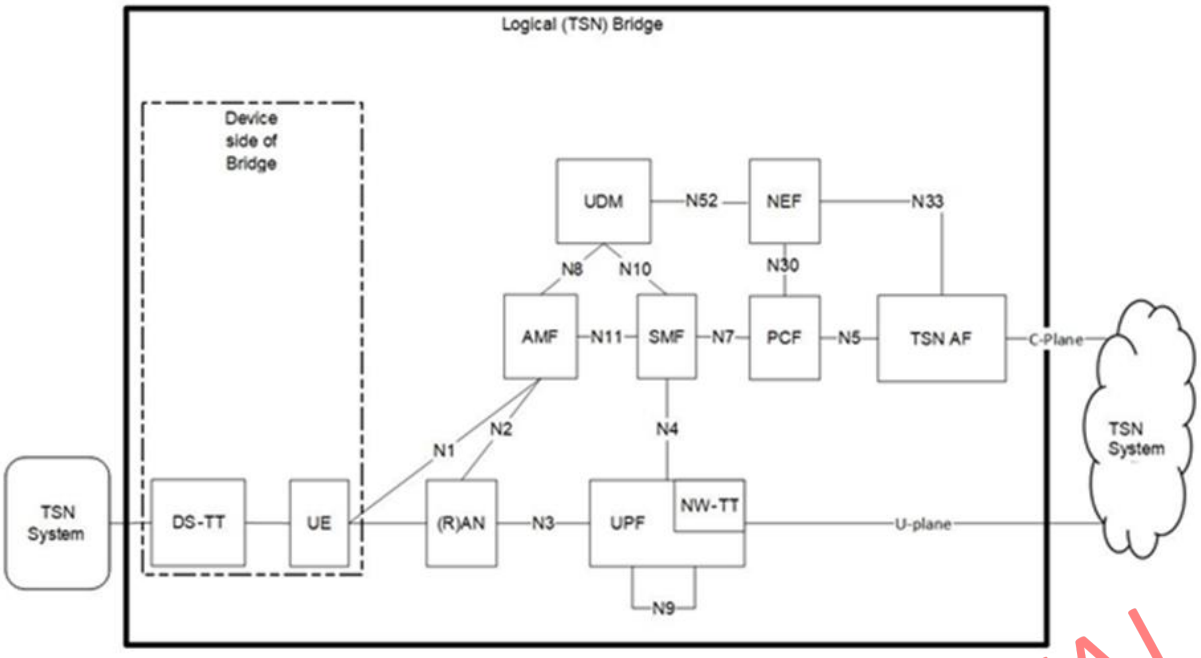


Figure 30: TSN architecture

5.5.1 NW-TT

NW TT (Network TSN Translator): supports link layer connectivity discovery and reporting as defined in IEEE Std 802.1AB for discovery of Ethernet devices attached to NW-TT. When integrating normal devices, we cannot assume that would be a DS-TT does not support link layer connectivity discovery and reporting, then NW-TT performs link layer connectivity discovery and reporting as defined in IEEE Std 802.1AB for discovery of Ethernet devices attached to DS-TT on behalf of DS-TT. If NW-TT performs link layer connectivity discovery and reporting on behalf of DS-TT, it is assumed that LLDP frames are transmitted between NW-TT and UE on the QoS Flow with the default QoS rule. Alternatively, SMF can establish a dedicated QoS Flow matching on the Ethertype defined for LLDP (IEEE Std 802.1AB).

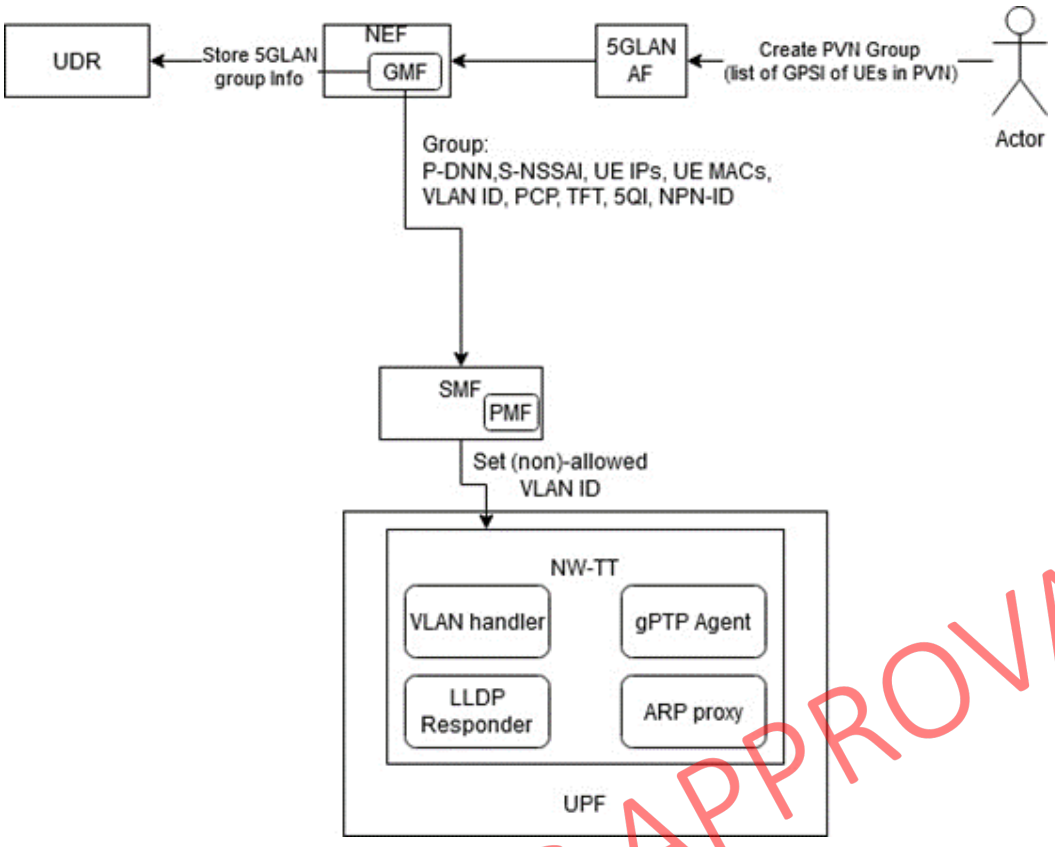


Figure 31: NW-TT diagram

5.5.2 DS-TT

UE is defined to have attached functions of time stamping in data frames, using the device side of the TSN translator (DS-TT). On this Network Function, a step of time synchronization is implemented using the TSi from the Suffix field of the gPTP messages (Sync or Follow Up messages), as it has been defined on 24.535 from 3GPP.

In order to achieve this function a 5G-Modem is integrated in a NPN following an inherent structure based on a 5G component, a TSN packet handler and wired network components. These last two components may be integrated in the same hardware such as a microprocessor but following the TSN standards defined on IEEE 802.1 Qbv, 802.1 CB, 802.1 As and 802.1 Qbu.

Following the structure of the Figure 5, it is possible to get a UE that may perform as a bridge between a 5G NPN and an industrial wired network in addition to implementing TSN.

As it points out, TSN main characteristics are performed: Seamless Redundancy that permits ultra-reliability due to the duplicity of buffers, Frame Preemption that includes a priority of the information that produces, and time critical communication. In addition to the Time Aware Shaper that allows scheduling of the outputs and Time Synchronization.

5.5.3 Interaction with other NEMO components

DS-TT is integrated with a CPE used in the Smart manufacturing use case to synchronize AGVs and robots.

5.6 Conclusion, Roadmap & Outlook

TSN is a crucial functionality when using 3GPP technology in factory automation. Mobile devices need to be synchronized in the environment that are not capable using satellite synchronization. 3GPP has continue further defining and improving synchronization accuracy in upcoming releases.

For example, in the release-17 work aims to define further enhancement for integration with IEEE TSN including support for uplink synchronization via 5GS, support for multiple working clock domains connected to the UE, and support for Time Synchronization of UE(s) with the TSN GM attached to the UE side via 5G System.

6 Proof of Concept: NEMO

The chapter also addresses the objectives, expected outcomes, and limitations of the Integration PoC, providing valuable insights into the complexities of system integration and its role in ensuring a robust and reliable operational framework. The main scope of the meta-OS is to manage the different workloads as defined in D3.1 and here the PoC of the main components in WP2 will have the scope to an initial demonstration of how the different component can be fine-tuned towards this objective.

