Toward Securing Federated Learning against Poisoning Attacks in Zero Touch B5G networks

Sabra BEN SAAD*, Bouziane BRIK‡, Adlen KSENTINI* *Senior, IEEE,*

***Abstract*—The zero Touch Management (ZSM) concept in 5G and Beyond networks (B5G) aims to automate the management and orchestration of running network slices. This requires heavy usage of advanced deep learning techniques in a closed-loop way to auto-build the suitable decisions, enabling to meet network slices' requirements. In this context, Federated Learning (FL) is playing a vital role in training deep learning models in a collaborative way among thousands of network slice participants while ensuring their privacy and hence network slice isolation. Specifically, running network slices may share only their model parameters with a central entity, e.g., Inter Domain Slice Manager, to aggregate them and build a global model. Thus, the central entity does not directly access the training data. However, FL is vulnerable to poisoning attacks, where an insider participant may upload poisoning updates to the central entity so that it can cause a construction failure of the global model and thus affect its global performance. Therefore, it is crucial to design security means to detect and mitigate such threats. In this paper, we design a novel framework to automatically detect malicious participants in the FL process. In particular, our framework first uses a deep reinforcement algorithm to dynamically select a network slice as a trusted participant, based mainly on its reputation. The selected participant will then be in charge of identifying poisoning model updates by leveraging unsupervised machine learning. We demonstrate the feasibility of our framework on top of a real dataset that we generate using the 5G OpenAirInterface (OAI) platform. Evaluation results show the efficiency of our framework in dealing with poisoning attacks even with the presence of several malicious participants.**

***Index Terms*—Zero Touch Management (ZSM), 5G and Beyond, Network slicing, Federated Learning, Poisoning attack, reinforcement and unsupervised learning, Dimensional Reduction.**

I. Introduction

Fifth-generation and Beyond (B5G) networks are exponentially growing as key enablers of various applications related to multiple vertical industries [1]. These emerging applications are characterised by heterogeneous requirements, including ultra-low latency, high bandwidth and communication reliability, support of massive device density, etc. To support that, B5G systems are based on the network slicing concept, which relies on network softwarization, i.e., building flexible and virtual networks tailored to services, to allow building various applications on top of common physical resources (radio, computation, and network). Indeed, network softwarization is built on top of three new technologies: Network Function Virtualization (NFV), Software-Defined Networking (SDN), and Cloud computing (central and edge). In addition, a network slice is composed and described by a set of physical and virtual Network Functions (PNF and VNF), interconnected with each other, and deployed on top of a common physical infrastructure.

Besides, a network slice is described in terms of main requirements and targeted performance in a Network Slice Template (NST), which is exploited then by an orchestrator and management framework[1], to orchestrate and manage life-cycle of network slices; from the instantiation to the destruction steps. Recently, a new architecture, called Zero-touch network and Service Management (ZSM), emerged to automate the management and orchestration of running network slices [3]. This requires heavy usage of advanced Deep Learning (DL) techniques in a closed-loop way to auto-build the suitable decisions and meet network slices' requirements. Specifically, the traffic/data generated by network slices are first monitored and then used to build analytic functions (using DL mechanisms). In general, the analytic functions aim to monitor network slice performances or Service Level Agreement (SLA) to predict/detect any degradation or violations. Noting that the Key Performance Indicator (KPI) of a network slice can be computation-oriented, network-oriented, and service-oriented [4]. While computation- and network-oriented KPIs can easily be monitored, through the NFV Infrastructure (NFVI) manager and SDN controller, to build analytic functions. However, service-oriented KPIs are difficult to be shared due to their confidentiality and privacy nature since they are directly linked to a running vertical industry's application and service, such as the IP address allocation, response/processing time of a particular VNF, statistics on handled packets by a router.

In this context, Federated Learning (FL) is playing a vital role in training deep learning models in a collaborative way among thousands of network slice participants while ensuring their privacy, hence network slice isolation. Rather than uploading their data to train their models, running network slices share only their local model parameters, during several rounds, with a central entity, e.g. Inter Domain Slice Manager, to aggregate them and build a global model. The central entity thus does not have direct access to the training data. Therefore, FL is more than required to create analytic functions about service-oriented KPIs of running network slices while ensuring their confidentiality and privacy.

However, FL is vulnerable to poisoning attacks, where an insider participant, a malicious network slice, may upload poisoning updates to the central entity so that it can cause a construction failure of the global model. For instance, a

*Eurecom, Sophia Antipolis, France. ‡University of Bourgogne, France. This work has been partially supported by the European Union's H2020 MonB5G (grant no. 871780) project.

[1] ETSI uses Management and Orchestration (MANO), while 3GPP uses the term Communication Service Management Function (CSMF), to refer to the framework that will orchestrate and manage life-cycle of network slices.

malicious participant may consider poisoned latency values in building its local model in such a way that the aggregated global model cannot then detect/predict latency-related SLA violations. Another example is the analytic function. It implements an intrusion detection system, that build poisoning models by one/several participant(s). The function can prevent the detection of system intrusions, even with one participant presence. Thus, poisoning attacks may affect the global performance of FL-based models for running network slices as well as the whole system. Therefore, it is crucial to design security means to detect and mitigate such threats.

In this paper, we design a novel framework to automatically detect malicious participants in the FL process. First of all, based on a real test bed, we generate a realistic dataset that focuses on the latency as a service-oriented KPI of running network slices. We focus here on the latency experienced by the key element of 5G CN on-boarded in a network slice, namely Access and Mobility Management Function (AMF). This dataset is then exploited to build an analytic function about the latency prediction in a federated way. In addition, the basic idea of our framework is to select dynamically one participant as a trusted entity by leveraging deep reinforcement learning. The trusted participant will then be in charge of identifying poisoning model updates using unsupervised machine learning. The main contribution of this work are summarized as follows:

- We use OpenAirInterface (OAI)[2] to generate a real dataset about the latency KPI of the AMF run as a VNF. This service-oriented KPI corresponds to the response time to handle UE attach requests when considering different configurations, such as available RAM memory and the number of CPUs.
- Exploiting our dataset, we build a DL-based model in a federated way between several running network slices. Our model enables us to predict the latency of the AMF function and hence anticipate any latency-related SLA violation.
- We also build an online Deep Reinforcement Learning (DRL) model that dynamically selects a network slice as a trusted participant at each federated learning round, i.e., when participants send their local models towards the central node, based on several metrics such as their reputations (participants).
- At each FL round, the trusted participant applies a dimensionality reduction scheme and unsupervised machine/deep learning to detect the poisoned model(s) and hence malicious participant(s).

The rest of this paper is organized as follows. In Section II, we present a review of related works. Section III describes the design and specification of the proposed framework that detects and mitigates the malicious updates due to the model/data poisoning attacks. In Section IV, we evaluate our framework. Finally, section V concludes the paper.

II. Related Work

A few solutions have been designed to deal with poisoning attacks when building learning models relaying on FL. These works can be classified into two main categories: works dealing with poisoned local models [5] [6] and those dealing with data poisoning of malicious participants [7] [8].

