University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Degree in Cybersecurity

# Distributed Intrusion Detection System for Heterogeneous Data in Industrial Control Systems

*Master Thesis*

*Supervisor and Co-supervisor*
Prof. Mauro Conti
Dr. Federico Turrin

*Master Candidate*
Francesco Freda
2020383

Academic year 2022-2023

*It takes 20 years to build a reputation and few minutes of cyber-incident to ruin it.*

— *Stéphane Nappo*

Dedicated to my family.

# Abstract

In recent decades, Industrial Control Systems (ICS) have been affected by a wide range of cyberattacks that had a huge impact on the real world and people's safety. These cyberattacks can compromise Critical Infrastructure (CI) in all countries and all fields, like agriculture, water supply, and transportation systems. For this reason, CI protection became an important concept related to the defense against cyberattacks. In the last few years, one of the main technique to protect these systems are Intrusion Detection Systems (IDSs), which allow to efficiently detect potential anomalies and cyberattacks, while being easy to deploy on existing networks.

Nowadays, the techniques implemented inside industrial IDS to achieve the best performance in the detection of cyber anomalies are based on Machine Learning (ML) and Deep Learning (DL). However, proposed approaches mostly include black box methods, with a lack of generalization, and explainability and they require big computing power. Furthermore, these complex techniques may be specialized in the detection of well-defined cyberattacks, leaving the door open for other zero-days attacks. Some recent results show how simpler approaches based on static rules are comparable, sometimes better, than more complex algorithms.

In this thesis, we propose a Distributed Intrusion Detection System (DIDS) using transparent and straightforward detectors. The detector is distributed because it includes all the heterogeneous types of data that characterize the ICS: physical and network. Network detectors are applied at different points inside the ICS to monitor the traffic, while the physical detector leverages the information from the Supervisory Control and Data Acquisition (SCADA) devices as input to find anomalies in the field devices processes. Indeed, most of the modern industrial IDS focus only on a single type of data, with the consequences of missing meaningful information. Moreover, our DIDS is compatible with the majority of industrial protocols.

We tested the proposed methodology on two digital twin scenarios, each one including six different cyber attacks. The distributed approach demonstrates effectiveness in correctly identifying all attacks, as opposed to an approach that considers only one source of information. As a matter of fact, during our experiments, the distributed detector was able to identify six out of six attacks with zero false detections in both scenarios.

Finally, after having identified the anomalies, we propose a method based on Random Forest which allows us to assign the type of attack to each anomaly. This approach obtains 83% Macro-averaged Precision, 85% Macro-averaged Recall, and 84% Macro-averaged F1 score.

# Acknowledgments

*First of all, I would like to express my gratitude to Prof. Mauro Conti and Dr. Federico Turrin, respectively supervisor and co-supervisor of my thesis, for helping and supporting me in drafting this important document.*

*Lastly, I would like to warmly thank my parents and friends for encouraging and advising me during my studies.*

*Padova, September 2023* Francesco Freda

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

In today's society, critical activities are carried out using sophisticated infrastructures. The Industrial Control System (ICS) is one of these, which is a class of automation systems used in manufacturing and industrial facilities to provide control and monitoring functions. Different types of ICS exist, such as Supervisory Control and Data Acquisition (SCADA), Distributed Control System (DCS), or even Programmable Logic Controller (PLC).

These infrastructures are classified as *critical assets* to protect services not only in the physical but also in the digital world. At the national and EU levels, Critical Infrastructure (CI) protection has become a high priority due to numerous threats.

There are two major types of ICSs: the Operational Technology (OT) part, which includes hardware and software used for monitoring and managing industrial equipment (such as PLCs, sensors, and actuators), and the traditional Information Technology (IT). Their objective is to effectively control the underlying physical processes.

Due to the so-called IT/OT Convergence [1], the two infrastructures have been interconnected to facilitate the digitization of processes. The connection of ICSs to the internet and the incorporation of protocols such as TCP/IP has expanded the *attack surface* and made CI vulnerable to a wider range of attacks, leading to successful and unsuccessful attempts to compromise the behavior of ICSs. However, legacy ICS devices were usually not designed to implement adequate network security and are rarely replaced due to high costs and long device lifetimes. Consequently, ICSs are increasingly targeted by cyberattacks with potentially severe damage. To alleviate this situation, security mechanisms must be retrofitted for Internet-connected ICS devices.

ICS is considered an instance of Cyber-Physical Systems (CPS). This latter is a computer system in which a mechanism is controlled or monitored by computer-based algorithms. In this field, the classical CIA triad (Confidentiality, Integrity, Availability) is reversed in order of importance: *Availability, Integrity, and Confidentiality.* Indeed, reliability becomes the most important goal since, differently from IT systems, for an ICS it can guarantee human safety and fault tolerance, and so availability does too. For instance, in a nuclear plant environment, data availability, like the temperature of the core, is more important than its confidentiality.

Recent examples of attacks include the attempted poisoning of a Florida city's water supply by increasing its sodium hydroxide concentration [2]. Given the potential harms of attacks on ICS, detecting and preventing them promptly is critical. For this reason, the research has seen a rising interest in detecting intrusions into industrial networks. Such an Industrial Intrusion Detection System (IIDS) passively monitors processes to alert about anomalous behavior before any real damage can occur and promises to provide a non-intrusive, retrofittable, and easily deployable *security solution*. In contrast to traditional IDSs known from office or data center environments, IIDSs have the unique advantage that they can leverage the *predictability* and *repetitiveness* of ICSs to identify even advanced and stealthy attacks.

A promising approach for IIDS is the application of Machine Learning (ML) algorithms. They can be trained on historic ICS data, thereby learning properties of the physical system and the attacks that can be performed on it. Hence, ML algorithms supersede the manual crafting of system models in anomaly detection and signatures in signature-based detection. Furthermore, the ability of ML to generalize and abstract patterns allows even operating on new unseen data.

However, classifying ML-based IDSs as a signature or anomaly-based, i.e., determining whether an IDS learns normal behavior, attack signatures, or both, is non-trivial due to the *not-transparency of the learning process* within ML. The powerful underlying IIDS methodologies yield promising detection performances, however, at the cost of *complexity*, requiring *resource-intensive operations* and *hindering generalizability* [3]. Furthermore, the alarms raised by, e.g., artificial neural networks, are often *not explainable*, making it challenging to derive concrete actions for mitigating attacks [4].

Therefore, we pose two questions:

1. Is it possible to obtain a *transparent, lightweight, and reliable* system that is comparable in terms of performance to black box approaches?
2. Does a *distributed* IIDS monitoring both the network and physical processes work better than a sectorized one installed in a single point between these two?

To answer these questions, we developed a Distributed Intrusion Detection System (DIDS) composed of a network IDS and a physical IDS that both leverage *simple* metrics to detect attacks (e.g., maximum and minimum sensor values, whether a sensor/actuator does not change its value for a shorter or longer time, and so on). Indeed, with this simple approach, we avoid many of the drawbacks of complex solutions and we show that they are sufficient to detect most of the attacks employing *lightweight computational* operations. With the distributed approach it is possible to combine the multiple opinions from the network and physical detectors, in this way allowing a *wide overview* of the ICS.

Moreover, we implemented another detector to perform attack classification. This latter uses Random Forest (RF) predictor. We tested each detector on two ICS scenarios reproduced by a state-of-the-art Digital Twin (DT) which allows us to achieve a *high degree of fidelity* in the experiments. Finally, we compared the DIDS with Bidirectional Long Short Term Memory (BLSTM) model, since, by what Wolsing et al. [5] report in their paper, BLSTM is one of the detectors with the best performance on the dataset they used.

## 1.1 Contributions

The main contributions of this thesis are:

- The development of a DIDS that leverages multiple IIDSs over a simulated ICS network and the information from the SCADA devices. Each IIDS conducts a simple detection task, is “transparent” and anomaly-based, to guarantee a fast detection time, an explainable detection process, and the possible detection of zero-days attacks respectively. The DIDS takes in input abstracted ICS network packets using Industrial Protocol Abstraction Layer (IPAL) [31] to deal with different types of protocols and, more in general, different types of ICS infrastructures. These characteristics solve parts of the problems related to the IIDS: high complexity, not transparency, and sometimes limited detection to a specific class of attacks and/or ICS.