6.1 CMDT (Cybersecure Microservices' Digital Twin)

The primary objective of this PoC is to provide an initial report on the development of the CMDT and the main functionalities mapped in it. All the functionalities described in Chapter 2 will be delivered as the project advances. In this deliverable, the first initial version will be showcased. With respect to the CMDT fully equipped of its main functions, we will demonstrate how the workload/microservice asset managed by this component can be registered in storage and how certain functionalities related to the given workload can be displayed in user interfaces made of the react client. This part of the workflow is consistent with what is in deep described in both deliverables D3.1 and D2.1. This flow in fact regard both the CMDT and the Meta-Orchestrator that play pivotal roles in managing workloads that are processed within the NEMO meta-OS.

Here the list of the functionalities enabled for this PoC

- React client
- Back-end with enabled functionalities of update/modify the workloads as descriptors

The CMDT can receive via API or for sake of clarity manually uploaded a document showing the information needed as descriptor

6.1.1 Model

We're presenting a basic example to show how CMDT, works. In this test, we'll show that CMDT can both receive and send data. Specifically, we'll focus on how it handles JSON messages that an external actor sends to us (it can be a NEMO component). We'll demonstrate how this message can be uploaded to CMDT and then downloaded again explaining the flow of information through it.

We will therefore identify that the functions made available are essentially only these 2 interfaces (APIs) which we report below as the endpoints, which we will subsequently specify in detail as the project goes on.

6.1.2 Interfaces

- **Upload**

endpoint: http://demo2552160.mockable.io/Nemo_endpoint1

Method:Post

- **Downloads**

Endpoint: http://demo2552160.mockable.io/Nemo_endpoint2

Method: Get

In the screen there is the possibility to add a descriptor or to be directly connected to the interface (Figure 32)

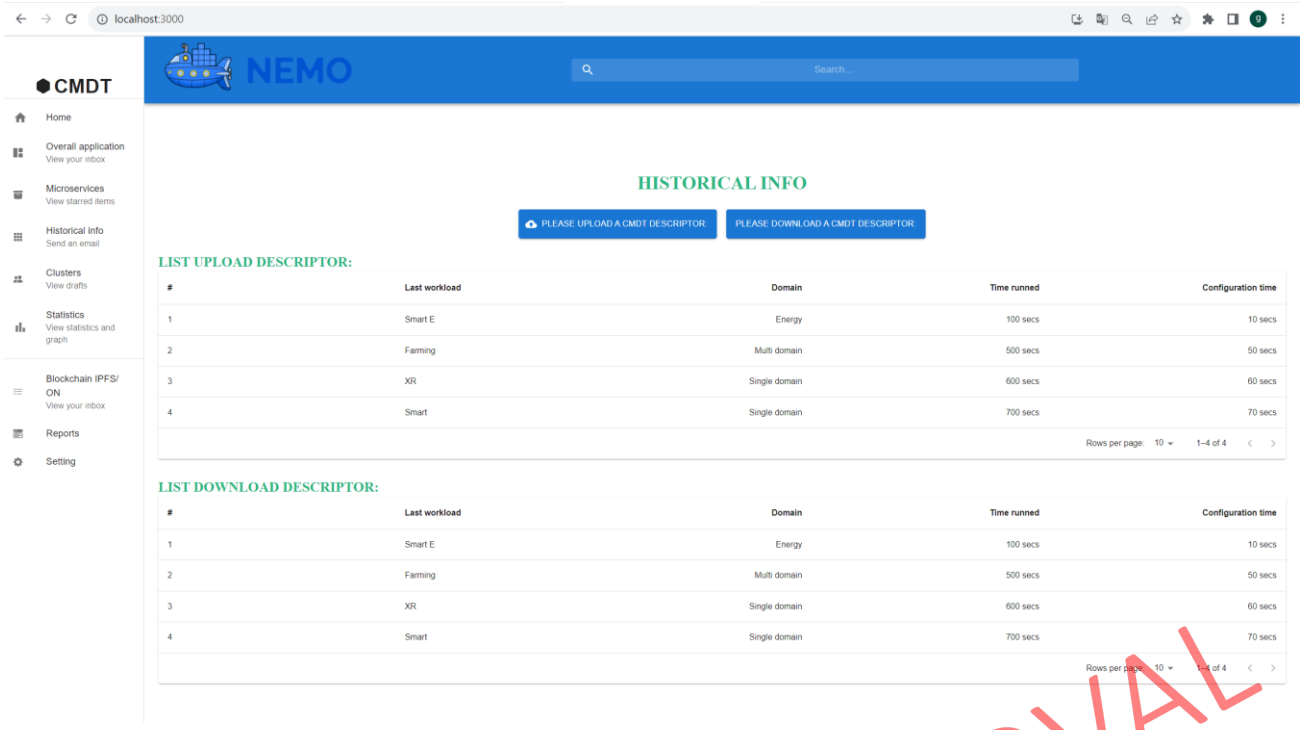


Figure 32: CMDT web page

In particular, in order to download a certain "descriptor", by clicking the "PLEASE DOWNLOAD A CMDT DESCRIPTOR" button, the following window will open where it is possible to type the name of the desired file (e.g. YAML or JSON) and save it for example in local computer (Figure 33)..

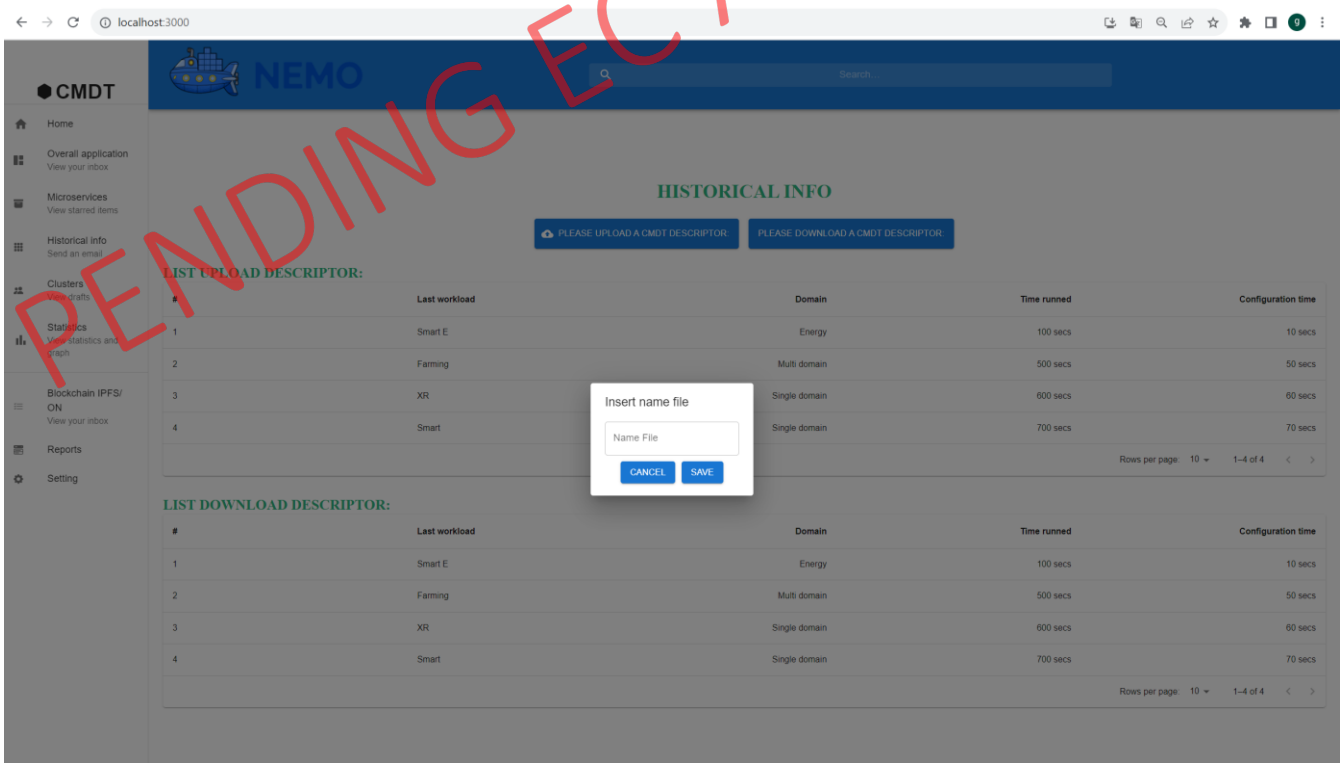


Figure 33: Download "Descriptor"

In this CMDT "proof of concept", however, to set up part of the backend you could decide to store this data for example on a Mongo DB (Figure 34):