A. Local Model Poisoning

The objective of model poisoning attacks is to poison the local model updates of the FL clients before sending them to the FL server or inserting hidden backdoors into the FL global model. This attack impacts thus the performance of the global model by giving misclassifications or wrong predictions. In [5], the authors demonstrated how the federated learning (FL) model could be poisoned. They showed that any malicious clients could introduce hidden backdoor functionality into the joint global model, e.g., to ensure that an image classifier model can predict labels for some input data, which were introduced (labels) by the malicious client. The authors designed a new model-poisoning attack based on model replacement and evaluated it on top of several assumptions on the standard FL. Another scheme is proposed in [6] to study the resilience of distributed implementations of Stochastic Gradient Descent (SGD) against Byzantine failures, including network asynchrony, software bugs, and attackers aiming to compromise the whole system as well as biases in local datasets. The authors first showed that current approaches do not tolerate Byzantine failures. Then, they proposed a resilience property of the aggregation rule that can ensure model convergence despite Byzantine participants. Although FL introduces new application scenarios in B5G networks, such as edge computing and on-device learning, it inherits the same critical threats, such as the poisoning and membership inference attacks, as in the other contexts. A novel blockchain-based FL scheme is designed in [9] to deal with model poisoning attacks. This scheme is based on blockchain to create smart contracts and hence prevent malicious clients from involving in the FL process. Therefore, the central server may easily identify unreliable clients by executing smart contracts.

B. Data Poisoning

This type of attack is known as contamination of the training data. It takes place during the FL training phase of the machine learning model. Generally, a malicious node tries to poison the training data by injecting carefully designed samples to compromise the whole FL learning process. This attack is also called a“dirty-label poisoning attack”. In [7], the authors first studied the threats that may target federated learning in terms of Sybil-based poisoning attacks. Then, they designed a new detection and mitigation scheme that identifies poisoning Sybil-based on clients' updates. The basic idea of this scheme is to use an adaptive learning rate at each client level based on the similarity between inter-client contributions. The authors also showed that their scheme may deal with the existing poisoning attacks, such as backdoor poisoning attacks and Sybil-based label-flipping. Noting that the label-flipping attack

[2] https://openairinterface.org/

TABLE I: Comparison of poisoning attack detection solutions.

Works	Poisoning Attack type		Securing side		B5G	Used Technique
	Model Poisoning Attack	Data Poisoning Attack	Client	Server		
Yi Liu, *et al.* [9]	✓		✓		✓	Smart Contract and Blockchain
Peva Blanchard, *et al.* [6]	✓			✓		A resilience property of the aggregation rule
Phillip Rieger, *et al.* [16]		✓		✓		Analyzes the parameter updates of the model's output layer
Mustafa Safa Ozday, *et al.* [17]		✓		✓		Adjusting the aggregation server's learning rate
M. Jagielski, et al. [15]		✓	✓			Model Robustifying: Trimmed versions of the loss function
Y. Zhao,*et al.* [12]	✓			✓		Generative adversarial network (GAN)
M. Subedar, et al. [14]		✓	✓			Probabilistic modeling of deep features
J. Steinhardt, et al. [13]		✓	✓			Training Data Filtering Input Manipulation Detection
Our Solution	✓	✓	✓		✓	**Supervised/Unsupervised Learning: Dimensional Reduction Algorithms, Deep Reinforcement Learning**

is a special case of data poisoning, where the labels of two input observations are flipped while data features (inputs) are kept unchanged. Another example of a poisoning attack is backdoor data poisoning [8], where an adversary can modify individual features or small regions of the original training dataset (some pixels of images) to embed backdoors into the FL model. As an example, the attacker creates a stamp on an input image so that the FL model behaves according to the adversary's objective if the input contains the backdoor feature.

The work proposed in [16] introduced DeepSight, a novel model filtering approach that mitigates backdoor attacks on FL. It is based on different techniques that allow characterizing the distribution of data used to train model updates. They proposed a threshold metric to build a classifier that analyzes the parameter updates of the model's output layer in order to measure the homogeneity of its training data. In [17], the authors proposed a lightweight defense mechanism that requires minimal changes to the FL process to prevent backdoor attacks. The defense is based on adjusting the aggregation server's learning rate, per dimension and per round, based on the sign information of agents' updates.

C. *Discussion and comparison*

Even though there are some works addressing the challenge of how to deal with poisoning attacks in the FL context, however, most of them focus on how to adjust learning parameters at the clients' side, such as learning rate [7], or designing new rules for local models aggregation at the central node side [6], in order to be able in detecting malicious participants. In addition, these works dealt with either local model poisoning or data poisoning attacks, not with both. In B5G networks context, few works have been proposed that are mainly based on blockchain to prevent the participation of malicious clients [9]. However, blockchain will require more computing and storage resources in addition to those needed by B5G networks for their management and orchestration. Moreover, this work can only evaluate the updates every round, using a mandatory test data set as parameters input in the "evaluate function", which is implemented in the smart contract. More than that, the increased number of participants in the proposed solution affects the communication overhead and the delay, which in turn reduces the accuracy of the FL model. Additionally, the proposed framework is a costly solution.

In Table. I, we compare existing works according to several criteria such as targeted poisoning attack, the used techniques, secure side (client or central node), and B5G context or not.

III. Our Trust federated deep learning Framework

In this section, we describe our framework to secure federated learning in B5G networks against data and model poisoning attacks, named "TQFL" for "Trust deep Q-learning Federated Learning". The design of our framework comprises four main steps, starting from generating a realistic dataset to designing a detection scheme of poisoning attacks: (i) The generation of a realistic dataset about the AMF function's latency of running network slices and its (latency) related parameters. (ii) Building a deep learning model to predict the AMF function's latency of each running network slice in a federated way in order to prevent any latency-related SLA violation. (iii) Building an online Deep Reinforcement Learning (DRL) model that dynamically selects a network slice as a trusted participant (see step 1 in Fig. 1). (iv) After the first FL rounds (see steps $2, 3$ in Fig. 1), the trusted participant applies a dimensionality reduction scheme and unsupervised machine/deep learning to detect the malicious participant (s) (see steps $4, 5, 6$ in Fig. 1).

Before we proceed, we first give an overview of our framework in the next subsection.

A. *Overview of TQFL Framework*

As depicted in Fig. 1, we consider n running network slices that may be initiated by different vertical industries, such as intelligent transportation, Industrial IoT, and eHealth verticals.