- Testing of the DIDS on the data produced by two DT scenarios implemented by Digital HydrAuLic SIMulator (DHALSIM) with high-fidelity and reproducibility features.

- Evaluation and comparison between the developed distributed detector and another state-of-the-art not-distributed supervised approach, called Bidirectional Long Short Term Memory (BLSTM). The performances are measured in terms of correctly detected attacks and correctly ignored normal executions. The results show that DIDS has excellent outcomes on the examined attacks and zero false detected normal executions compared to BLSTM detector.

- The development and performance evaluation of a method to classify the attack(s) in action on the ICS network. This approach uses RF to fingerprint the traffic shape and identify occurring attacks. The identification of the ongoing attack may be useful in performing mitigation strategies against the attacker(s).

## 1.2 Thesis Organization

This thesis is organized as follows. In Chapter 2, an overview about ICS security, Inter-Arrival Mean (IAM) and Inter-Arrival Range (IAR) is given. Furthermore, the ML and Random Forest (RF) that are used in this work are also described. The related works composed by different Intrusion Detection Systems applied to the ICS field are presented in Chapter 3. In Chapter 4, an overview of the general architecture of the implemented Distributed Intrusion Detection System (DIDS) is presented, while the specific components, jointly with the testbed and launched cyber-attacks, are going to be discussed in Chapter 5. In Chapter 6 we present the two datasets used to conduct the experiments, the experimental setup of the DIDS and the classifier, and the data analysis together with the results of the performed experiments. Chapter 7 concludes the thesis and proposes future works.

# Chapter 2

# Background Knowledge

## 2.1 Industrial System Security

For the purpose of this work, the general understanding of what an Industrial Control System (ICS) is, what are its basic constituents and the possible cyber-attacks that can be performed on it, is the base needed to implement an IIDS.

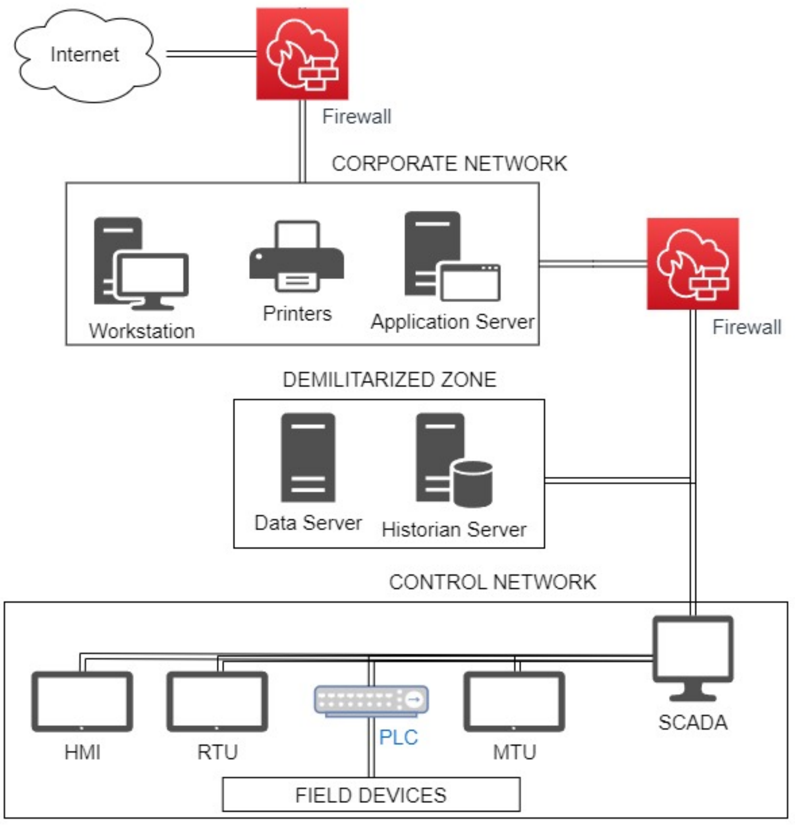

**Figure 2.1:** ICS general infrascture.

The reference architecture of the ICS, visible in Figure 2.1, is divided into logical segments with similar functions or similar requirements [6]:

- Enterprise Zone, or IT network, includes the traditional IT devices and systems such as the logistic business systems and the enterprise network.
- Demilitarized Zone (DMZ) controls the exchange of data between the Control Zone and the Enterprise Zone, managing the connection between the IT and the OT networks in a secure way.
- Control Zone, sometimes also referred to as OT network, includes systems and equipment for monitoring, controlling, and maintaining the automated operation of the logistic and physical processes.
- Safety Zone includes devices and systems for managing ICS security by monitoring for anomalies and avoiding dangerous failures.

**ICS Devices** An ICS include heterogeneous hardware and software components such as sensors, actuators, physical systems and processes being controlled or monitored, computational nodes, communication protocols, SCADA systems, and controllers. These systems are composed of standard network traffic over TCP/IP stack and data from physical processes and low-level components.

Control can be fully automated or may include a human in the loop that interacts via a Human Machine Interface (HMI). ICSs are widespread in modern industries (e.g., gas pipeline, water treatments) and CIs (e.g., power plant and railway) [6].

Industrial protocols are specifically designed to deal with real-time constraints and legacy devices in an air-gap environment. Many protocols do not implement any encryption or authentication mechanism due to these constraints, opening several vulnerabilities surfaces.

ICS is composed of a wide range of heterogeneous devices and components with a specific role in the system. Between the main classic ones, i.e., routers and switches, other specific devices may be used, that can be divided into two main groups:

- Field components, such as sensors, actuators, motor drives and gauges.
- Control system components, such as PLCs, Remote Terminal Units (RTUs), Intelligent Electronic Devices (IEDs), etc.

PLC is a specialized industrial computer used to automate functions within manufacturing facilities. It reads input signals from sensors, executes programmed instructions using these inputs and orders from supervisory controllers, and creates output signals that may change switch settings or move actuators. May be specialized for specific industrial uses with multiple specific inputs and outputs. Processing overhead and delay in the execution of the PLC functioning may impair a whole production process. These components are generally connected to the local network to communicate with supervisory processes.

SCADA devices are used to monitor and control centralized data acquired from different field sites. Furthermore, they manage the communication between the various devices and represent the remote connection point for the remote operators with the OT network. Over the year, SCADA systems protocols moved from proprietary standards towards open international standards, resulting in attackers knowing precisely the protocols. That is why there is a gain of interest in reinforcing industrial control systems security.

The HMI is a software installed on dedicated flat panel screens that permit operators to check and monitor the automation processes such as process values, alarms, and data trends. An operator can use the HMI to send manual commands to controllers, for instance, to change some values in the production chain.

A Data Historian is a software application used to collect real-time data from the processes and aggregate them into a database for analysis.

To sum up, controllers such as PLC are mainly used to interact with the Field Devices that can instead directly operate on the processes. HMI and Data Historian are used to control and manage the system data. Instead, SCADAs are used to set up all the connections between different components.

**ICS Protocols** Several new protocols have been developed to support the specific requirements of the OT environment, like fault tolerance and reliability. The majority of these protocols were designed to operate in an air-gapped environment. Therefore originally, less importance was given to the security aspects with respect to the real-time constraint. The main industrial protocols are Modbus, DNP3, S7Comm, PROFINET, IEC 60870, Common Industrial Protocol (CIP) and Open Platform Communications (OPC).

The communication between the master device and field devices relies on SCADA-specific protocols built upon different communication technologies like serial communication and TCP/IP. Between all the existing protocols, we can cite:

- CIP is supported by ODVA. CIP encompasses a comprehensive suite of messages and services for the collection of manufacturing automation applications – control, safety, synchronization, motion, configuration and information. It allows users to integrate these manufacturing applications with enterprise-level Ethernet networks and the Internet. It is supported by hundreds of vendors around the world and is media-independent. CIP provides a unified communication architecture throughout the manufacturing enterprise. It is used in EtherNet/IP, DeviceNet, CompoNet, and ControlNet.
- There are several Modbus protocols: Modbus RTU and Modbus ASCII are used in serial communication, often RS232. Modbus TCP is used for TCP communication. The Modbus protocol uses a synchronous request-response communication mode. The SCADA master initiates requests/commands stating the request type and starting address. The field device then responds by sending the requested data.
- IEC 60870 is a standardized application layer protocol built upon TCP/IP stack. The protocol allows balanced/unbalanced communications. In the unbalanced mode, only the master can initiate communications to field devices. On the contrary, both the master and field devices can initiate communications in the balanced mode. This protocol allows both synchronous and asynchronous messages. The field devices can send Spontaneous and Periodic messages from predefined addresses, called Information Object Address (IOA).