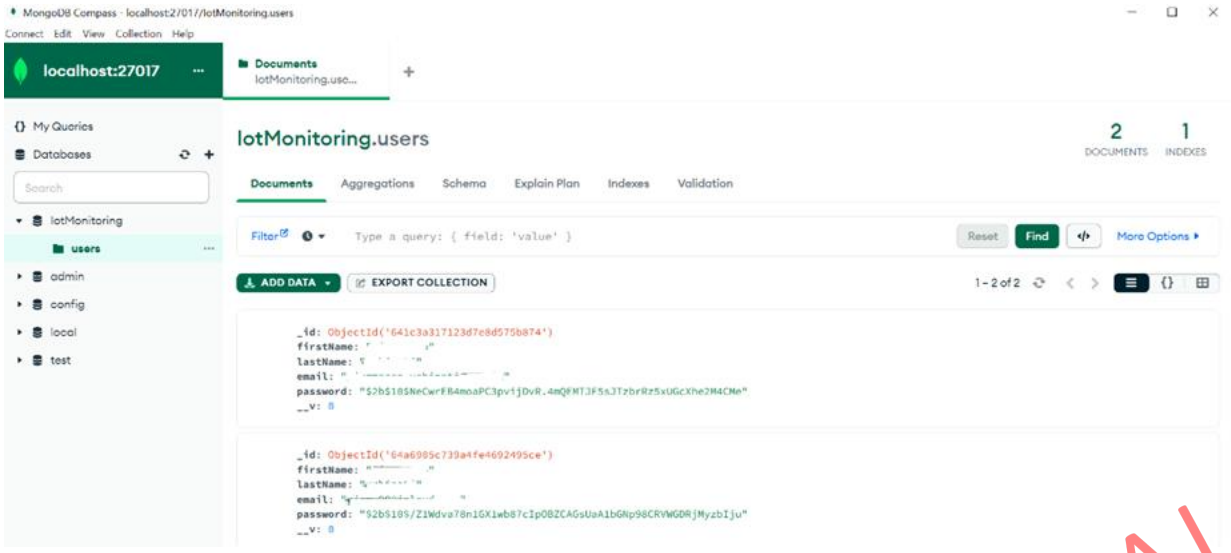


Figure 34: Mongo DB Interface

As an example, code snippet for the backend, see the Mongo DB connection code above (See "Code Snippet Example")

Furthermore, in the drawer blocked on the left, you can click on the "Statistics" item

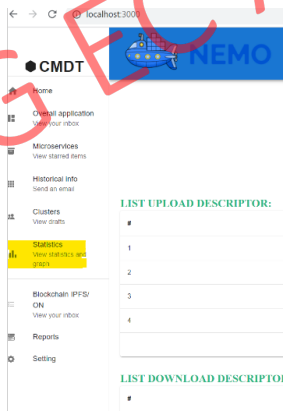


Figure 35: "Statistics"

and another page opens where we will display "Statistics and graphs" (Figure 35)

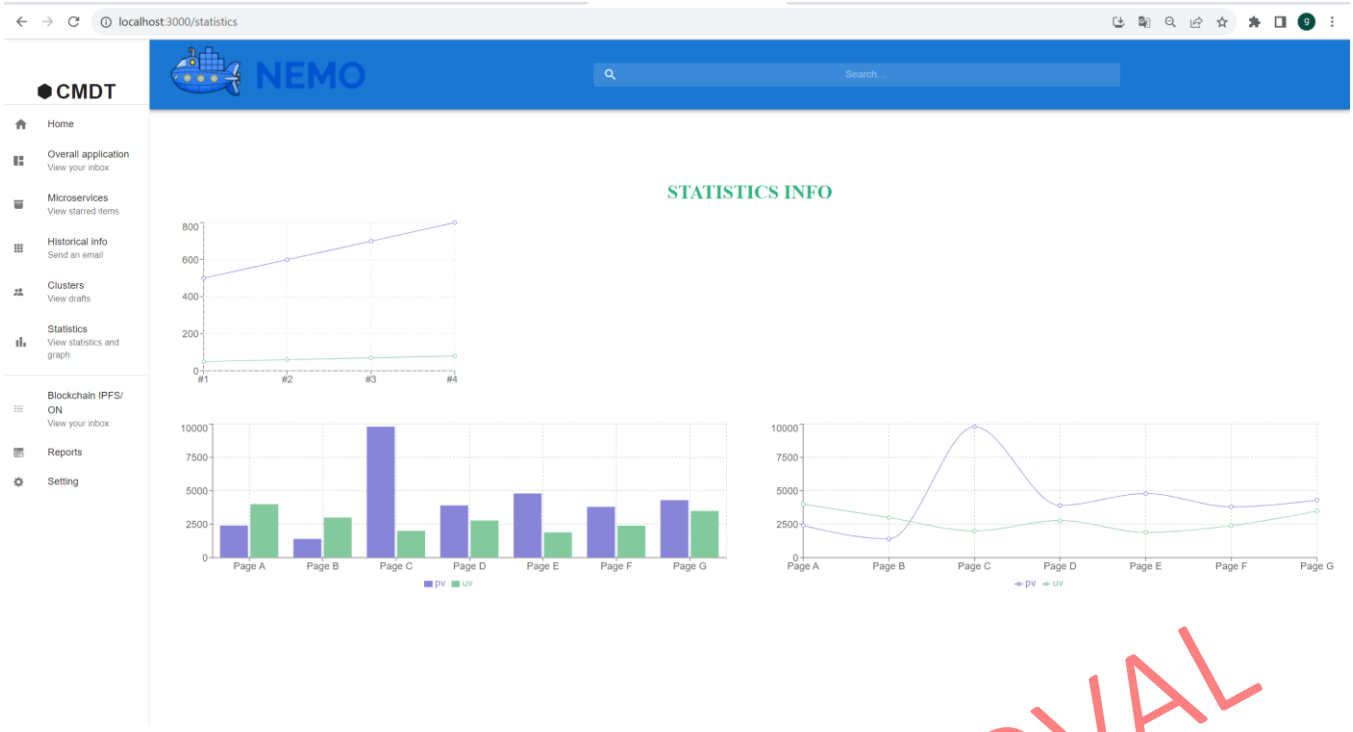


Figure 36: Dashboard web page

We will use different graphs (Histograms, continuous line, etc) to represent at full capacity and in the future, (obtaining data from other NEMO component on the APIs exposed by us) for example the number of clusters or CPU use which vary over time, or by displaying on the dashboard the number of downloads made over time of the descriptors and reliability and security indicators of the associated microservice, etc. (Figure 36).

6.1.3 Api documentation

The repository where you can download the PoC software, it is on Gitlab at the following url:

https://gitlab.com/giurbinati/nemo_cmdt/-/tree/develop?ref_type=heads

6.2 Meta-Network Cluster Controller

In this section we will describe the proofs of concept for the main technologies that we are using as the basis for the mNCC module.

6.2.1 ALTO exposure capabilities

ALTOs proof of concept has been realized with an own deployed implementation of the ALTO protocol described before. This implementation will be the basis for the NEMO contributions in the network exposure sub-module. The changes to be included will be related with the metrics exposed and the integration with the rest of NEMO's ecosystem.

The network testbed used to realize the proof of concept is a virtual environment where we count with 6 simulated routers and three client networks (Figure 37).:

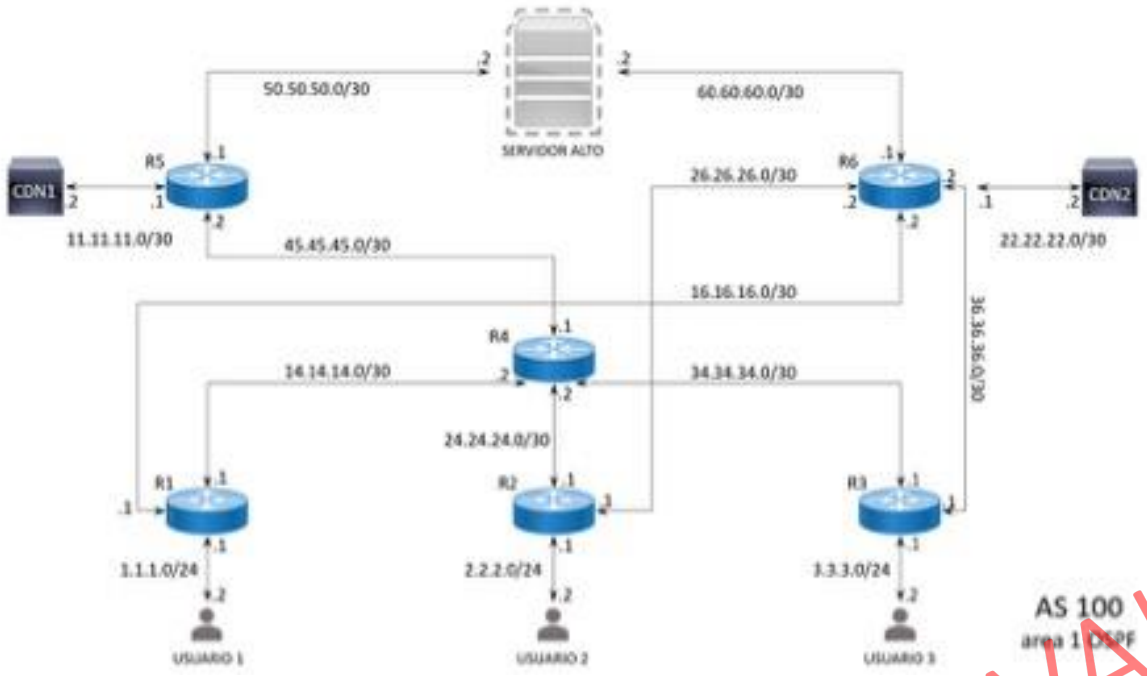


Figure 37: The network testbed used to realize the proof of concept

6.2.1.1 Model

The Data Model defined for the maps to be exposed is included in RFC 7285:

NetworkMap Data model

The NetworkMap will count with two sections: first a metadata section and then a data section. In the first one, it will be defined the version of the map returned and the resource id:

```
object {
  ResourceID resource-id;
  JSONString tag;
} VersionTag;

object {
  ResourceID resource-id;
  JSONString tag;
} VersionTag;
```

In the Data section it will be included a map of PID nodes associated with a list of prefixes that represents the client networks reachable through that node:

```
object {
  NetworkMapData network-map;
} InfoResourceNetworkMap : ResponseEntityBase;

object-map {
  PIDName -> EndpointAddrGroup;
} NetworkMapData;
```

```
object {
  NetworkMapData network-map;
} InfoResourceNetworkMap : ResponseEntityBase;

object-map {
  PIDName -> EndpointAddrGroup;
} NetworkMapData;
```

CostMap Data Model

The Costmap will count with two sections: first a metadata section and then a data section. Metadata section will be just like the metadata in the NetworkMap, but :

```
object {
  ResourceID resource-id;
  JSONString tag;
} VersionTag;

object {
  ResourceID resource-id;
  JSONString tag;
} VersionTag;
```


In the Data section it will be included a map of PID nodes associated with another map of PID nodes and a metric value. This representation makes reference to a matrix returned as a map of maps:

```
object {
  CostMapData cost-map;
} InfoResourceCostMap : ResponseEntityBase;

object-map {
  PIDName -> DstCosts;
} CostMapData;

object-map {
  PIDName -> JSONValue;
} DstCosts;

object {
  CostMapData cost-map;
} InfoResourceCostMap : ResponseEntityBase;

object-map {
  PIDName -> DstCosts;
} CostMapData;

object-map {
  PIDName -> JSONValue;
} DstCosts;
```

If we count with more than one cost metric available, it has to be notified in the metadata field, showing which one is being used:

```
object {
  JSONString cost-type-names<1..1>;
} CostMapCapabilities;

object {
  JSONString cost-type-names<1..1>;
} CostMapCapabilities;
```

6.2.1.2 Interfaces

To realize this Proof of Concept, we have used a HTTP API Rest interface, based on the Python3 module flask. The API is launched on a localhost, at port 5000 (Figure 38).:

```
root@altoserver-alberto:~/cdn-alto/alto-tid# python3 alto_core.py
IP ADDRESS 50.50.50.1 port 179
API running on http://127.0.0.1:5000
```

Figure 38: Code execution and API created.

And it counts with the next services available and therefore the next URIs (Figure 39).:

```
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# cat index.html.1

            <h1>ALTO PoC's API</h1>
            <h2>Services exposed:</h2>
            <p><ul>
            <li>All disjunts paths between A & B: <b><tt> /all/&ltstring:a&gt/&ltstring:b&gt </b></tt></li>
            <li>Shortest path between A & B: <b><tt> /best/&ltstring:a&gt/&ltstring:b&gt </b></tt></li>
            <li>Costs map: /costmap </li>
            <li>PIDs map: /networkmap </li>
            <li>Filtered Cost map: /costmap/filter/<string:pid></li>
            </ul></p>
        root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas#
```

Figure 39: Index of Resources available. 1

Execution

The main functionality to probe is the exposure of the Network map (Figure 40) and the Cost Map (Figure 41). Both shown in the model sub-section. Once the information is requested, it will be delivered with a metadata section that identifies the map and the relates the costmap with the network map associated.

```
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# wget localhost:5000/costmap
--2023-10-10 14:41:19--  http://localhost:5000/costmap
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:5000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1013 [application/json]
Saving to: 'costmap'

costmap                     100%[=================================================>]    1013  --.-KB/s    in 0s

2023-10-10 14:41:19 (258 MB/s) - 'costmap' saved [1013/1013]

root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# cat costmap
"{'meta':{'type':'alto-costmap+json','dependent-vtag':[{'resource-id':'networkmap-default','tag': '1696948876907609'}],'cost-t
ype': {'cost-mode' : 'numerical','cost-metric' : 'routingcost'}},'cost-map':{'pid100:0a0a0a03': {'pid100:0a0a0a03': 0, 'pid100
:0a0a0a04': 1, 'pid100:0a0a0a06': 1, 'pid100:0a0a0a01': 2, 'pid100:0a0a0a02': 2, 'pid100:0a0a0a05': 2}, 'pid100:0a0a0a01': {'p
id100:0a0a0a01': 0}, 'pid100:0a0a0a02': {'pid100:0a0a0a02': 0, 'pid100:0a0a0a04': 1, 'pid100:0a0a0a06': 1, 'pid100:0a0a0a03':
2, 'pid100:0a0a0a01': 2, 'pid100:0a0a0a05': 2}, 'pid100:0a0a0a04': {'pid100:0a0a0a04': 0, 'pid100:0a0a0a03': 1, 'pid100:0a0a0a
01': 1, 'pid100:0a0a0a02': 1, 'pid100:0a0a0a05': 1, 'pid100:0a0a0a06': 2}, 'pid100:0a0a0a05': {'pid100:0a0a0a05': 0, 'pid100:0
a0a0a04': 1, 'pid100:0a0a0a03': 2, 'pid100:0a0a0a01': 2, 'pid100:0a0a0a02': 2, 'pid100:0a0a0a06': 3}, 'pid100:0a0a0a06': {'pid
100:0a0a0a06': 0, 'pid100:0a0a0a03': 1, 'pid100:0a0a0a01': 1, 'pid100:0a0a0a02': 1, 'pid100:0a0a0a04': 2, 'pid100:0a0a0a05': 3
}}}"
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas#
```

Figure 40: Costmap request and response

```
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# wget localhost:5000/networkmap
--2023-10-10 14:42:12--  http://localhost:5000/networkmap
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:5000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 267 [application/json]
Saving to: 'networkmap'

networkmap                  100%[=================================================>]     267  --.-KB/s    in 0s

2023-10-10 14:42:12 (64.0 MB/s) - 'networkmap' saved [267/267]

root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# cat networkmap
"{'meta' : {'type':'alto-networkmap+json','vtag' : [{'resource-id':'networkmap-default','tag':'1696948876907609'}]},'network-m
ap':{'pid100:32323201': {'ipv4': ['0.0.0.0/0']}, 'pid100:0a0a0a03': {'ipv4': ['3.3.3.0/24']}, 'pid100:0a0a0a01': {'ipv4': ['1.
1.1.0/24']}}}"
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas#
```

Figure 41: Networkmap request and response

Also, in case the client requests for an unavailable resource, the system returns a standard response, as it is stablished in the HTTP definition (Figure 42).

```
root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas# wget localhost:5000/albacete
--2023-10-10 14:02:58--  http://localhost:5000/albacete
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (localhost)|127.0.0.1|:5000... connected.
HTTP request sent, awaiting response... 404 NOT FOUND
2023-10-10 14:02:58 ERROR 404: NOT FOUND.

root@altoserver-alberto:~/cdn-alto/alto-tid/pruebas#
```

Figure 42: Request and response for a non-existing resource (http default response).