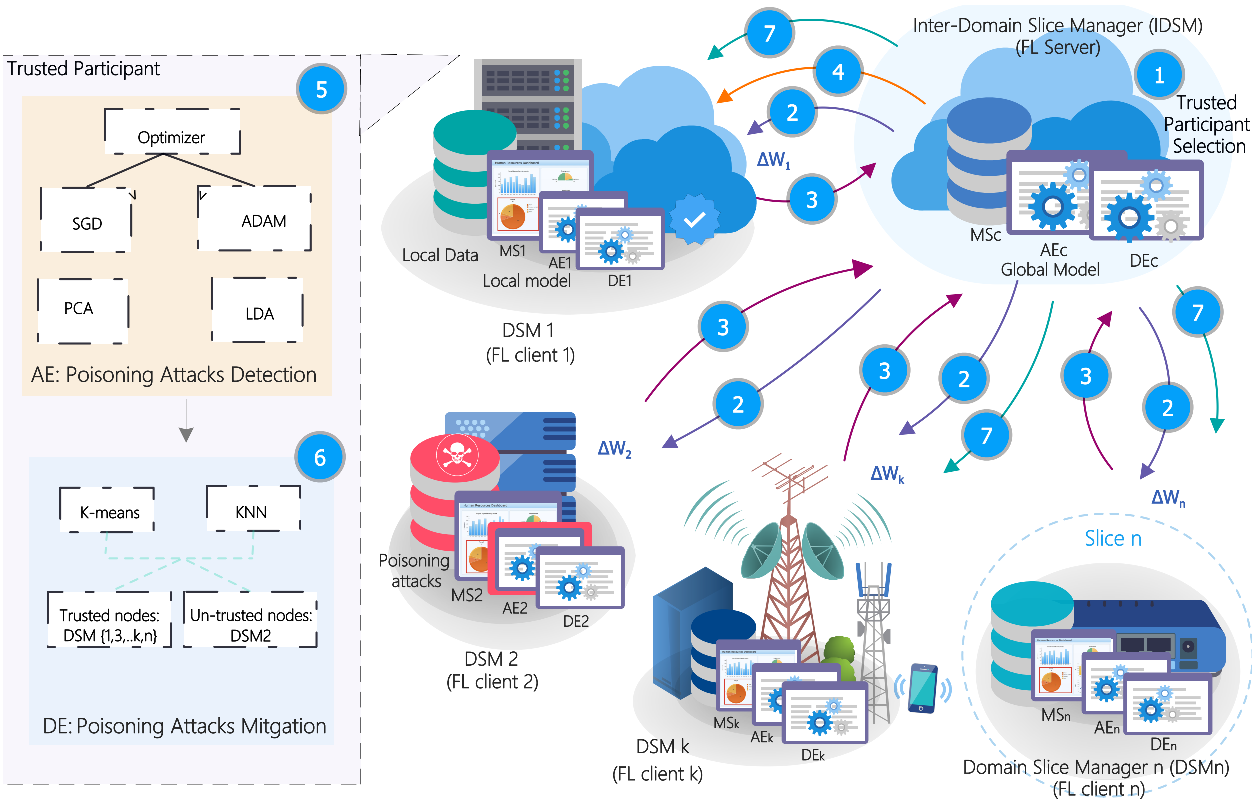


Fig. 1: Overview of TFQL Architecture.

The running network slices are interconnected to an Inter-Domain Slice Manager (IDSM), which is in charge of the management and orchestration of network slices. To enable Zero Touch Management (ZSM), the IDSM side includes an Analytic Engine (AE) for building learning models and a Decision Engine (DE) to make suitable decisions based on AE's outputs. On the other side, each running network slice is managed locally by a Domain Slice Manager (DSM), which also includes a Monitoring System (MS) for monitoring data and in-slice traffic (i.e., KPIs) from different VNFs, and an Analytic Engine (AE) for building learning models.

B. Generation of Realistic Dataset

Machine/Deep learning algorithms require data to create learning models. However, the more the datasets are realistic and large, the more the deep learning models are accurate and adaptable for various situations. Hence, the first critical step toward developing accurate learning models is the data set collection (data acquisition or monitoring). With the lack of a real dataset, we conducted a real testbed using Eurecom OpenAirInterface (OAI) platform to generate a realistic dataset called EARCD for Eurecom AMF Resource Consumption Dataset. OAI implements 5G radio access and core networks as open-source software. We emulated ten instances of Access Mobility Function (AMF), running as VNFs inside ten isolated network slices. The network slices differ from each other in terms of their AMFs' configurations. For instance: the AMF of network slice 1 has 1GB of memory, and 1 CPU, the AMF of network slice 2 has 2GB of memory and 2 CPUs, etc. In addition, we used my5G-RANTester [3] tool to emulate user equipment (UEs) and one gNB. my5G-RANTester enables to emulate data and control planes of UEs and gNBs. my5G-RANTester relies mainly on the release 15 of 3GPP standard about NG-RAN (Next Generation-Radio Access Network). We used my5G-RANTester tool to generate attach request packets, which will be then handled by the different network slices' AMFs. By increasing the number of UEs, we are able to generate up to 560 attach request per second, covering different traffic densities. Besides, it is clear that the latency values increase as the number of attach request increases. However, the latency values depend greatly on the configurations of network slices' AMFs. Fig. 2 depicts the average latency values according to the received number of attach requests for two different configurations of AMFs' VNFs. We clearly observe that AMF with 2048 MByte of memory and 2 CPUs succeeds in decreasing the latency when compared to AMF with 1024 MByte and 1 CPU. Therefore, we generated ten local datasets (ten network slices), by varying the number of handled attach request/s, where each dataset contains 2813 samples (rows). In addition, each local dataset contains five features as input data, including RAM capacity, CPU capacity, RAM used, CPU used, the number of attach requests, and latency in terms of average duration of UEs attachment as output data. The latter corresponds to the response time (latency in second) to handle UE attach requests

[3] https://github.com/my5G/my5G-RANTester

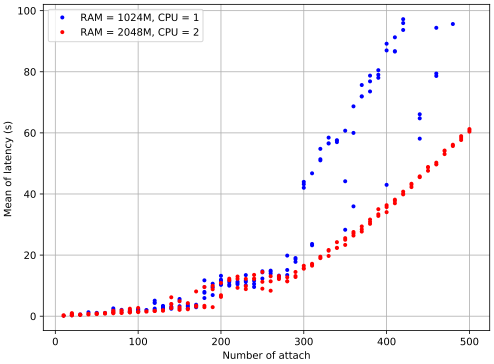


Fig. 2: Latency of multiple attach with different configurations of CPU and RAM.

by the network slices' AMFs.

C. Latency KPI Prediction in Federated Way

To enable zero-touch management of the latency KPI related to the UEs attach requests, we built a deep learning model that enables each domain slice manager (DSM) to predict the average needed latency for UEs attach requests in a federated way. As depicted in Fig. 1, each running network slice includes Monitoring System (MS), Analytic Engine (AE), and Decision Engine (DE); they allow it to monitor needed information (used CPU, used memory, etc.), build local learning models, and make the suitable decisions based on made predictions, respectively. Thus, we create a deep learning model in a federated manner that comprises four main steps:

1) **Data Pre-Processing**: it is important to preprocess the collected data before feeding it to the deep learning model since the output of this step may directly affect the performance of the learning model. This step checks if the data is on the same format and scale, does not include null and redundant values, and includes all needed features for the training step.
2) **Learning Initialization**: In this step, the Inter-Domain Slice Manager (IDSM), as the central node, creates an initial global model and defines the learning hyper-parameters in terms of batch sizes, number of epochs, learning rate, neural network architecture, etc. These parameters are then sent to the network slices' DSMs, as clients. Noting that the network slices implement an artificial neural network (ANN), which comprises one input layer of 5 neurons (the five input features), seven hidden layers of 20 neurons, and one layer of 1 neurons (Latency value). In addition, the activation function of neuron nodes is rectified linear function, and two different optimizers are used: Stochastic Gradient Descent (SGD) and ADAM optimizers (see subsection IV-A for more details).
3) **Training of Local Models**: each network slice's DSM, through its AE, updates the parameters of its local model. Indeed, each DSM gathers needed data through its MS and splits it into many batches. Then, the DSM's AE performs the average gradient on each batch, with respect to the current model, during several epochs and with a particular learning rate. Once done, the network slices' DSMs send their local models back to the central IDSM.
4) **Building of Global model**: the central IDSM aggregates the DSMs' local models, using *FedAvg* algorithm proposed in [10], which is based on distributed SGD, as weights optimizer. The aggregated global model is then sent back to the DSM nodes. Algorithm 1 describes the main instructions performed by the central IDSM.

Algorithm 1 The central IDSM

```
Require: Iterations $k$, Network Slices $N$.
Ensure: Aggregated Model $L^{i+1}$.

Initialize $L_0$
$i$ = 1
while  $i \leq k$ do
    $j$ = $|1|$
    while $j \leq |N|$ do
        $R_j^{i+1} \leftarrow DSMUpdate(j, R^i)$
    end while
    $L^{i+1} \leftarrow \frac{1}{|N|} \sum_{t=1}^{t=|N|} R_t^{i+1}$
end while
return  $L^{i+1}$ to Network slices' DSMs. =0
```

D. Poisoning Attacks Detection

In this section, we present our poisoning attack detection scheme.