**ICS Cyber-attacks** In a CPS scenario there are two possible attack vector surfaces on the system:

· Network-based attacks, targeting packets, protocols or routing policies. The most common examples are Reconnaissance Attack, Man-In-The-Middle (MITM) Attack, Injection Attack, Replay Attack, and Denial of Service (DoS).

· Physical-based attacks, aimed at corrupting the physical process of the devices. To achieve these attacks, the attacker could have previously obtained access to the system with one or more of the network attacks previously described. An example is Device Manumission which physically tamper the field device to compromise the data recorded to induce wrong measurements in the system.

Sometimes, these two attack categories' goals may also converge or combine to reach a specific target.

**ICS Digital Twin** A Digital Twin (DT) is a virtual model designed to accurately reflect a physical object. As explained by Alcaraz et al. [39], the purpose of a DT is to use specification-based techniques, mathematical models, and application programming interfaces to represent physical assets through digital assets. In our thesis, the considered assets are the ones inside an ICS. The main aim is to anticipate errors, variations, and relevant deviations that may change a system's natural behavior. All the DT components run on servers and/or virtualized resources. In turn, these servers are connected to the physical world in order to interact with real-world components.

A common subdivision of a DT is provided by Grieves [51] and Alcaraz et al. [39]:

· Physical space: comprises the real-world ICSs composed of sensors, actuators, and controllers.

· Digital space: simulates physical assets using digital assets capable of representing states, conditions, and configurations, and making decisions regarding those assets.

· Communication space: creates the connection between the digital and the physical spaces. The DT is able to interfere in the production operations using information flows and processes.

Data from physical assets is processed by digital assets, which create new useful information that may be sent back to physical assets. As a result, the virtual and physical worlds are connected thanks to a *digital thread*.

**Intrusion Detection System** The Intrusion Detection System (IDS)s aim is to identify malware, malicious access, or any kind of attack to defend internal networks. They represent one major research problem in cyber security and as there are several risks concerning networks there are different systems built to secure an environment from external attacks. The IDSs provide a wall of defense as they can be used to detect and analyze types of malicious network communications and computer systems usage, whereas conventional firewalls cannot perform this task.

The IDS specialized in the detection of intrusions inside an ICS are called Industrial Intrusion Detection System (IIDS). In contrast to traditional IDSs known from office or data center environments, IIDSs have the unique advantage that they can leverage the predictability and repetitiveness of ICSs to identify even advanced and stealthy attacks. IIDS can be classified as:

- Network Intrusion Detection System (NIDS) analyzes incoming network traffic.
- Physics-based Intrusion Detection System (PIDS), in OT world, monitors abnormal behaviors at the process level controlling the reported values of different field devices, like sensors and actuators.
- Distributed Intrusion Detection System (DIDS) combines some of the previous types of IDSs to have detectors at multiple points and to improve the performance.

It is also possible to classify IDS by detection approach: signature-based IDSs, which detect attacks using pre-configured signatures, e.g., a specific sequence of network packets, and anomaly-based IDSs that attempt to model the expected behavior of a system and consider deviations as potential intrusion, e.g., a control parameter outside physical bounds. Thus, while signature-based IDSs can only identify known attacks, anomaly-based IDSs promise to also detect novel attacks.

Hybrid IDS is a technique that combines signature-based and anomaly-based IDSs to resolve the disadvantages of the two legacy IDSs. Indeed, signature-based IDS performance is inaccurate when applied to zero-day attacks. Furthermore, a small modification to an attack would change its signature, thus making it difficult to identify an attack by a signature-based IDS. In the case of anomaly-based IDSs, a ML algorithm models the normal behavior of the network and identifies everything outside of the learned model as an anomaly.

## 2.2 Machine Learning

Since this research makes use of RF, a supervised ML algorithm, it is required to have a basic knowledge of what ML is and to discuss some of its basic elements.

Machine Learning is a branch of Artificial Intelligence (AI) with the aim of creating systems that learn and improve performance based on the data they use. It is used for solving problems by helping machines 'discover' their 'own' algorithms, without the help of any human-developed algorithms. The development of these kind of algorithms by human programmers would be cost-prohibitive.

The main goal of ML is, given a collection of samples, called training data, to be able to make predictions about novel, but incomplete, samples. Based on the learning paradigms, ML algorithms can be classified into different categories:

- Supervised Learning uses labeled data to train the predictors. In labeled data, the output is given with the sample. The model just needs to map the inputs to the respective outputs.
- Unsupervised Learning uses unlabeled data to train the predictors. Unlabeled data doesn't have a fixed output variable. The model learns from the data, discovers the patterns and features in the data, and returns the output.
- Reinforcement Learning trains a machine to take suitable actions and maximize its rewards in a particular situation. It uses an agent and an environment to

produce actions and rewards. The agent has a start and an end state. However, there might be different paths to reaching the end state, like a maze. In this learning technique, there is no predefined target variable.

To establish how good our ML is for a certain task we use the loss function. There are different loss functions based on the task we want the ML model to conduct. To compute the loss function we would go over each training example in our dataset, compute the output $y$ for that sample, and then compute the specific loss. If this loss is big, then our network doesn't perform very well.

We speak about overfitting when a predictor has excellent performance on the training set, but has very poor performance on the true "world". Intuitively, overfitting occurs when our model fits the training data "too well".

As stated above, in general, the best performing and state-of-art models in the ML and Deep Learning (DL) fields are highly resource-consuming, complex, and black-box. For the purpose of this work, the research and use of light and explainable models are fundamentals to efficiently deploy them on cheap and low power consumption devices and have a better understanding of why the detectors found an ongoing attack, respectively.

### 2.2.1 SMOTE

Imbalanced classification involves developing predictive models on classification datasets that have a severe class imbalance, that is, a lot of samples for some categories, and very few samples for some others categories. Most ML techniques will have poor performance on the minority class, although typically its performance on the minority class is important. That is our case when we are going to deal with the Attack Classification task: we have a lot of samples related to normal network traffic, and not so many network traffic samples related to attacks.

Synthetic Minority Over-sampling TEchnique (SMOTE) is a type of data augmentation for the minority class that synthesizes new examples from the existing ones described for the first time by Chawla et al. [38]. This is a type of data augmentation for tabular data and can be very effective. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space, and drawing a new sample at a point along that line [37].

Specifically, a random example from the minority class is first chosen. Then, $k$ of the nearest neighbors for that example are found. A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space. This procedure can be used to create as many synthetic examples for the minority class as are required.

### 2.2.2 Decision Tree

Digital Twin (DT) is a non-parametric supervised learning tree-like model that is mostly used for classification and regression problems. It is a predictor that predicts the label associated with an instance $x$ by traveling from a root node of a tree to a leaf. At each node on the root-to-leaf path, the successor child is chosen on the basis of a splitting of the input space. Usually, the splitting is based on one of the features of $x$ or on a predefined set of splitting rules. A leaf contains a specific label.

A popular splitting rule at internal nodes of the tree is based on thresholding the value of a single feature. In such cases, we can think of a Decision Tree as a splitting of the instance space into cells, where each leaf of the tree corresponds to one cell. So, a tree with $k$ leaves can shatter a set of $k$ instances.

To avoid overfitting in Decision Tree, we aim at learning a decision tree that on one hand fits the data well while on the other hand is not too large. So, we should prefer smaller trees over larger trees.

Decision Tree learning algorithms are based on heuristics such as a greedy approach, where the tree is constructed gradually, and locally optimal decisions are made at the construction of each node. Such algorithms cannot guarantee to return the globally optimal Decision Tree but tend to work reasonably well in practice.

A general framework for growing a Decision Tree is as follows. We start with a tree with a single leaf (the root) and assign this leaf a label according to a majority vote among all labels over the training set. We now perform a series of iterations. On each iteration, we examine the effect of splitting a single leaf. We define some "gain" measure that quantifies the improvement due to this split. Then, among all possible splits, we either choose the one that maximizes the gain and perform it, or choose not to split the leaf at all.