6.2.2 Network performance probe

6.2.2.1 Interfaces

The network probe is deployed as a Docker container, configured with all the necessary settings for operation. This configuration encompasses critical information such as the host and port for network socket connections and the primary network features to be collected. (See Figure 43 – 44- 45)

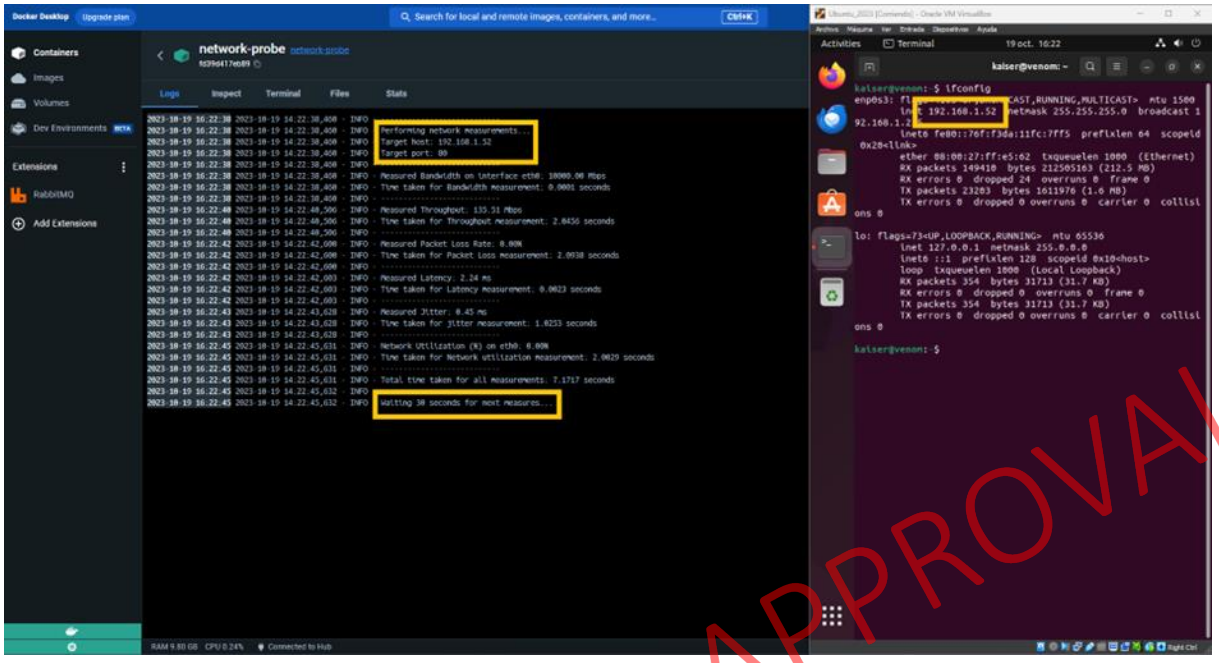


Figure 43: Network probe

The network probe offers two operational modes, each designed for different scenarios. The first mode is a live execution mode, where measurements are taken at predefined intervals, with a default delay of 30 seconds between each measurement. In the figure above, within the Docker interface, it is highlighted that the probe awaits 30 seconds before taking a measurement. The second mode involves waiting for external trigger communication to initiate measurements.

Within the range of available measurements, several have already been implemented and are functioning effectively. These include assessing bandwidth, throughput, packet loss, latency, jitter, and congestion on specific network interfaces. To validate the accuracy and reliability of these measurements, we conducted tests using the probe against a virtual machine. We implemented traffic control on a Linux system to introduce impairments into the network. First, latency impairments in the figure below.

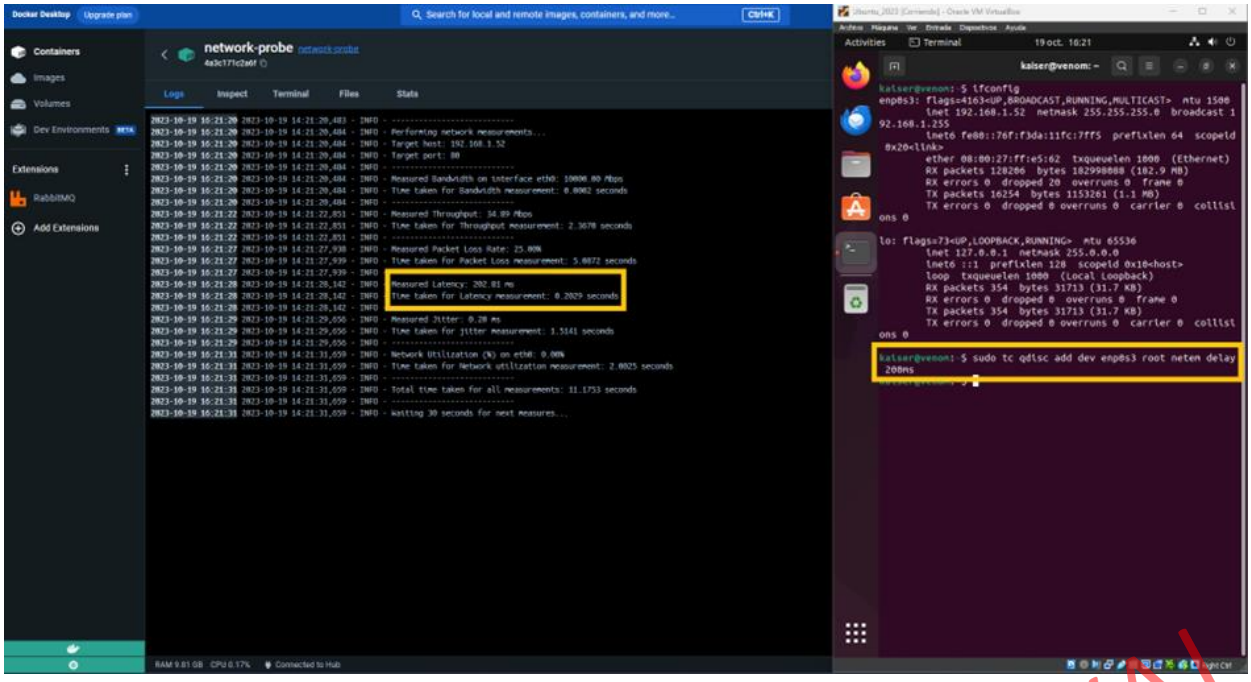


Figure 44: Latency impairments

As shown in the figure below, we conducted latency measurements by introducing a 200ms delay on the network interfaces. This resulted in a recorded latency of 202.81ms, which accounts for the original 2ms latency in the absence of impairments and the additional delay introduced by the impairments.

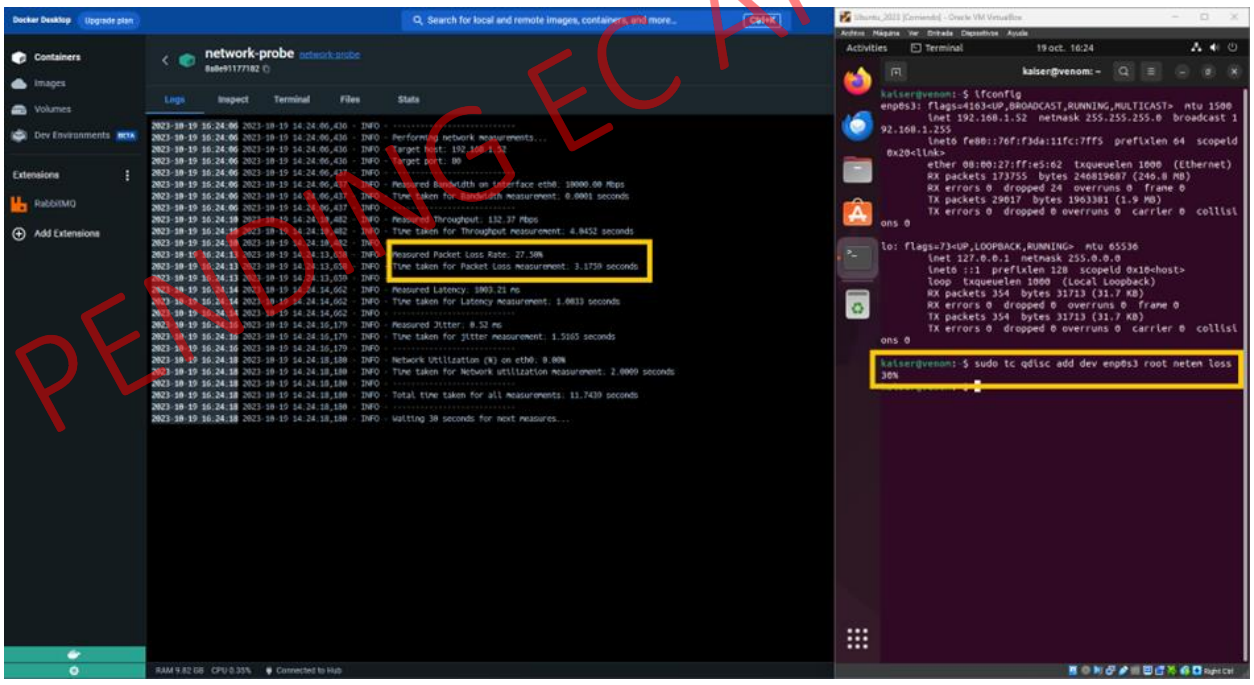


Figure 45: Delay introduced by the impairments

In a similar manner, we tested packet loss by introducing a 30% loss rate on the network interface, as depicted in the figure above. The results indicated a packet loss of 27.50%, contrasting with the baseline measurements where no packet losses were observed.