1) Trust Participant Selection using Deep Reinforcement learning: After the first FL rounds, the central IDSM of running network slices selects a running network slice (DSM) as a trusted node. To do so, we design a new deep reinforcement learning-based model to derive an optimal policy about trust node selection while considering several criteria related to such nodes, such as their reputation, detection rate of malicious nodes, and their accuracy in building learning models. Deep Reinforcement learning is a process that enables one or a set of agents to learn on how to make suitable decisions through error and trial and based on their (agents) previous experiences. Specifically, each agent interacts with the environment to receive either penalties or rewards for actions it made. Hence, the main objective of deep reinforcement learning is to derive an optimal policy about agents' actions that maximizes agents' cumulative reward. In our study, the central IDSM is the agent that interacts with running network slices' DSM (environment) in discrete time steps, as shown in Fig.1. In what follows, we first formalize our problem using Markov Decision Process (MDP). Then, we apply the Deep Q-Network algorithm (DQN) to predict the best trust participant to select at each federated learning stage. Markov Decision Process Model The problem is often modeled using a MDP, where, at every timestep t, the central IDSM manager, is in a state s_t, takes an action a_t. Then, it (IDSM) will receive a scalar reward from the network slices' DSM (environment). In addition, the system passes from the state s_t to a state s_{t+1}, according to environment dynamics $p(s_{t+1}|s_t, a_t)$. Therefore, the central IDSM manager attempts to learn a policy $(a|s)$, that helps to map from observations to actions, and maximizing its rewards. In our study, a state $s \in S$ is a three-sized tuple $(MC; TC; GMA)$, where:

- MC_i is the number of malicious models (participants), that are detected by network slice's DSM_i;
- TC_i is the number of trust models, that are detected by network slice's DSM_i;
- GMA_i is the accuracy of the global model, after detecting and removing malicious models, by network slice's DSM_i;

At every timestep t and when aiming to take an action a, the central IDSM can select a network slice i, among the n running slices, that has the highest reputation Rep. Then, the system may transit to a new state s_{t+1}, that corresponds to the number of detected malicious clients MC_i, the number of trust clients as well as the new accuracy of the new global model. Furthermore, when the system moves to a new state s_{t+1}, a reward $r_{t+1} = R(s, a)$ is associated with the transition $p(s_{t+1}|s_t, a_t)$. For this end, we model the reward that the central IDSM expects to get from the selected network slices' DSM, as follows:

$$r_{t+1} = \begin{cases} \text{GMA}_{t+1} & \text{if } MC_i >= 1 \text{ or } GMA_{t+1} >= GMA_t \\ \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

It is clear that the received reward from a participant i affects directly its reputation in the federated learning process. Additionally, the reward increases, when the reputation of the network slice i increases as follows:

$$Rep_i = Max\left(0, r_{t+1}\right) \tag{2}$$

Based on equation 1, the central IDSM's reward will be affected not only by the number of observed malicious participants but also by the accuracy of the new global model. Therefore, the IDSM manager aims to derive the optimal policy in selecting a trusted participant, which optimizes the learning model's accuracy and its detection of poisoning attacks. Deep Reinforcement Learning As we mentioned before, our reinforcement scheme is based on DQN, which combines Q-learning and a deep neural network to learn an optimal policy from input data. Q-learning is a reinforcement learning algorithm that aims to identify the optimal policy of action selection, maximizing the total reward, called Q-value, for any finite MDP. The Q-value of each action is calculated and stored in a table named Q-table. In addition, at every timestep (learning episode), the Q-values are updated using the following formula:

$$Q\left(s_t, a_t\right) = Q\left(s_t, a_t\right) + \Theta\left(r_{t+1} + \lambda \underset{a \in A}{max} Q(s_{t+1}, a)\right) \tag{3}$$

With Θ is the learning rate and λ is the discount factor, indicating the future rewards importance. Besides, the basic idea of DQN is to use a neural network to predict the Q-value of all possible actions based on the current state. In particular,DQN comprises two neural networks: target network $Q'(s'; a; \theta')$ and a prediction network $Q(s; a; \theta)$. The prediction network is updated after each learning epoch, while the target network is directly updated from the prediction network after every several iterations. Hence, DQN aims to reduce the loss function between both neural networks as follows:

$$L = \left(r + \lambda \underset{a' \in A}{max} Q'(s', a', \theta') - Q(s, a, \theta)\right)^2 \tag{4}$$

Where θ is the learning weights of the Q-network, that is updated using gradient back propagation optimizer. In our study, we implement a fully connected neural network. It comprises one input layer with three neurons that correspond to the three states (MC, TC, GMA), two hidden layers with 24 neurons each, and one output layer to predict the Q-values of the running network slices. Noting that, we tried several configurations in terms of the number of intermediate layers and their number of neurons. We selected the best configuration that provided better performance.

2) Dimensionality Reduction of model updates:

Once a trusted client is selected, it will be in charge of detecting whether the received updates include a malicious model or not. First of all, the selected trust client receives the n model updates of network slices from the central IDSM, and then applies dimensionality reduction techniques to be able to present the model updates in 2D dimensions. Indeed, dimension reduction is the transformation of a dataset/matrix from a high-dimensional space into a low-dimensional space; in such a way, the low-dimensional representation retains the meaningful properties of the original data.
Fig. 3 shows a dimensionality reduction of our FL model updates in 3D when applying the linear discriminant analysis reduction technique. The different colors of the points, which are defined by 3 axes (x:0, y:1, z:2), present the updates of different nodes (10 clients). However, as we can notice, the reduction to 3D will not provide a clear visualization to interrupt and classify the model updates. That is why we decided to apply dimensionality reduction to 2dimensions (2D).

In our study, to design an efficient detection scheme, we apply two different techniques on the FL clients updates, with two different optimizer (ADAM, and SGD): **(i) Principal Component Analysis (PCA)** as an unsupervised technique (ignores class labels) and **(ii) Linear Discriminant Analysis (LDA)** as a supervised technique.

i) Principal Component Analysis (PCA):

PCA is a statistical technique that helps explain data of high dimensions by extracting only some principal components of such data. The process of PCA comprises four main steps:

1) **Standardization of model updates' values**: the trusted client receives n model updates, where each one contains five values that correspond to the five input features, including RAM capacity, CPU capacity, RAM used,

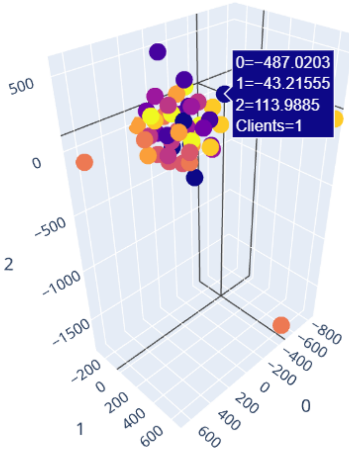


Fig. 3: LDA visualization with 3 dimensional applied on the node updates.

CPU used, and the number of attach requests. The first step of standardization consists of putting the feature values in the same range and format in order to make sure that they will equally contribute to the final analysis, using the following equation:

$$Str(value) = \frac{(value - mean)}{standarddeviation} \tag{5}$$

2) **Covariance Matrix**: The second step is the calculation of the "Covariance Matrix" between the input features of our dataset. Notably, the aim is to understand how these input features vary from the mean with respect to each other. The covariance matrix is a $m \times m$ symmetric matrix (where m is the number of input features, 5 in our case). In fact, the covariance matrix describes the correlations between all the possible pairs of our input features as follows:

$$\text{Cov(input features)} = \begin{pmatrix} Cov(x_1,x_1) & Cov(x_1,x_2) & \dots & Cov(x_1,x_m) \\ Cov(x_2,x_1) & Cov(x_2,x_2) & \dots & Cov(x_2,x_m) \\ Cov(x_3,x_1) & Cov(x_3,x_2) & \dots & Cov(x_3,x_m) \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ Cov(x_m,x_1) & Cov(x_m,x_2) & \dots & Cov(x_m,x_m) \end{pmatrix}$$

The correlation between the two features depends mainly on the covariance sign. If it is positive, both features increase or decrease together, which means they are strongly correlated. Nevertheless, if it is negative, one increases whereas the other decreases, which signifies that they are inversely correlated.