The training procedure of a Decision Tree uses a gain function which given a training set and feature, evaluates the gain of a split of the tree according to the feature. There are several gain measures:

· The simplest definition of gain is the decrease in training error.

· Another popular gain measure is the Information Gain or Entropy, that is, the difference between the entropy of the label before and after the split: $-\sum_i p_i^2 \cdot \log_2(p_i)$, where $p_i$ is the probability of class $i$. The Entropy depicts the disorder of the features with the target. The focus is on purity and impurity in a node.

· The Gini Index measures the probability of a random instance being misclassified when chosen randomly: $1-\sum_i p_i^2$, where $p_i$ is the probability of class $i$. The Gini Index varies between 0 and 1, where 0 represents the purity of the classification and 1 denotes the random distribution of elements among various classes. A Gini Index of 0.5 shows that there is an equal distribution of elements across some classes.

### 2.2.3 Deep Learning

Deep Learning (DL) is the branch of ML which is based on Artificial Neural Networks (ANN) architecture. An ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

Deep feedforward networks are the quintessential DL models. The goal of a feedforward network is to approximate some function $f^*$ [46]. A feedforward network defines a mapping $y = f(x, \theta)$ and learns the values of the parameters $\theta$ that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from $x$, through the intermediate computations used to define $f$, and finally to the output $y$. There are no feedback connections in which the outputs of the model are feedback into itself. When feedforward neural networks are extended to include feedback connections, they are called Recurrent Neural Networks (RNN), and we are going to present it in subsection 2.2.4.

ANNs are built on the principles of the structure and operation of human neurons. ANN's input layer, which is the first layer, receives input from external sources and passes it on to the hidden layer, which is the second layer. Each neuron in the hidden layer gets information from the neurons in the previous layer, computes the weighted total, and then transfers it to the neurons in the next layer. These connections are weighted, which means that the impacts of the inputs from the preceding layer are more or less optimized by giving each input a distinct weight. These weights are then adjusted during the training process to enhance the performance of the model.

The whole ANN is composed of artificial neurons, also known as units, which are arranged in a series of layers. The complexities of neural networks will depend on the complexities of the underlying patterns in the dataset whether a layer has a dozen units or millions of units. Commonly, ANN has an input layer, an output layer as well as hidden layers.

Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network. Then, after passing through one or more hidden layers, this data is transformed into valuable data for the output layer. Finally, the output layer provides an output in the form of an ANN's response to the data that comes in.

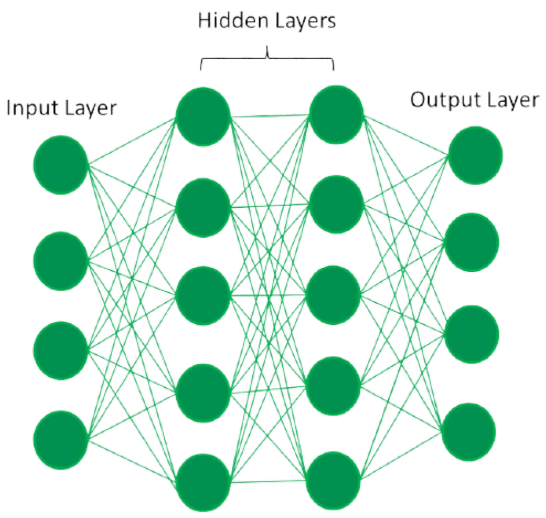

**Figure 2.2:** Fully Connected ANN [47].

Units are linked to one another from one layer to another in the bulk of Neural Networks. Each of these links has weights that control how much one unit influences another. The Neural Network learns more and more about the data as it moves from one unit to another, ultimately producing an output from the output layer. The Figure 2.2 shows a fully connected ANN example.

In the process of training an ANN, we want to start with a bad-performing neural network and wind up with a network with high accuracy. In terms of loss function, we want our loss function to be much lower at the end of training. Improving the network is possible because we can change its function by adjusting weights. The problem of training is equivalent to the problem of minimizing the loss function.

One of the simplest algorithms that optimize loss functions is called Stochastic Gradient Descent. All the algorithms that optimize loss functions in neural networks are based on backpropagation. Backpropagation is a gradient-based algorithm that performs feedforward and backward passes to adjust a neural network model's parameters, aiming to minimize the loss function called Mean Squared Error (MSE). Gradient-based means that it is not only using the information provided by the function but also by its gradient. The MSE is equal to:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{2.1}$$

Where $Y_i$ and $\hat{Y}_i$ are the correct label and predicted label related to the sample $X_i$, respectively.

### 2.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a family of neural networks for processing sequential data, that is, a sequence of values $x_1, ..., x_n$. RNN can scale to much longer sequences than would be practical for networks without sequence-based specialization. Most RNNs can also process sequences of variable length.

Parameter sharing makes it possible to extend and apply the model to examples of different lengths and generalize across them. Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence.

A related idea is the use of convolution across a 1-D temporal sequence. This convolutional approach is the basis for time-delay neural networks. The convolution operation allows a network to share parameters across time but is shallow. The output of convolution is a sequence where each member of the output is a function of a small number of neighboring members of the input. The idea of parameter sharing manifests in the application of the same convolution kernel at each time step. Recurrent networks share parameters in a different way. Each member of the output is a function of the previous members of the output. Each member of the output is produced using the same update rule applied to the previous outputs. This recurrent formulation results in the sharing of parameters through a very deep computational graph.

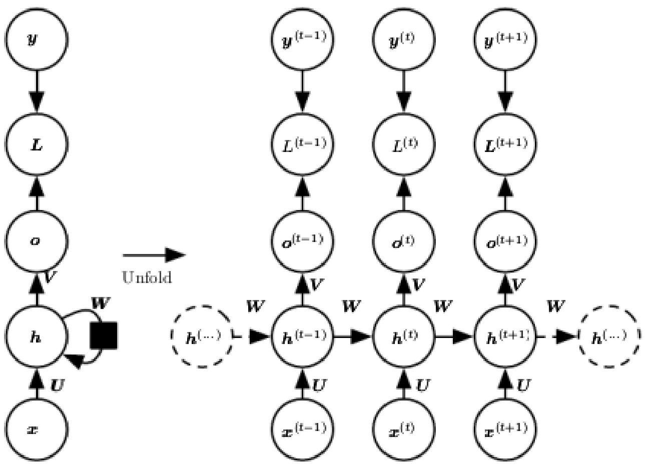

**Figure 2.3:** RNN that produce an output at each time step and have recurrent connections between hidden units [46].

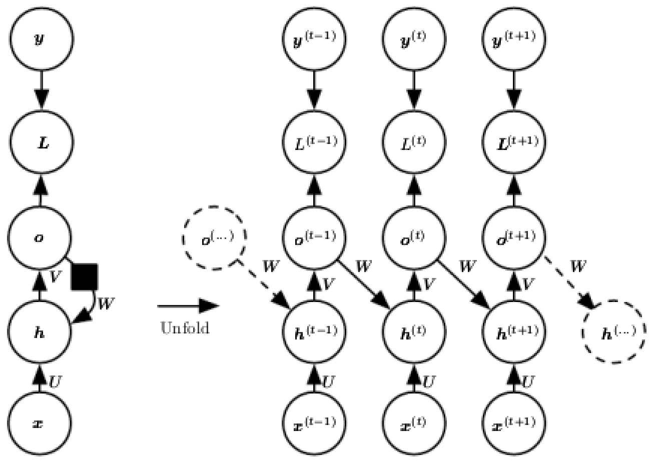

**Figure 2.4:** RNN that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step [46].

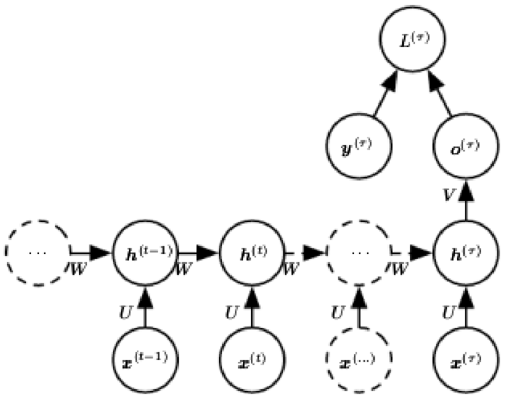

**Figure 2.5:** RNN with recurrent connections between hidden units, that read an entire sequence and then produce a single output [46].