6.2.2.2 Api documentation

The network probe tool provides a range of execution options to customize and control its behavior. These options allow you to tailor the probe's operation according to your specific needs. Below is a list of available command-line options:

- --verbose:
  - Enable verbose output, providing detailed information during the execution.
- --host <target_host>:
  - Specify the target host's IP address. This option is required to define the host for network socket connection.
- --port <target_port>:
  - Specify the target port for network socket connection. The default port is 80.
- --live:
  - Enable live execution mode, which initiates measurements at predefined intervals.
- --delay <seconds>:
  - Set the delay in seconds for live execution. The default delay is 30 seconds.
- --duration <seconds>:
  - Define the test duration in seconds. The default duration is 4 seconds.
- --bandwidth:
  - Enable bandwidth measurement during the probe execution.
- --throughput:
  - Enable throughput measurement during the probe execution.
- --packet-loss:
  - Enable measurement of the packet loss rate during the probe execution.
- --latency:
  - Enable measurement of latency during the probe execution.
- --jitter:
  - Enable measurement of jitter during the probe execution.
- --congestion:
  - Enable measurement of network congestion during the probe execution.

These options offer the flexibility to configure and control the network probe's behavior, allowing you to collect the specific network data that is most relevant to the requirements.

6.3 CFDRL

The objective of the POC for CFDRL is to showcase the capacity to put together the following functionalities:

- Learn a RL policy.
- Learn in a distributed way thought Federated Learning.
- Implement poisoning attacks and robust aggregation strategies that foil the poisoning attacks.

The code is in Python and relies on the Flower Framework.

Input: A dynamic environment in the shape of Open AI gym type of module: This essentially means that the dynamics environment provides the current state and is able to be interacted with through the step function, step(action)-> (next_state, reward) that outputs the next state and immediate reward given the action chosen by the RL policy.

Output: the learned RL policy where states are mapped to recommended actions.

POC

— RL-FL

- RL Algorithm: DQN (Mnih et. al., 2015)
- RL environment: acrobot (quite simple env)
- FL aggregation: Vanilla Federated Averaging (McMahan et al., 2017)
- Shared information: The models
- Models: Neural Netwoks, MLP type implemented in Pytorch.
- FL Framework: Flower At the moment, with few interactions
- Observation: The RL agents are learning well independently, when not sharing the model (no FL) When sharing the model, the RL agents learn even better collaboratively

This code can be run and monitored in a local dashboard (Figure 46)

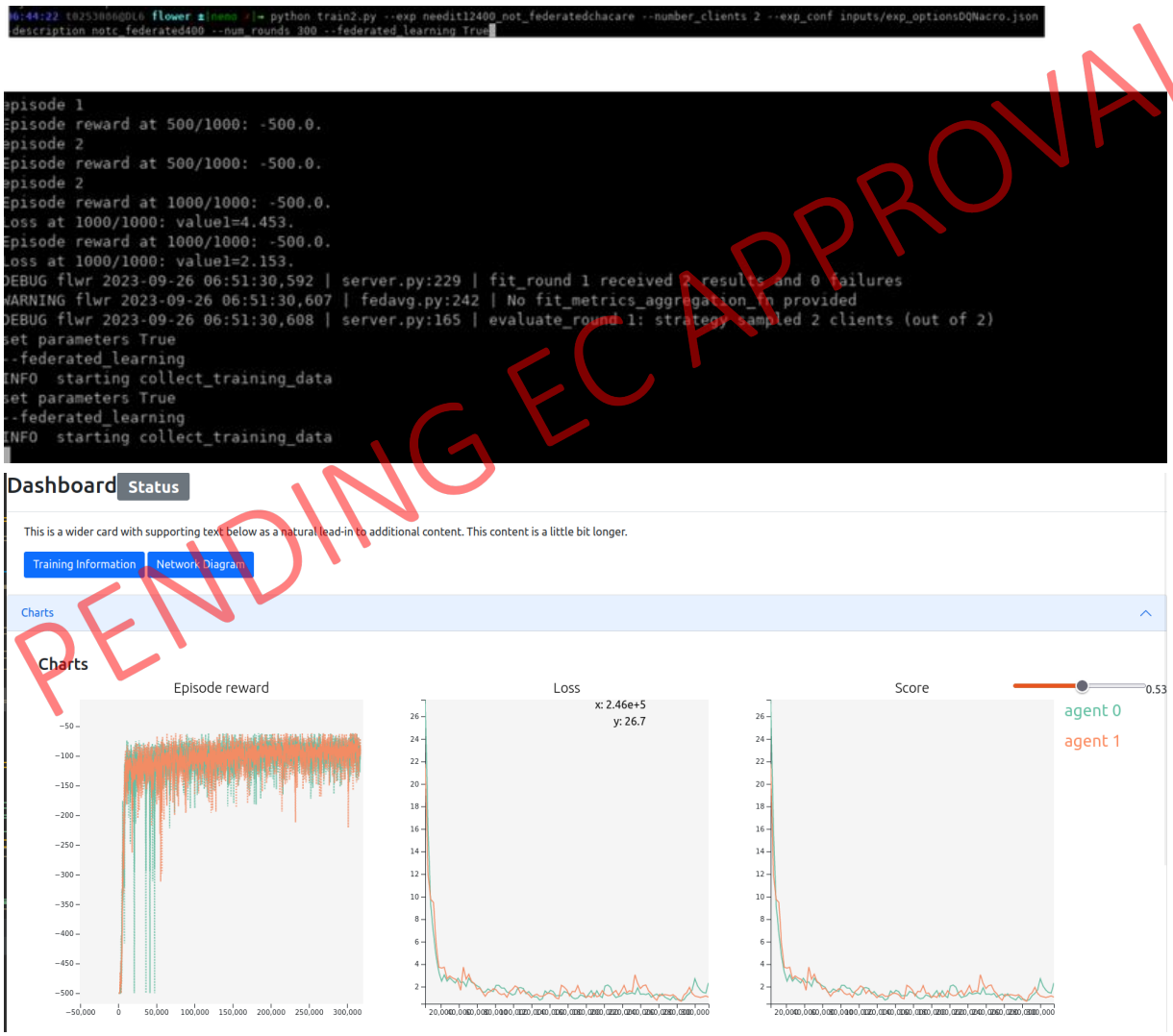


Figure 46: Network probe

The view above is our local dashboard displaying live the learning curves and the loss minimised by the models.

— Attacks and secure aggregation

A piece of code has been implemented in Flower that generates poisoning attacks from the Worker Nod /Data owners by adding adversarial noise to the shared model. The code has been shared internally between the partners and is executable on Google Colab.

— Combination

The next step is to combine these two pieces of code into one and combining their features.

Below is a code snippet of an implemented attack (i.e., weight poisoning attack) implemented in the flower framework

```
if self.perturbationVector == "InverseSign":
        perturbationVector = [-np.sign(best_parameter) for best_parameter in best_parameters]
    if self.perturbationVector == "InverseStd":
        parameters = [parameters for parameters, _ in total_weights_results]
        perturbationVector = []

        for parameterIdx in range(len(parameters[0])):
            arr = []
            for clientIdx in range(len(parameters)):
                arr.append(parameters[clientIdx][parameterIdx])
            numpyArr = np.stack(arr)
            perturbationVector.append(-np.std(numpyArr, axis=0))

    # optimize for gamma
    gamma_init = 100.0
    gamma = gamma_init
    gamma_succ = 0.0
    t = 0.000001 # threshold
    step = gamma_init

    while True: # this is a c++ do-while loop emulation
        malicious_parameters = []
        for idx in range(len(perturbationVector)):
            malicious_parameters.append(best_parameters[idx] + gamma*perturbationVector[idx])

                new_malicious_weights_results = [(malicious_parameters, num_examples) for _, num_examples in
malicious_weights_results]
        new_total_weights_results = new_malicious_weights_results + benign_weights_results

        are_equal = True
                        krum_parameters = aggregate_krum(new_total_weights_results, self.num_malicious_clients,
self.num_clients_to_keep)
```

```
        for idx in range(len(krum_parameters)):
            if np.array_equal(malicious_parameters[idx], krum_parameters[idx]) == False:
                are_equal = False
                break

        if are_equal:
            gamma_succ = gamma
            gamma = gamma + step/2
        else:
            gamma = gamma - step/2
        step = step/2

        if abs(gamma_succ - gamma) < t:
            break

        # print(gamma, gamma_succ)

    malicious_parameters = []
    for idx in range(len(perturbationVector)):
        malicious_parameters.append(best_parameters[idx] + gamma_succ*perturbationVector[idx])
            new_malicious_weights_results = [(malicious_parameters, num_examples) for _, num_examples in
malicious_weights_results]
    total_weights_results = new_malicious_weights_results + benign_weights_results
```