3) **eigenvectors and eigenvalues of the covariance matrix**: eigenvectors and eigenvalues are the core of PCA, enabling to identify the principal components. These parameters are new variables that are constructed as mixtures or linear combinations of the initial features. Indeed, the eigenvectors (principal components) describe the directions of the new feature space, while the eigenvalues show the variance of the data along the new feature axes. Both parameters are calculated using the following formula:

$$\begin{aligned} & Cov(inputfeatures) * v = \lambda * v \\ & Where: \\ & \qquad \textit{- Cov(input features) is the covariance matrix} \\ & \qquad \textit{- v is the eigen vectors} \\ & \qquad \textit{-}\ \lambda\ \textit{is the eigen value} \end{aligned} \tag{6}$$

$$FinalData = FeatureVector^T * StandardizedData^T \tag{7}$$

Therefore, eigenvectors and eigenvalues enable to create new g-dimensional data that gives g principal components.

4) **Computation of feature vector**: This step aims to select which principal components to keep and which ones to remove (those that have lesser significance or lower eigenvalues). The output of this step is a matrix of vectors named "Feature vector", which has the selected components as columns. Finally, the initial data are aligned with the new principal component, using equation 7. In addition, the feature vector with EigenVectors is used to align/reorient the data from original axes to new principal component axes.

ii) Linear Discriminant Analysis (LDA):

LDA is also a dimensionality reduction technique that aims to find not only the component axes, maximizing data variance (PCA), but also a feature subspace that maximizes class separability. Thus, LDA enables to project a feature space (a dataset with nb dimensional samples) into a smaller subspace k while maintaining the class-discriminatory information.

The LDA process also comprises different steps. First, we consider a matrix of n classes of model updates that correspond to the n clients (network slices) as follows:

$$\text{C} = \begin{pmatrix} C_1 \\ C_2 \\ \dots \\ C_n \end{pmatrix}$$

And each class comprises five input features, as follows:

$$F = \begin{pmatrix} F(1,c_1) & F(2,c_1) & \dots & F(5,c_1) \\ F(1,c_2) & F(2,c_2) & \dots & F(5,c_2) \\ F(1,c_3) & F(2,c_3) & \dots & F(5,c_3) \\ \dots & \dots & \dots & \dots \\ F(1,c_n) & F(2,c_n) & \dots & F(5,c_n) \end{pmatrix}$$

1) **Computing the d-dimensional mean vectors:** The first step of LDA is to compute a d-dimensional mean vector M(C_i) for the different classes n.

2) **Compute the Scatter matrix (in between class and within the class scatter matrix):** The second step is to compute the "Scatter Matrices" which are equivalent to the variance. In fact, we have two matrices of E x N dimensions to calculate: "The within-class" and the "between-class scatter matrix". The objective of these matrices is to determine the variability within a class (Intra class scatter) as well as between different

classes (inter-class Scatter). Both consist of calculating the distance between different points(inter/intra classes). The first matrix (within-class scatter) Sca is calculated as follows:

$$Sca = \sum_{j=1}^{n} Ss_j$$
$$Where:$$
$$Ss_j = \sum_{x \in Di}^{(number\_of\_samples)} (x - m_i)(x - m_i)^T, \tag{8}$$
$$Ss_j \textit{ is a scatter matrix for every class,}$$
$$m \textit{ is the mean vector,}$$
$$m_i = \frac{1}{n_i} \sum_{x \in Di}^{(number\_of\_samples)} x_k$$

Likewise, the class-covariance matrices are computed by adding the scaling factor $\frac{1}{N-1}$ to the within-class scatter matrix so that the equation 8 becomes:

$$\xi_i = \frac{1}{N-1} \sum_{x \in Di}^{(number\_of\_samples)} (x - m_i)(x - m_i)^T$$
$$Sc_j = \sum_{i=1}^{n} (N_i - 1)\xi_i \tag{9}$$

The between-class scatter matrix SB is computed by the following equation:

$$S_B = \sum_{i=1}^{n} N_i (m_i - m)(m_i - m)^T \tag{10}$$

where m is the overall mean, and m_i and N_i are the sample mean and sizes of the respective classes.

3) **Computation of the eigenvectors and eigenvalues for the scatter matrices:** After that, the next step is the computation of the eigenvectors and corresponding eigenvalues for the scatter matrices, as we did for PCA.
4) **Selecting linear discriminant for the new feature subspace:** Following, the sort of the eigenvectors should be applied by decreasing eigenvalues and choosing k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix (where every column represents an eigenvector).
5) **Transforming the samples onto the new subspace:** Next, the selected $d \times k$ eigenvector matrix will be used to transform the samples onto the new subspace dimension.

E. Poisoning Attacks Mitigation

Once applying dimensionality reduction techniques and depicting network slices' updates in a 2D plan, the last step consists of grouping the received updates into several clusters, in order to determine malicious updates/models. To ensure an effective detection of malicious updates, we chose to apply two different clustering algorithms. The first is k-means which is an unsupervised learning algorithm. It considers no labeled update models' data. The second is k-nearest neighbors (KNN), as supervised learning algorithm. It considers labeled data about model updates. Both algorithms aim to divide the received update models, after the first FL rounds, into k clusters that share similarities and are dissimilar to the model updates belonging to another cluster. We note that we leverage the model update of the trust participant as a reference that helps us to make the difference between malicious and trust models.

IV. Performance Evaluation

In this section, we evaluate the performance of our TQFL framework in order to validate it.

A. Experiment Setting

We developed the main components of our framework, in terms of FL-based latency model, DQN-based participants selection model, and clustering and dimensionality reduction-based attack detection model, using TensorFlow Python library and leveraging our EARCD dataset. We evaluate our FL-based model to predict the latency KPI, in terms of Mean Squared Error (MSE) on top of two different weight optimizers (SGD and ADAM) in order to determine the suitable optimizer that improves the prediction accuracy of our both FL-based and attack detection models.

We note that we evaluate our FL-based model in and without the presence of malicious nodes to show the impact of poisoning attacks on the performance of our FL model. The malicious nodes (network slices) generate poisoning attacks by introducing new malicious/incorrect data samples and building their local models on top of such data. For example, the malicious nodes may consider low latency values, even when receiving a high number of attach requests and vice versa. In addition, we trained our DQN model for more than 5000 learning episodes. Once converged, we deployed our DQN-based model at the inter-domain slice manager, which is in charge of selecting a trusted participant at each FL round.

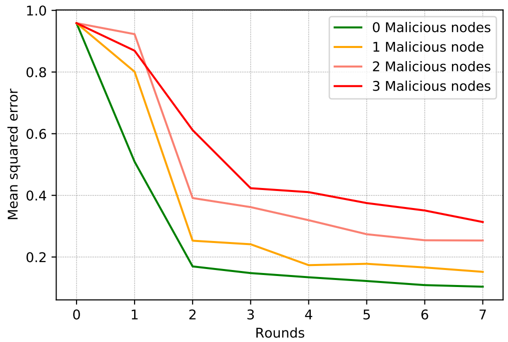


Fig. 4: Mean Squared error of our FL model when using ADAM Optimizer.

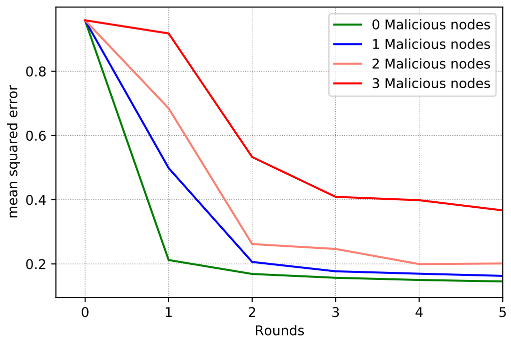


Fig. 5: Mean Squared error of our FL model when using SGD Optimizer.