There are a wide variety of RNN. Here we give some examples:

· Recurrent networks that produce an output at each time step and have recurrent connections between hidden units, illustrated in Figure 2.3.

· Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step, as shown in Figure 2.4.

· Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output, visible in Figure 2.5.

The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:

1. From the input to the hidden state.
2. From the previous hidden state to the next hidden state.
3. From the hidden state to the output.

It is also possible to introduce depth in each of these three blocks, creating a Deep RNN. We can think of the lower layers in the hierarchy inside the RNN graph as playing a role in transforming the raw input into a representation that is more appropriate, at the higher levels of the hidden state.

Also RNN uses backpropagation and the information about the gradient to minimize the MSE during the training phase.

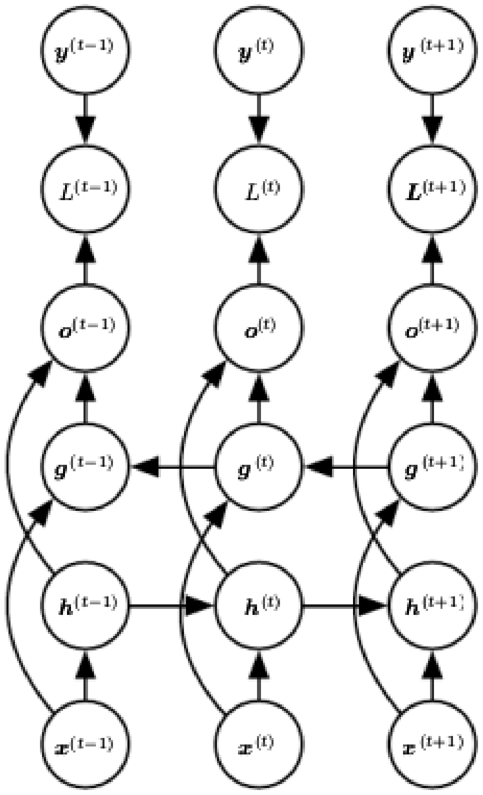

**Figure 2.6:** Computation of a typical bidirectional RNN [46].

**Bidirectional RNNs** All of the recurrent networks we have considered up to now have a "causal" structure, meaning that the state at time $t$ only captures information from the past, $x_1, ..., x_{t-1}$, and the present input $x_t$. However, in many applications, we want to output a prediction of $y_t$ which may depend on the whole input sequence. Bidirectional RNN were invented to address that need.

As the name suggests, bidirectional RNNs combine a RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence. Figure 2.6 illustrates the typical bidirectional RNN. The output units compute a representation that depends on both the past and the future.

**Long Short-Term Memory** When training a vanilla RNN using backpropagation, gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization). This is known as the mathematical challenge of learning long-term dependencies in RNN. Even if we assume that the parameters are such that the recurrent network is stable, the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions compared to short-term ones.

Long Short Term Memory (LSTM) network is a RNN, aimed to deal with the vanishing gradient problem. It aims to provide a short-term memory for RNN that can last thousands of timesteps.

As visible in Figure 2.7, a common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell:

- Forget gates decide what information to discard from a previous state by assigning a value between 0 and 1.
- Input gates decide which pieces of new information to store in the current state, using the same system as forget gates.
- Output gates control which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.

Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time steps.

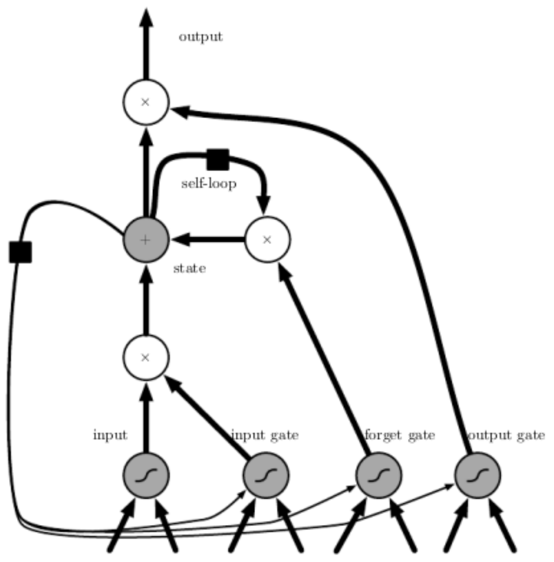

**Figure 2.7:** Block diagram of the LSTM recurrent network "cell." [46].

## 2.3 Detection Approaches

In this section, we are going to describe the specific math and detection models that are the basic knowledge to understand each detector used inside the DIDS.

### 2.3.1 Inter-Arrival Mean and Range Models

Sampling distribution of the sample mean and sample range are two metrics widely used in statistical process control to monitor the stability of a production process. In this work, the implemented NIDSs used these two concepts inside the metrics called Inter-Arrival Mean (IAM) and Inter-Arrival Range (IAR).

As explained by Lin et al. [30], assume to take a few sample sets of a certain attribute from $W$ items, e.g., height of students, as $X = \{x_1, ..., x_W\}$. The sample mean is defined as:

$$\bar{X} = \sum_{i=1}^{W} \frac{x_i}{W}. \tag{2.2}$$

It can provide a measure of central tendency. The distribution of $\bar{X}$ is called the sampling distribution of the sample mean. For a finite number of sample means $\bar{X}_j$, $j = 1, ..., k$, it is possible to compute their center of distribution as

$$\hat{X} = \sum_{j=1}^{k} \frac{\bar{X}_j}{k}. \tag{2.3}$$

Based on CLT, for any population with mean $\mu$ and standard deviation $\sigma$, the sampling distribution of the sample mean $\bar{X}$ tends toward being normally distributed when the sample size increases, with:

$$\mu_{\bar{X}} = \mu. \tag{2.4}$$

$$\sigma_{\bar{X}} = \frac{\sigma}{W}. \tag{2.5}$$

Where $\sigma_{\bar{X}}$ is the standard deviation of the population when the sample size increases. The sample range can state the natural variation in a process, and it is computed as:

$$R_j = max(X_j) - min(X_j). \tag{2.6}$$

The distribution of $R_j$ for finite sets of $X_j$ is called sampling distribution of the sample range. The center of this distribution is:

$$\bar{R} = \sum_{j=1}^{k} \frac{R_j}{k}. \tag{2.7}$$

People in this area usually assume the population they take samples from follows a normal distribution. Under this assumption, one can estimate the standard deviation $\sigma_R$ with $\bar{R}$ and sample size. However, we do not make any specific assumption on the distribution of the population, but instead we try to estimate it.

*Sample mean and sample range display variations from their historical distribution* when the process is stable. If the variations exceed predefined thresholds, Upper Limitation (UL) and Lower Limitation (LL), it means the system conditions changed. In this work, we use mean and range to model the message inter-arrival times. For the sake of simplicity, we refer to the sampling distribution of the sample mean and the sample range as the mean model and the range model in the rest of the paper.

For every event set $E = \{e_1, ..., e_m + 1\}$, there exists a corresponding set of inter-arrival times $T = \{t_1, ..., t_m + 1\}$ in the dataset. Now, we show how to build mean model and range model.

**Mean model** Instead of computing the center of the sampling distribution of the sample mean $\hat{X}$ as in equation 2.3, we use CLT to construct the mean model based on equations 2.4 and 2.5. Since the $\mu$ and $\sigma$ are unknown, we estimate them with the mean and standard deviation of the subpopulation $T$ with size $m$:

$$\mu \approx \bar{T} = \frac{1}{m}\sum_{i=1}^{m} t_i. \tag{2.8}$$

$$\sigma \approx S_T = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(t_i - \bar{T})^2}. \tag{2.9}$$

We set detection thresholds UL as $\mu_{\bar{X}} + N\sigma_{\bar{X}}$, while LL as $\mu_{\bar{X}} - N\sigma_{\bar{X}}$, where $N$ is a performance parameter called threshold level. Note that the LL needs to be positive to provide detection capability since all the inter-arrival times are positive.

$$\mu_{\bar{X}} - N\sigma_{\bar{X}} > 0. \tag{2.10}$$

Equations 2.9, 2.8 and 2.10 imply that the LL is positive when the sample set size $W > (\frac{N\sigma}{\mu})^2$.