While in the following there is a snippet of the code implementing a certain countermeasure (i.e. Differential privacy) to FL attacks.

```
  def aggregate_fit(self, server_round: int, results: List[Tuple[ClientProxy, FitRes]],
    failures: List[Union[Tuple[ClientProxy, FitRes], BaseException]],) -> Tuple[Optional[Parameters], Dict[str, Scalar]]:

    if not results: # boilerplate code to handle exceptions
        return None, {}
    if not self.accept_failures and failures:
        return None, {}

    accepted_results = [] # get the privacy budget of each client
    disconnect_clients = []
    epsilons = []
    i = 0
    for c, r in results:
        histories[i] = r.metrics["history"]
        if r.metrics["accept"]:
            accepted_results.append([c, r])
            epsilons.append(r.metrics["epsilon"])
        else:
            disconnect_clients.append(c)
        i = i + 1
```

```
for c in disconnect_clients:
    c.reconnect(ReconnectIns(seconds=None), timeout=None)

results = accepted_results
if epsilons:
    self.max_epsilon = max(self.max_epsilon, max(epsilons))
print(f"Privacy budget ε at round {server_round}: {self.max_epsilon}")

# convert results
total_weights_results = [(parameters_to_ndarrays(fit_res.parameters), fit_res.num_examples) for _, fit_res in results]
```

We have also performed some initial experiments with this attack and the countermeasure, and we have the following results

The basic CIGAR10 reference FL model has a 40% accuracy after 5 epochs of training which is reduced to 30% when we applied the poisoning attack; thus the reduction of the accuracy is 25%. When we applied differential privacy we have 50 epochs in training (due to gradient clipping, virtual steps and noise) and the accuracy gets to 33% while the system is under poisoning attack; thus the reduction in the accuracy now is less than 20%. Obviously, this is a very basic experiment which has been done only for demonstrating the feasibility/effectiveness of our approach.

7 Conclusions

In this document, we have reported the work done in the WP2 so far. The goal of this initial report is to set up a solid foundation for the rest of the work to be done in the Workpackage and across the project. It regards mainly the specifications and the developments of the main asset of WP2 given the architecture created in WP1.
Task 2.1 extended the Meta-Level DT concept to the micro-services space, introducing the Cybersecure Micro-services' Digital Twins (CMDT). It has provided a detailed description of all the functionalities in place in the NEMO ecosystem providing also an initial PoC of the front and back end functionalities.
In parallel Task 2.2 introduced the CF-DRL concept leveraging local data for cloud-aggregated training to optimize offloading decisions, resource allocation, and caching. The CF-DRL also at the same time assured secure techniques for safe local ML training aggregation.
Lastly, Task 2.3 highlighted The NEMO meta-Orchestrator technical functionalities and the initial phase of the development for efficient decentralization in the IoT to Edge to Cloud Continuum with the main purpose to simplify the complexity of distributed computing, making it more manageable via intelligent decision-making and optimization workflows for efficiency.
These foundational works are a ground base for the ensuring a coherent progression towards achieving the overarching objectives of the project.

8 References

[1] G. Steindl and W. Kastner, "Semantic Microservice Framework for Digital Twins," *Applied Sciences*, vol. 11, no. 12, 2021.

[2] N. Moustafa and S. Jill, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," 2015.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[4] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, "Decentralized collaborative learning of personalized models over networks," in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 509–517.

[5] C. Hu, J. Jiang, and Z. Wang, "Decentralized Federated Learning: A Segmented Gossip Approach," *arXiv preprint arXiv:1908.07782*, 2019.

[6] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, "DisPFL: Towards Communication-Efficient Personalized Federated Learning via Decentralized Sparse Training," in *Proceedings of the 39th International Conference on Machine Learning*, 2022. [Online]. Available: https://proceedings.mlr.press/v162/dai22b.html

[7] H. Ye, L. Liang, and G. Li, "Decentralized Federated Learning with Unreliable Communications," *arXiv e-prints*, 2021, doi: 10.48550/arXiv.2108.02397.

[8] E. T. M. BELTRÁN, M. Q. PÉREZ, P. M. S. SÁNCHEZ, and others, "Decentralized Federated Learning: Fundamentals, State-of-the-art, Frameworks, Trends, and Challenges," *arXiv preprint arXiv:2211.08413*, 2022.

[9] N. RODRÍGUEZ-BARROSO, D. JIMÉNEZ-LÓPEZ, M. V. LUZÓN, and others, "Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges," *Information Fusion*, vol. 90, pp. 148–173, 2023.

[10] P. LIU, X. XU, and W. WANG, "Threats, attacks and defenses to federated learning: issues, taxonomy and perspectives," *Cybersecurity*, vol. 5, no. 1, pp. 1–19, 2022.

[11] F. H and Q. Q, "Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning," *Future Internet*, vol. 13, no. 4, p. 94, 2021, [Online]. Available: https://doi.org/10.3390/fi13040094

[12] Y. Zhao, X. Zhang, P. Peng, Y. Xu, Z. Xie, and N. Zheng, "Federated learning with differential privacy: A survey," *IEEE Trans Dependable Secure Comput*, vol. 18, no. 1, pp. 325–346, 2021.

[13] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1175–1191.

[14] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, p. 12, 2019.

[15] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Fedmc: A communication-efficient algorithm for privacy-preserving federated learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 7834–7844.

[16] Y. Aono, T. Hayashi, and T. Yamada, "Privacy-preserving deep learning via additively homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1287–1301.

[17] A. Ghosh, P. Mohassel, V. Rastogi, and D. Song, "Towards practical privacy-preserving analytics for iot and cloud-based systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2018, pp. 634–651.

[18] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated Reinforcement Learning: Techniques, Applications, and Open Challenges," Aug. 2021, doi: 10.20517/ir.2021.02.

[19] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*, 2020.

[20] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[21] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed Backdoor Attacks against Federated Learning," in *International Conference on Learning Representations*, 2020.

[22] X. Zhou and others, "Deep model poisoning attack on federated learning," *Future Internet*, vol. 13, no. 3, p. 73, 2021.

[23] M. Fang and others, "Local model poisoning attacks to byzantine-robust federated learning," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020.

[24] A. N. Bhagoji and others, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, PMLR, 2019.

[25] S. Kariyappa, A. Prakash, and M. K. Qureshi, "MAZE: Data-Free Model Stealing Attack Using Zeroth-Order Gradient Estimation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, 2021, pp. 13809–13818. doi: 10.1109/CVPR46437.2021.01360.

[26] Y. Chen, Y. Gui, H. Lin, W. Gan, and Y. Wu, "Federated Learning Attacks and Defenses: A Survey," *arXiv e-prints*, 2022, doi: 10.48550/arXiv.2211.14952.

[27] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *The 23rd International Symposium on Research in Attacks, Intrusions and Defenses*, 2020, pp. 301–316.

[28] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *USENIX Security Symposium*, 2014, pp. 17–32.

[29] M. Fredrikson, S. Jha, and T. Ristenpart, "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: Association for Computing Machinery, 2015, pp. 1322–1333. doi: 10.1145/2810103.2813677.

[30] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and Verifiable Federated Learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019, doi: 10.1109/tifs.2019.2929409.

[31] L. Lyu *et al.*, "Privacy and Robustness in Federated Learning: Attacks and Defenses," 2020.

[32] R. A. Mallah, G. Badu-Marfo, and B. Farooq, "On the Initial Behavior Monitoring Issues in Federated Learning," *IEEE Access*, vol. 9, pp. 161046–161054, 2021, doi: 10.1109/ACCESS.2021.3131102.

[33] R. Gosselin, L. Vieu, F. Loukil, and A. Benoit, "Privacy and Security in Federated Learning: A Survey," *Applied Sciences*, vol. 12, no. 19, p. 9901, 2022, doi: https://doi.org/10.3390/app12199901.

[34] A. P. Kalapaaking, I. Khalil, M. S. Rahman, M. Atiquzzaman, X. Yi, and M. Almashor, "Blockchain-based Federated Learning with Secure Aggregation in Trusted Execution Environment for Internet-of-Things," *IEEE Trans. Ind. Inform.*, 2022.