TABLE II: The parameter settings.

The parameter settings	Value
Federated Learning (FL)	
Number of layers	4
Number of neurons	20
Optimizer 1	SGD
Learning rate	0.0001
Optimizer 2	ADAM
Activation function	ReLU
Loss function	MSE
Reinforcement Learning (DQN)	
Number of layers	2
Number of neurons	24
Optimizer	ADAM
Learning rate	1e-2
episodes	5000
Dimensional Reduction Algorithm (DRA)	
Dimension of LDA/PCA	2D
Clustering Algorithm (CA)	
Number of cluster (k)	1, 2

Moreover, to evaluate our attack detection scheme, after a given number of FL training rounds, the ten network slices' DSMs send their model updates to the inter-domain network slice manager. The latter first selects one network slice as a trusted party before sharing with it the model updates. Table. II gives more details about parameter settings used in our simulation.

B. *Evaluation of Latency prediction in Federated way*

Following, we present the results without malicious nodes for two different optimizer: ADAM and SGD. Fig. 4 depicts the MSE metric of our FL model during several FL rounds, with and without the presence of malicious network slices. On top of the ADAM optimizer, we clearly observe that MSE of our model without malicious nodes is lower than with malicious nodes, around 0.1035681. However, it increases as we add more malicious nodes that will inject more malicious/incorrect data in terms of latency KPI. Therefore, poisoning data attack affects highly not only the performance of our FL model in terms of MSE, but also the model convergence towards an accurate global model. Similarly, Fig. 5 shows the MSE metric of our FL model on top of the SGD optimizer. We also see that the MSE increases as the number of malicious network slices increases as well, while the FL model without malicious nodes outperforms the other models (with malicious nodes), with a MSE of 0.173584. Hence, these results confirm the results of Fig. 4 and show that poisoning attacks can have a negative impact on the performance of our FL model, especially in terms of global model convergence, whatever the used learning optimizer (ADAM or SGD).

C. *Evaluation of Trust Participant Selection*

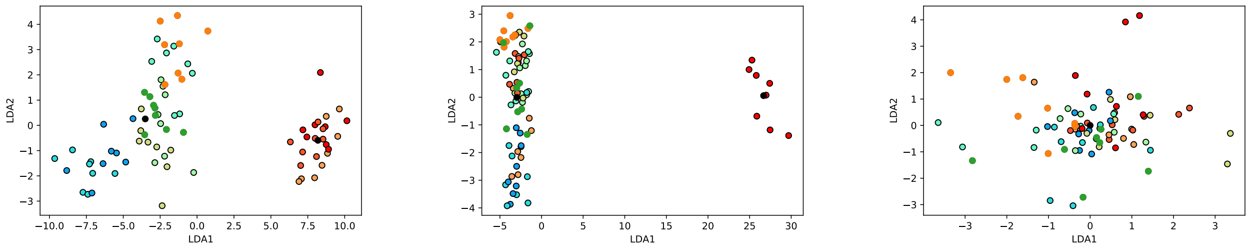
TABLE. III shows the results of the selected DSM based on our DQN-based scheme, for two different time instances $t1$, and $t2$. These results are obtained based on the nodes' reputations ($Rep_i, i \in [1, 10]$). As we observe, when $t = t1$, the $DSM4$ was selected as trust node, since it maximizes the reputation value. Similarly, when $t = t2$, the network slices' DSMs have different reputation values. However, $DSM8$ is selected, because, it presents the highest value. In other words, our DQN-based algorithm enables to select the network slice that maximize the reward. Since, the reputation value is determined based on the reward using equation 2.

D. *Evaluation of Data Poisoning Attack Detection*

After showing the negative effect of the data poisoning attacks on the performance of the global FL model, in this subsection, we evaluate the performance of our combined dimensionality reduction and unsupervised clustering scheme against data poisoning attacks, and on top of the two different optimizers: ADAM and SGD. We note that for data poisoning attacks, the malicious node tries to inject incorrect latency values, e.g., high latency value, even when the node has high resource capacity in terms of memory and computing.

1) ***Combining LDA with k-means:*** Fig. 6 and Fig. 8 depict the detection results when combining LDA and K-means on top of ADAM and SGD optimizers, respectively. We also vary the percentage of malicious network slices' DSM and show the trusted nodes that are selected at each FL round (nodes in green color). As we see, our scheme can clearly detect the malicious nodes, even with only one malicious node. Specifically, the trusted node applies both LDA and K-means, and then all nodes that are in the same cluster with it are considered correct models, while the nodes (models) that are in the other (s) cluster (s) will be considered as malicious. Therefore, our trust participant selection algorithm helps us not only to select a trusted node but also to determine malicious nodes when performing dimensionality reduction and unsupervised clustering. Moreover, determining the trusted cluster of nodes will also help the FL server (IDSM) to select a trusted participant for the next FL round.

2) ***Combining LDA with KNN:*** Fig. 7 and Fig. 9 also show the clustering of local models when applying both LDA and KNN on top of ADAM and SGD optimizers, respectively. Whatever the number of malicious nodes, we also observe that there are always some isolated points that represent the infected models sent by the malicious DSMs. However, for the LDA technique, we see that the isolated models (infected) are identified better with the ADAM optimizer than with the SGD optimizer (Figs. 6 and 7). Hence, the LDA (with KNN or k-means) technique gives better detection on top of the

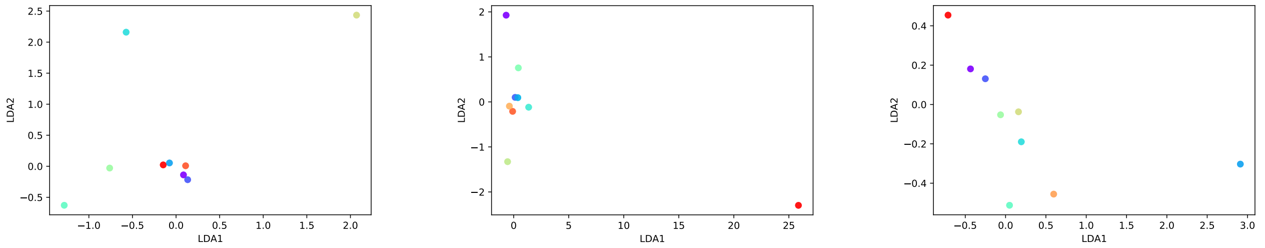


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0.

Fig. 6: LDA + K-means for different number of malicious nodes (ADAM optimizer).

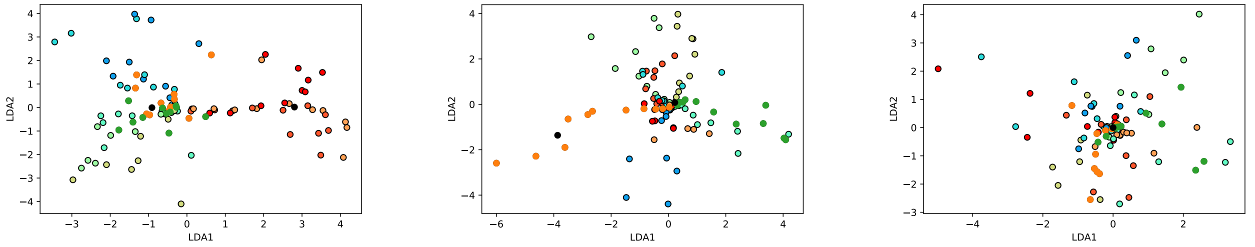


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0.

Fig. 7: LDA + KNN for different number of malicious nodes (ADAM optimizer).