**Range model** We continue to use the selected sample size $W$ and event set $E$. For every $W + 1$ events, there exists a set of inter-arrival times $T^j = \{t_1^j, ..., t_W^j\}$, $j = 1, ..., \lfloor \frac{m+1}{W+1} \rfloor$. We calculate the sample range $R_j$ using equation 2.6, the center of the sampling distribution of the sample range $\bar{R}$ is computed using equation 2.7 and using the equation 2.9 with $\bar{R}$ instead of $\bar{T}$ we estimate $\sigma_R$. The detection threshold UL is defined as $\bar{R} + N\sigma_R$, while we set LL as the smallest event inter-arrival time in the learning period since the range model can be asymmetric.

### 2.3.2 Random Forest

Another way to reduce the danger of overfitting in Decision Tree is by constructing an ensemble of trees called Random Forest (RF). It is a supervised ML model that, as Decision Tree, is widely used in classification and regression tasks. RF is a classifier consisting of a collection of Decision Trees, where each tree is constructed by applying an algorithm on the training set and an additional random vector sampled i.i.d. from some distribution. The prediction of the random forest is obtained by a majority vote over the predictions of the individual trees [7], as visible in Figure 2.8.

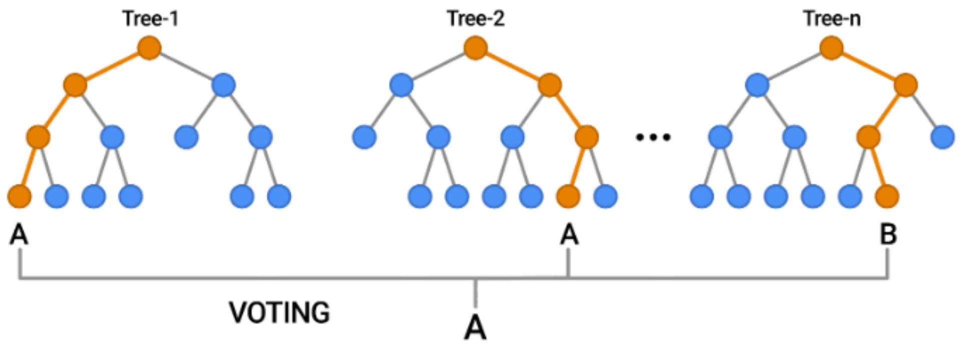

**Figure 2.8:** RF's voting outcome example [41].

A Random Forest is based on an ensemble technique called Bootstrap Aggregation (or Bagging). The algorithm consists on repeatedly taking (with replacement) a number of random records from the dataset where individual decision trees are constructed for each sample. Each decision tree will generate an output and the final result consists on majority voting or averaging, for classification and regression respectively. Due to the fact that Random Forest is an ensemble method composed of a multitude of individual classifiers, it is slower than a single decision tree but better avoids overfitting and in general, is a more stable model.

### 2.3.3 Bidirectional Long Short-Term Memory

The Bidirectional Long Short Term Memory (BLSTM) basically is a bidirectional RNN in which its units are LSTM units, as visible in Figure 2.9.

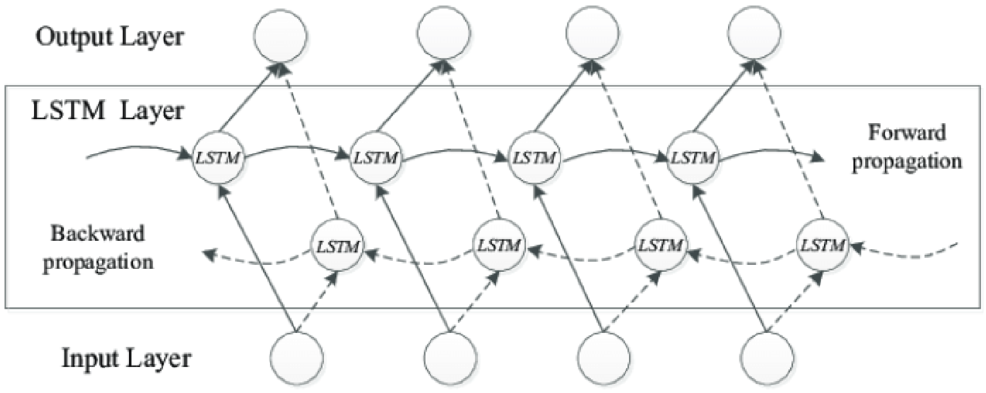

**Figure 2.9:** Example bidirectional LSTM network [48].

BLSTM is usually composed of an embedding layer, a bidirectional LSTM, a dropout layer, the REctified Linear Unit (RELU), a dropout layer, and sigmoid activation functions. The main purpose of the embedding layer is to represent the input so it can be incorporated into the model and be better adapted to the corresponding tasks. The dropout layer is used for regularization on DL which prevents model overfitting, while RELU and sigmoid are two well-known activation functions.

# Chapter 3

# Literature Review

Intrusion detection is a well-formed problem for IT and OT systems. The area of anomaly and intrusion detection in ICS has been widely studied. Extensive surveys are devoted to classifying research in this field [6] [8] [9]. Our review of related works focuses on NIDS, PIDS, and DIDS to detect cyber-attacks.

## 3.1 Physics-based Intrusion Detection System

Some approaches are based on the operational data from the physical system. Based on the data collected, a model is trained on the normal and abnormal behavior of the process. The trained model is then placed in the knowledge base of IDS. It is later used by the detector to detect intrusions.

Traditional PIDS rely on statistical techniques [16], such as the mean and standard deviation of sensor readings. Lately, ML techniques are being used extensively as physics-based approaches to secure ICS. To develop a PIDS, some studies used autoregressive models [11], [12] or linear dynamical system modeling [13], [14], [15] for system state prediction. Unfortunately, both approaches' assumptions include linearity of the modeled system which is not typically fulfilled in ICSs. Goh et al. [22] used an unsupervised learning approach based on Recurrent Neural Networks (RNNs) and the Cumulative Sum method to identify anomalies in a replicate of a water treatment plant. Junejo et al. [24] proposed a behavior-based ML approach for intrusion detection that models the physical process of the CPS to detect any anomalous behavior or attack. Raman et al. [28] presents a SCADA specific Probabilistic Neural Network (PNN)-based anomaly detector using a supervised approach to detect anomalies possibly resulting from attacks. Wolsing et al. [5], considering the repetitive nature of physical processes in IIDSs, propose simpler detection methods, like checking the value of the processes is between a maximum and minimum range, as an alternative of complex models with all their disadvantages.

## 3.2 Network-based Intrusion Detection System

Anthi et al. [17], Anton et al. [18] and Colelli et al. [19] present some IDSs which uses different supervised approach, like Support Vector Machine (SVM), to map all features into a vector space and derive decision boundaries to separate individual classes, and RF to split a dataset's features into similar classes to detect cyber-attacks in industrial control systems networks. Lai et al. [20], Feng et al. [21] used Neural Networks (NNs) to classify the features of network packets as benign or malicious. Indeed, ML-based approaches increase utility compared to deterministic signature-based intrusion detection through (i) generalizability across domains and (ii) the ability to identify novel, not previously seen, anomalies. Kharitonov et al. [25] implement a semi-supervised algorithm allowing to train models without any knowledge about future attacks. The periodic behavior of industrial plants allows the building of a normal model of the communication of network participants. Perez et al. [26] assesses the performances of ML techniques such as SVM, RF and BLSTM. Grammatikis et al. [27] implements a decision tree classifier responsible for recognizing specific DNP3 cyberattacks, and an autoencoder DNN capable of detecting DNP3 anomalies either due to a potential security violation or an electricity disturbance. Lin et al. [30] note that SCADA traffic exhibits persistent and stable communication patterns, so studied three attack scenarios formed by valid requests only and then proposed an anomaly detection system, which uses sampling distribution of sample mean and sample range to model the timing of repeated events.