[35] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, "Privacy preservation in federated learning: An insightful survey from the GDPR perspective," *Comput Secur*, vol. 110, p. 102402, 2021, doi: https://doi.org/10.1016/j.cose.2021.102402.

[36] S. Augenstein *et al.*, "Generative Models for Effective ML on Private, Decentralized Datasets," in *Proceedings of the International Conference on Learning Representations*,

[37] B. Ghazi, R. Pagh, and A. Velingker, "Scalable and Differentially Private Distributed Aggregation in the Shuffled Model," *arXiv preprint*, 2019.

[38] H. Priya.N, A. H. S. M, S. G. S, and P. Rathika, "Improving Security with Federated Learning," in *2021 International Conference on Computational Performance Evaluation (ComPE)*, Shillong, India, 2021, pp. 234–239. doi: 10.1109/ComPE53109.2021.9752023.

[39] T. Ha, T. K. Dang, H. Le, and T. A. Truong, "Security and privacy issues in deep learning: a brief review," *SN Comput Sci*, vol. 1, no. 5, p. 253, 2020.

[40] Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, pp. 1333–1345, 2017.

[41] F. Wibawa, F. O. Catak, M. Kuzlu, S. Sarp, and U. Cali, "Homomorphic Encryption and Federated Learning based Privacy-Preserving CNN Training: COVID-19 Detection Use-Case," in *Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference (EICC '22)*, New York, NY, USA: Association for Computing Machinery, 2022, pp. 85–90. [Online]. Available: https://doi.org/10.1145/3528580.3532845

[42] J. Park and H. Lim, "Privacy-Preserving Federated Learning Using Homomorphic Encryption," *Applied Sciences*, vol. 12, no. 2, p. 734, Jan. 2022, doi: 10.3390/app12020734.

[43] D. Li and J. Wang, "FedMD: Heterogenous federated learning via model distillation," in *International Workshop on Federated Learning for User Privacy and Data Confidentiality*, Vancouver, Canada, 2019, pp. 1–8.

[44] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," in *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Machine Learning on the Phone and other Consumer Devices (MLPCD)*, Montreal, Canada, 2018.

[45] S. Saha, F. Bovolo, and L. Bruzzone, "Unsupervised deep change vector analysis for multiple-change detection in vhr images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 6, pp. 3677–3693, 2019.

[46] S. Saha and N. Sheikh, "Ultrasound image classification using acgan with small training dataset," *arXiv preprint arXiv:2102.01539*, 2021.

[47] S. Saha, Y. T. Solano-Correa, F. Bovolo, and L. Bruzzone, "Unsupervised deep transfer learning-based change detection for hr multispectral images," *IEEE Geoscience and Remote Sensing Letters*, 2020.

[48] S. Saha and T. Ahmad, "Federated Transfer Learning: concept and applications," *arXiv e-prints*, 2020, doi: 10.48550/arXiv.2010.15561.

[49] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, PMLR, 2018.

[50] J. Schulman and others, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[51] T. Rashid and others, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 7234–7284, 2020.

[52] C. Nadiger, A. Kumar, and S. Abdelhak, "Federated reinforcement learning for fast personalization," in *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2019, pp. 123–127.

[53] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems," *IEEE Robot Autom Lett*, vol. 4, no. 4, pp. 4555–4562, 2019.

[54] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported internet of things," *IEEE Access*, vol. 7, pp. 69194–69201, 2019.

[55] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., in Proceedings of Machine Learning Research, vol. 48. New York, New York, USA: PMLR, Nov. 2016, pp. 1928–1937. [Online]. Available: https://proceedings.mlr.press/v48/mniha16.html

[56] L. Espeholt *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*, 2018, pp. 1407–1416.

[57] D. Horgan *et al.*, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[58] A. Nair *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[59] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International conference on machine learning*, 2018, pp. 4295–4304.

[60] H. H. Zhuo, W. Feng, Y. Lin, Q. Xu, and Q. Yang, "Federated deep reinforcement learning," *arXiv preprint arXiv:1901.08277*, 2019.

[61] X. Fan, Y. Ma, Z. Dai, W. Jing, C. Tan, and B. K. H. Low, "Fault-tolerant federated reinforcement learning with theoretical guarantee," in *Advances in Neural Information Processing Systems*, 2021, pp. 1007–1021.

[62] S. Tzamaras, R. Ciucanu, M. Soare, and S. Amer-Yahia, "FeReD: Federated Reinforcement Learning in the DBMS," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 4989–4993.

[63] I. Goodfellow *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[64] X. Cao, G. Sun, H. Yu, and M. Guizani, "PerFED-GAN: Personalized Federated Learning via Generative Adversarial Networks," *IEEE Internet Things J*, vol. 10, no. 5, pp. 3749–3762, Mar. 2023, doi: 10.1109/JIOT.2022.3172114.

[65] J. Zhang, L. Zhao, K. Yu, G. Min, A. Y. Al-Dubai, and A. Y. Zomaya, "A Novel Federated Learning Scheme for Generative Adversarial Networks," *IEEE Trans Mob Comput*, pp. 1–17, 2023, doi: 10.1109/TMC.2023.3278668.

[66] A. Wainakh *et al.*, "Federated Learning Attacks Revisited: A Critical Discussion of Gaps, Assumptions, and Evaluation Setups," *Sensors*, vol. 23, no. 1, p. 31, Dec. 2022, doi: 10.3390/s23010031.

[67] L. Shi *et al.*, "Data poisoning attacks on federated learning by using adversarial samples," in *2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*, IEEE, Jul. 2022, pp. 158–162. doi: 10.1109/ICCEAI55464.2022.00041.

[68] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurr Comput*, vol. 25, no. 4, pp. 556–571, Feb. 2013, doi: 10.1002/cpe.2858.

[69] C. Dwork, "Differential Privacy," 2006, pp. 1–12. doi: 10.1007/11787006_1.

[70] I. Mironov, "Rényi Differential Privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, IEEE, Aug. 2017, pp. 263–275. doi: 10.1109/CSF.2017.11.

[71] L. F. Gonzalez, I. Vidal, F. Valera, and D. R. Lopez, "Link Layer Connectivity as a Service for Ad-Hoc Microservice Platforms," *IEEE Netw*, vol. 36, no. 1, pp. 10–17, Jan. 2022, doi: 10.1109/MNET.001.2100363.

[72] O. P. Kubernetes Official Documentation, " [Online]. Available at: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (accessed on October 24, 2023)."

[73] M. Mahalingam *et al.*, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," Aug. 2014. doi: 10.17487/rfc7348.

[74] O. N. O. S. (ONOS), open source S. controller for building next-generation S. solutions, "Open Networking Foundation, "[Online]. Available at: https://opennetworking.org/onos (accessed on October 24, 2022)."

[75] Ricardo Vilalta, "https://teraflow-h2020.eu/blog/upcoming-etsi-teraflowsdn-release-2-features."

[76] D. K. R. V. L. G. R. M. J. M. S. G. S. P. V. O. G. de D. P. S. A. F. and D. K. T. Xirofotos, "D3.1: Preliminary Evaluation of Life-cycle Automation and High Performance SDN."

[77] A. M. Da Costa and L. M. C. Murillo, "Integration of Network Slice Controller for Enhanced Intent-Based Networking in 5G/6G Networks," in *Proceedings of the 18th Workshop on Mobility in the Evolving Internet Architecture*, New York, NY, USA: ACM, Oct. 2023, pp. 31–36. doi: 10.1145/3615587.3615989.

[78] R. R. J. T. B. W. X. L. D. D. S. B. Luis M. Contreras, "IETF Network Slice Controller and its associated data models".

[79] R. Alimi *et al.*, "Application-Layer Traffic Optimization (ALTO) Protocol." 2014.

[80] D. King and A. Farrel, "A PCE-based architecture for application-based network operations," 2015.

[81] Luis M. Contreras, "Considering ALTO as IETF Network Exposure Function".

[82] Y. R. Y. Y. L. D. D. S. R. L. M. C. Qin Wu, "Application-Layer Traffic Optimization (ALTO) Performance Cost Metrics".

[83] [Online]. Available: https://kafka.apache.org/documentation/., "Apache Kafka,".

[84] K. Samdanis, M. Dohler, D. Sabella, and D. Giustiniano, "5G networks: Vision, requirements and challenges," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 74–80, May 2014.

[85] Y. Liu and Y. Li, "A survey of 5G network architecture," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2494–2530, Nov. 2016.

[86] Y. Li, M. Peng, and Y. Liu, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3180, Nov. 2019.

[87] Q. Wu, Y. Zeng, and R. Zhang, "5G wireless communications: opportunities and challenges," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 20–28, Jan. 2015.

[88] S. Parkvall, E. Dahlman, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*. Academic Press, 2018.