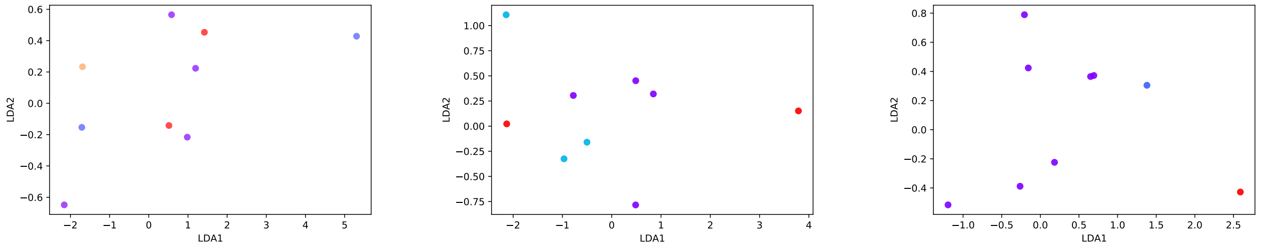


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0.

Fig. 8: LDA + K-means for different number of malicious nodes (SGD optimizer).



a: Nb of malicious nodes = 3.

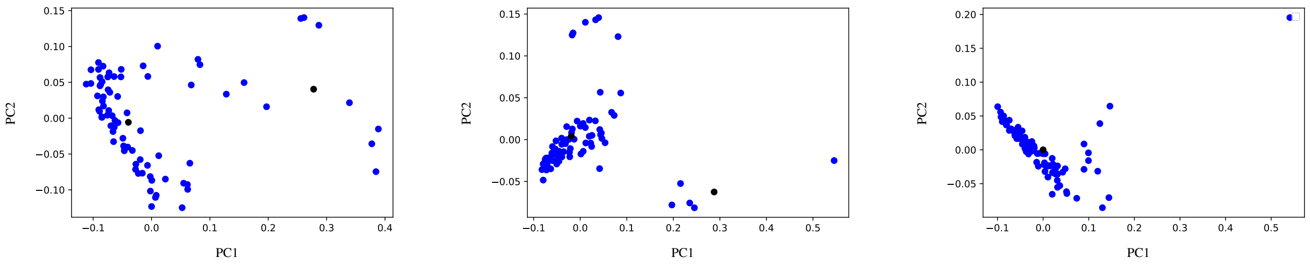
b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0.

Fig. 9: LDA + KNN for different number of malicious nodes (SGD optimizer).

ADAM optimizer. In fact, these last combinations show the clearest clustering (two separate groups) compared to other algorithms.

*3) **Combining PCA with K-means:*** As we did for LDA, we also evaluate the performance of PCA technique when combined with clustering unsupervised algorithms. Figs. 10

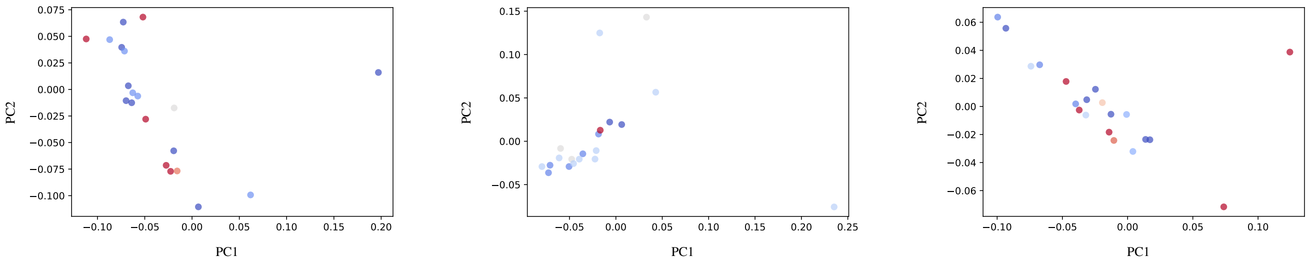


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0

Fig. 10: PCA + k-means for different number of malicious nodes (ADAM optimizer).

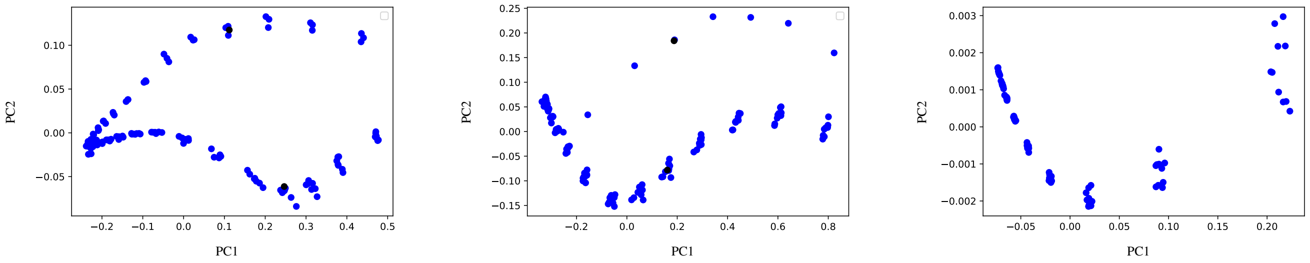


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0

Fig. 11: PCA + KNN for different number of malicious nodes (ADAM optimizer).

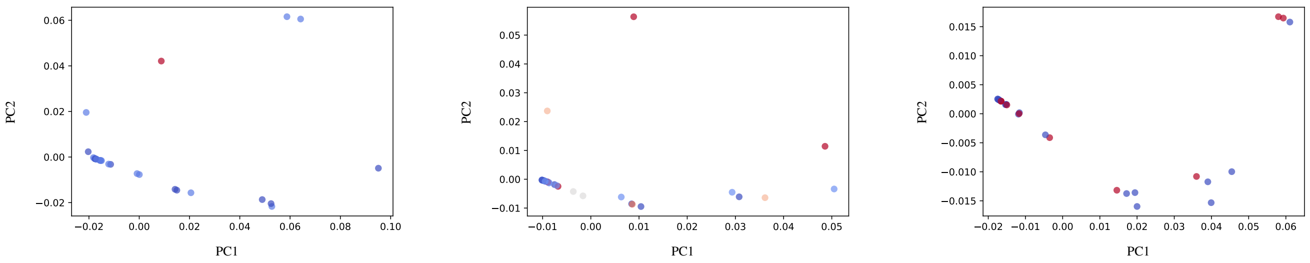


a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0

Fig. 12: PCA + k-means for different number of malicious nodes (SGD optimizer).



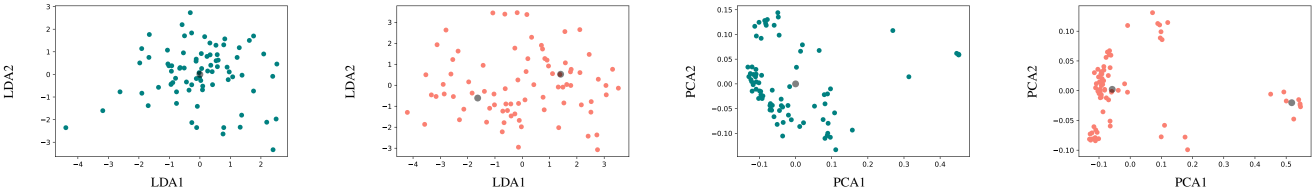
a: Nb of malicious nodes = 3.

b: Nb of malicious nodes = 1.

c: Nb of malicious nodes = 0

Fig. 13: PCA + kNN for different number of malicious nodes (SGD optimizer).

and 12 shows the clustering detection when combining PCA with K-means, on top of the ADAM and SGD optimizers, respectively. We remark that both optimizers succeed in separating and identifying infected models by incorrect data. However, infected models are better identified on top of the SGD optimizer as compared to the ADAM optimizer. Thus,



a) without malicious nodes, (ADAM Optimizer).

b) with malicious nodes, (ADAM Optimizer).

c) without malicious nodes, (SGD Optimizer).

d) with malicious nodes, (SGD Optimizer).