## 3.3 Distributed Intrusion Detection System

Homayouni et al. [23] developed an LSTM-Autoencoder-based approach to finding anomalies in multivariate time-series data. Nevertheless, their approach is supervised, and this means that it works with a strict set of well-defined attacks. In [34], a security vendor presents a commercial solution able to monitor IT traffic, OT traffic and process state, combining behavior and signature detection. The implementation details were not disclosed to the scientific community, which made it difficult to study and compare. KingFisher, an IDS architecture implemented by Bernieri et al. [35], uses unsupervised learning detection combined with VAE, that is the probabilistic version of the classic Autoencoder. However, there is still too much gap with a realistic scenario in which more attacks can be in place, and deep studies were not conducted to prove the real effectiveness of KingFisher.

Our work combines more simple and understandable metrics used by Wolsing et al. [5] and Lin et al. [30] to detect the presence of cyber-attacks in both the network traffic and operational data source. This allows the creation of a not-too-complex, transparent, and high-reliability anomaly detector, and thanks to a RF model, it is also possible to classify the well-known attacks on the network. Table 3.1 shows the different features and data sources used in all the previously mentioned works, and the features implemented in each work. We can see that our DIDS is the only one that considers two sources of information for the detection, identifies the ongoing attack by fingerprinting it, and uses simple detection methods.

**Table 3.1:** Papers comparison on the features implemented: network traffic or operational data source and Fingerprint of the ongoing Attacks (FA).

| Features / Papers | Network | Operational | FA |
|---|---|---|---|
| Ye et al. [16] | x | ✓ | x |
| Hadžiosmanović et al. [11] | x | ✓ | x |
| Mashima et al. [12] | x | ✓ | x |
| Mishra et al. [13] | x | ✓ | x |
| Murguia and Ruths [14] | x | ✓ | x |
| Mo et al. [15] | x | ✓ | x |
| Goh et al. [22] | x | ✓ | x |
| Junejo et al. [24] | x | ✓ | x |
| Raman et al. [28] | x | ✓ | x |
| Wolsing et al. [5] | x | ✓ | x |
| Anthi et al. [17] | ✓ | x | x |
| Anton et al. [18] | ✓ | x | x |
| Colelli et al. [19] | ✓ | x | x |
| Lai et al. [20] | ✓ | x | x |
| Feng et al. [21] | ✓ | x | x |
| Kharitonov et al. [25] | ✓ | x | x |
| Perez et al. [26] | ✓ | x | x |
| Grammatikis et al. [27] | ✓ | x | x |
| Lin et al. [30] | ✓ | x | x |
| Homayouni et al. [23] | ✓ | ✓ | x |
| Nozomi Networks [34] | ✓ | ✓ | x |
| Bernieri et al. [35] | ✓ | ✓ | x |
| Our DIDS | ✓ | ✓ | ✓ |

# Chapter 4

# Project Overview

In this chapter, we are going to give a general overview of the components implemented for our thesis.

The purpose of this thesis is to prove that a DIDS that monitors *different points in the network* and considers both physical and network data, has an advantage over the traditional state-of-the-art detector specialized in a single type of data in a single point. For this reason, the DIDS developed checks all of these two sources of data. The DIDS consists of the following modules:

- **Attack Detector**, this module is used to detect anomalies caused by cyberattacks. It is composed of the following subcomponents:
    1. *Preprocessor*, is responsible for preprocessing operational and network data coming from the sensors, actuators, and network devices to convert them into a common format that is acceptable by the IIDSs.
    2. *Network Detector*, it consists of two or more NIDSs scattered in the network to detect if the system is under attack.
    3. *Physical Detector*, it uses four PIDS to detect if the system is under attack.
    4. *Combiner*, it combines the results of NIDS and PIDS to decide whether to raise an alarm.
- **Attack Identifier**, this component fingerprints the attacks' patterns with the goal of classifying them. It is composed of the following subcomponents:
    1. *Preprocessor*, collects traffic and splits it into bi-directional streams. It starts after the Attack Detector raises an alarm. It also cleans the data to give them in input to a ML model.
    2. *Attack Classifier*, identifies the specific attack that is in action given the statistical network traffic features.

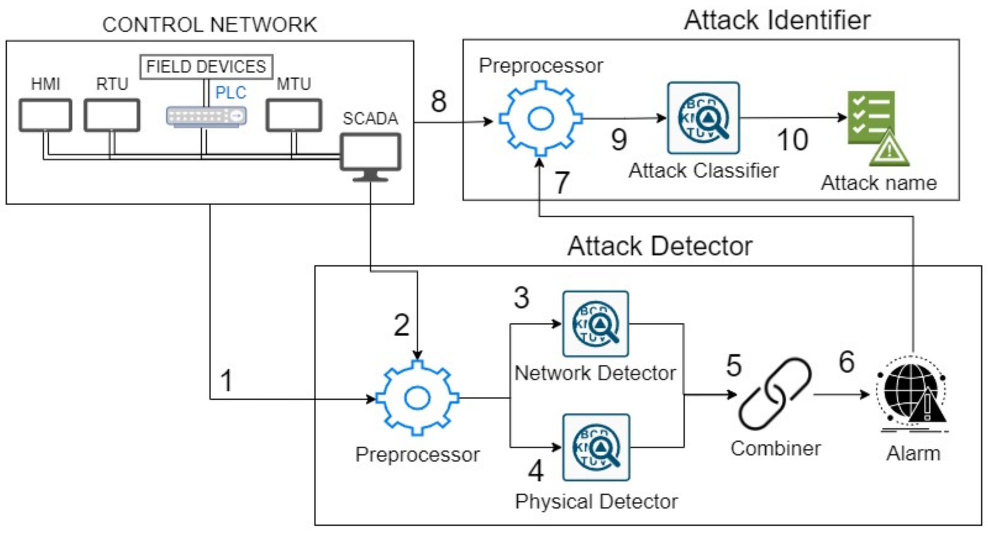

**Figure 4.1:** DIDS architecture and execution flow divided into the different modules.

In Figure 4.1 it is possible to see the pipeline of our thesis project and the order of the steps taken from the detector preprocessing phase to the classification phase. After the Attack Detector Preprocessor gets the network traffic and the operational data, it does its job and passes them in a JSON format to the Physical and Network Detectors, which look for the presence of attack(s). Their results are passed to the Combiner, which has to decide whether to raise an alarm effectively. If the alarm is raised, then the Combiner wakes up the Attack Identifier Preprocessor, which preprocesses the network traffic another time. The cleaned data are given in input to the Attack Classifier, which finds what is the identifier of the ongoing attack. Notice that the Network Detector and Physical Detector may run in parallel to optimize the work.

In the following section, we overview each module in detail. Except for the Attack Classifier, all the detection techniques used inside the DIDS are unsupervised.

## 4.1 Attack Detector

When an attack is happening inside the ICS, the Attack Detector module's goal is to recognize that *something ambiguous is happening*. In the following paragraphs, we are going to explain how each subcomponent contributes to reaching this objective.

**Preprocessor** The responsibility of the Preprocessor is to preprocess operational and network data so that it is possible to give them in input to the detectors contained in the DIDS. For this reason, the Preprocessor is composed of two subcomponents: IPAL Transcriber and Physical Preprocessor.

Industrial Protocol Abstraction Layer (IPAL) Transcriber is a tool developed by Wolsing et al. [31] which allows a *common representation* for many industrial protocols to uniform the preprocessing. Concerning the PIDS, the Physical Preprocessor preprocesses the information about the sensors and actuators with the same purpose as IPAL Transcriber.

During the conversion, the data are cleaned, to correct or remove corrupted/inaccurate records from the dataset, and labeled.

**Network Detector** After the preprocessing phase, we have common formatted files. So, to detect the presence of attacks in the ICS, we give in input the converted files from the network traffic data to two NIDSs using two detection techniques: IAM and IAR. They are going to be better explained in section 5.3.2.

**Physical Detector** Other than NIDSs, we want to monitor the physical processes to have a better estimate of what is happening in the system. To do that, we used four detection techniques for PIDS: MinMax, ST, GT, HM.

The preprocessed information that contains, for each iteration, all the actual values of all the sensors and actuators, are given in input to physical detectors, which apply their detection techniques and they will say if there is the presence of attacks or not examining each record of values. They are going to be better explained in section 5.3.3.