Fig. 14: DA + k-means for different number of malicious nodes.

TABLE III: The results of the selected DSM based on our DQN-based scheme.

t : $t1$									
RepDSM1	RepDSM2	RepDSM3	**RepDSM4**	RepDSM5	RepDSM6	RepDSM7	RepDSM8	RepDSM9	RepDSM10
0.2	0.4	0.3	**0.7**	0.61	0.48	0.37	0.68	0.54	0.62
Selected DSM			✓						
t : $t2$									
RepDSM1	RepDSM2	RepDSM3	RepDSM4	RepDSM5	RepDSM6	RepDSM7	**RepDSM8**	RepDSM9	RepDSM10
0.1	0.22	0.15	0.74	0.58	0.52	0.31	**0.8**	0.60	0.71
Selected DSM							✓		

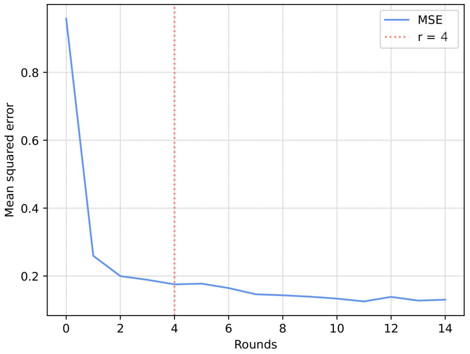


Fig. 15: Mean Squared error of the FL global model (ADAM optimizer).

PCA with K-means gives better performance in detecting infected models on top of the SGD optimizer. Indeed, this last combination show the clearest clustering (two separate groups) compared to other algorithms.

4) ***Combining PCA with KNN:*** Similarly, Figs. 11 and 13 depict the detection when combining PCA with KNN on top of the ADAM and SGD optimizers, respectively. As in Figs. 10 and 12, PCA with KNN on top of both optimizers clearly separates correct local models from infected ones and thus enables to detect/identify malicious DSMs. We also see that infected models are better identified when leveraging the SGD optimizer than the ADAM optimizer. Therefore, the PCA technique with either K-means or KNN gives better detection of malicious models on top of the SGD optimizer, which is confirmed in Figs. 10, 11, 12, 13. In fact, these last combinations show the clearest clustering (two separate groups) compared to other algorithms.

5) ***Impact of malicious nodes detection on the FL accuracy:*** In Fig. 4 and 5, we showed that poisoning attack affects highly not only the performance of our FL model in terms of MSE, but also the model convergence towards an accurate global model. Moreover, the observation confirms that even a single malicious node can influence the global model accuracy, as well as open the possibility of conducting two types of attacks. The first one is a Byzantine attack, in which the malicious node aims to prevent the FL global model from converging. Whereas the second is the poisoning attack in which the attacker intends to make the FL global model misinterpreting some of the inputs in its future use. Fig. 15 depicts the MSE metric of our global FL model when applying our combined dimensionality reduction and clustering scheme to detect and remove infected models. Noting that we apply our scheme after a given number of FL rounds $r = 4$, i.e., after collecting some updates from the network slices' DSMs. We clearly observe that our scheme improves MSE of the global FL model, which decreases as we increase the number of FL rounds, even when injecting some infected models from the fourth round. Therefore, our detection scheme enables not only to identify malicious DSMs, but also to improve the accuracy of the global FL model.

E. Evaluation of Model Poisoning Attack Detection

In this subsection, we evaluate our scheme against model poisoning attacks, where malicious DSMs try to send incorrect learning models, which are generated randomly for instance. Fig. 14 shows the results when combining LDA and K-means on top of the ADAM optimizer. We remark that LDA with K-means does not clearly succeed to identify the malicious models Fig. 14-b. However, when combining PCA and K-means on top of the SGD optimizer, the malicious models are better detected, as depicted in Fig. 14-d. Therefore, we can deduce that model poisoning attacks are better detected and identified, when using PCA with K-means techniques, as compared to LDA and K-means combination. In general, we can deduce that our scheme succeeds in providing stable performance in dealing with both data and model poisoning attacks that can target federated learning-based models. In particular, combining dimensionality reduction and unsupervised clustering learning helps us to not only detect/identify infected learning models but also to improve the global accuracy of the FL model.

V. Conclusion

In this work, we designed a novel secure federated learning framework, which leverages reinforcement deep learning, dimensionality reduction, and clustering unsupervised learning to deal with both data and model poisoning attacks that can target federated learning-based models in 5G and beyond networks. First of all, we generate a realistic dataset related to network slicing KPIs in order to build a federated learning model for latency KPI prediction. We then generate realistic data and model poisoning attacks on top of our federated learning-based model. To deal with them, a dynamic selection algorithm of a trusted node is first proposed, leveraging reinforcement deep learning. The latter is in charge of detecting and identifying malicious learning models and hence network slices. The numerical results show the efficiency of our framework in dealing with model/data poisoning attacks, while preventing the Byzantine attacks, and thus mitigating such attacks and ensuring stable performances of the federated learning model.

References

[1] Shah Zeb, Aamir Mahmood, Syed Ali Hassan, et al, Industrial digital twins at the nexus of NextG wireless networks and computational intelligence, Journal of Network and Computer Applications, 2022.

[2] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz , et al. Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning, the IEEE Symposium on Security Privacy, 2022.

[3] *Etsi*. Zero touch network Service Management (ZSM) [Online]. Available: https://www.etsi.org/technologies/zero-touch-network-service-management, (10, 01, 2022).

[4] Slawomir Kuklinski and Lechoslaw Tomaszewski, Key Performance Indicators for 5G network slicing, IEEE Conference on Network Softwarization (NetSoft), pp. 464–471, June 2019.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. CoRR,2018.

[6] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In NeurIPS, pages 119–129, 2017.

[7] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh, Mitigating sybils in federated learning poisoning. CoRR, 02 Sep 2018.

[8] Gu, T., Dolan-Gavitt, B., Garg, S.: BadNets: identifying vulnerabilities in the machine learning model supply chain. CoRR, 2017.

[9] Yi Liu, Jialiang Peng, Jiawen Kang, et al. A Secure Federated Learning Framework for 5G Networks, IEEE WIRELESS COMMUNICATIONS MAGAZINE, March 2020.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient learning of deep networks from decentralized data, in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, USA, 2017.

[11] Zakaria Jaadi, A Step-by-Step Explanation of Principal Component Analysis (PCA). Available: https://builtin.com/data-science/step-step-explanation-principal-component-analysis, (14, 10, 2021).

[12] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, S. Yu. PDGAN: a novel poisoning defense method in federated learning using generative adversarial network Proceedings of ICA3PP, Springer, 2019.

[13] Close J. Steinhardt, et al. Certified defenses for data poisoning attacks Proceedings of NeurIPS, pp. 3518-3530, 2017.

[14] Mahesh Subedar, Nilesh A. Ahuja, Ranganath Krishnan, et al, Deep Probabilistic Models to Detect Data Poisoning Attacks CoRR, 2019.

[15] Matthew Jagielski, Alina Oprea, Battista Biggio , et al. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning, 2018 IEEE Symposium on Security and Privacy, 2018.

[16] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, et al. DeepSight: Mitigating Backdoor Attacks in Federated Learning Through Deep Model Inspection, CoRR, 2022.

[17] Mustafa Safa Ozdayi, Murat Kantarcioglu, Yulia R. Gel, Defending against Backdoors in Federated Learning with Robust Learning Rate, CoRR, 2020.