**Combiner** To combine the results from all the IIDSs inside the DIDS, a methodology is needed. To tackle this task, the Combiner assigns a weight to each detector. The weights of the IIDSs which found an attack during the detection phase are summed all together, and the final sum is compared with a threshold. If it is greater than the threshold, an *alarm* is raised and the Attack Identifier module starts its execution, otherwise, nothing happens.

## 4.2 Attack Identifier

Each attack leaves traces inside the ICS, which the Attack Identifier module captures *to fingerprint* and *identify* it. In the following paragraphs, we are going to explain how each subcomponent contributes to reaching this objective.

**Preprocessor** The first step to classify the ongoing attack is to use CICFlowMeter [40]. Formerly known as ISCXFlowMeter, it is an Ethernet traffic Bi-flow generator and analyzer for anomaly detection that has been used in many Cybersecurity datasets.

In this project, CICFlowMeter is used to create a file containing *statistical features of bidirectional flows* in the network traffic. After the features are obtained, this component is responsible for cleaning the data to correct or remove corrupt or inaccurate records from the dataset. Finally, the data are labeled.

**Attack Classifier** This module takes in input the preprocessed statistical network traffic features to discriminate the traffic. This thesis makes use of RF to *classify the network traffic* as under a specific attack or not.

# Chapter 5

# Implementation

As stated by Conti et al. [6], there are different possible classifications of a testbed: physical, virtual, or hybrid testbed.

Both the network and physical layers may be configured using real hardware and software with physical testbeds. Researchers can use them to collect realistic measurement variation and latencies. However, physical testbeds are expensive both in construction and maintenance. They generally have a long building time, and they may not provide a safe execution of dangerous physical processes (e.g., the nuclear sector).

For this reason, we opted to use a DT (i.e., virtual testbed) to achieve a high-fidelity simulation. As opposed to physical testbeds, virtual testbeds use software simulations and emulation to reproduce the entire network and all its components. Due to the virtualized environment, it is difficult to simulate *high-fidelity* physical processes with a virtual testbed, other than using DTs. However, dangerous processes can be simulated in a laboratory. Furthermore, they are easy to update and upgrade, making them flexible and extensible.

In the next sections, we are going to discuss the used virtual testbed DHALSIM, the conducted cyber-attacks on it, and the implementation details of each module seen in chapter 4.

## 5.1 DHALSIM

The testbed used for the experiment is Digital HydrAuLic SIMulator (DHALSIM). As stated by Murillo et al. [33], DHALSIM is a cyber-physical simulation and emulation tool for smart Water Distribution System networks. The choice of this emulator is motivated by the fact that it provides realistic ICS traffic corresponding to *high-fidelity simulation of physical processes, effects of network traffic faults, and cyber-attacks on distributed control and physical processes.* The virtual testbed provides a way to generate different network topologies and offers a modeling platform on which Digital Twin (DT) of both physical and cyber layers could be developed.

As explained by Murillo et al. [33], the physical simulation tool used by DHALSIM is EPANET [42], while the network emulation tools are MiniCPS [43] and Mininet [44]. EPANET is used to model the physical layer of a smart water network, while Mininet and MiniCPS are used to create virtual networks. Mininet is a platform to create virtual networks inside a single host machine. These virtual networks can connect virtualized guests. MiniCPS is built on top of Mininet and provides an implementation of two popular industrial communication protocols: ENIP/CIP and Modbus.

The experiments we performed were applied on two different ICS infrastructures reproduced with DHALSIM: Anytown and KY15. These latter two are examples of water distribution systems chosen by the authors of the simulator. They have in common the general structure of the network, as visible in Figure 5.1, which consists always of a *star topology* with one central router (Router0) and a variable number of subnetworks connected to it. Each subnetwork is composed of a router connected to the central router and to a switch, this latter connected to a PLC, that is connected to a variable number of sensors and actuators. Also, the number of iterations used in each execution and the kind of attacks launched to the ICS are constantly used on both topologies. However, KY15 is *more complex* than Anytown: this latter architecture has four subnetworks, and uses three PLCs, two actuators, and two sensors; while KY15 architecture has seven subnetworks, uses six PLCs, seven actuators and seven sensors.

DHALSIM uses iterations as an atom for the timestamp. We have analyzed the iteration time to give an estimate of how many seconds it requires since Murillo et al. [33] work doesn't specify it. We found the mean and standard deviation of the difference between the next iteration's initial timestamp and the actual iteration's initial timestamp, which we'll call "iteration time" from now on, for each run simulation. For what concern the Anytown topology, the mean iteration time is 3.646060 seconds and the standard deviation iteration time is 1.433604 seconds, while for the KY15 topology, the mean iteration time is 18.177534 seconds and standard deviation iteration time is 55.581810 seconds. However, the iteration time depends not only on the ICS topology but also on the computer used to run the simulation. For this reason, this mapping between iteration and seconds is *purely indicative*, and for the sake of the comparison that is going to be discussed in sections 6.5 and 6.6.

What changes between the specific infrastructures is the number of subnetworks, sensors, and actuators, but also the demand pattern and the sensors and actuators allowable values. There is always one PLC, one router, and one switch inside each subnetwork.

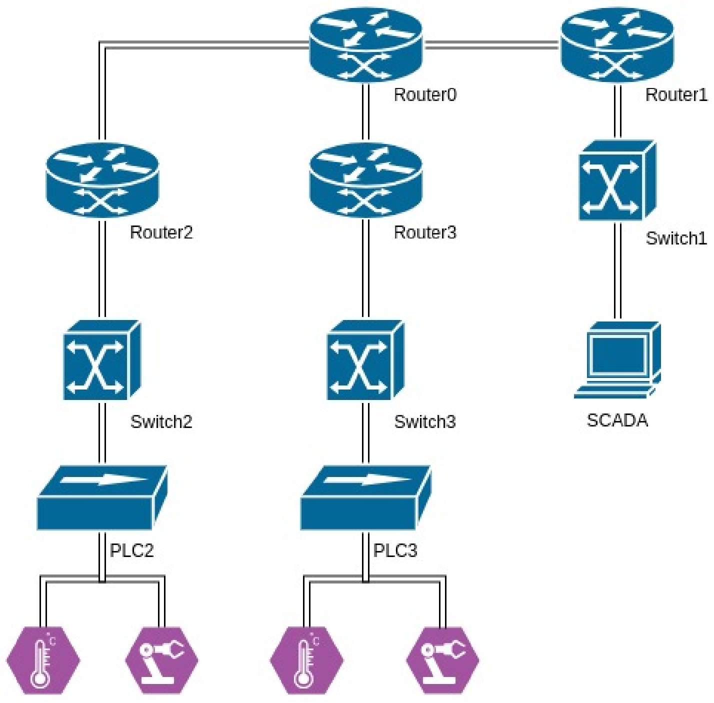

**Figure 5.1:** ICS base topology used in all the runned simulations.

The outputs of the testbed are PCAP files containing the network packets, one PCAP for each interface in the network except for the central router (Router0), the SCADA operational data that contains for each iteration all the values of each sensor and actuator, and the ground truth file, useful for the labeling process.

## 5.2 Cyberattacks

During the simulation, DHALSIM allows to launch a set of pre-defined and configurable attacks. The attacks used to create our dataset are:

- Device attack, is performed at the PLC itself to manipulate actuators value, which can be opened or closed. The attacker has physical access to the PLC being attacked.
- Naive Man-In-The-Middle (NITM) is an attack where the attacker will sit in between a PLC and its connected switch. The attacker will then modify the values of all CIP packet payload fields directed towards the other PLCs and SCADA device.
- Man-In-The-Middle (MITM) consists of an attacker sitting in between a PLC and its connected switch. This is the most simple attack, it will manipulate all CIP packets going through a network link, regardless of the tag (value of sensor/actuator) that the package includes.

- Server Man-In-The-Middle (SITM) are attacks where the attacker will sit in between a PLC and its connected switch. This attack causes the attacker to launch a CIP server and then serve the target using that server. It will create a new TCP connection and ENIP session between the attacker and the victim.
- During Concealment Man-in-the-Middle (CITM) attack, the traffic going towards a PLC and towards the SCADA is differentiated. For the PLC, it will manipulate the tag values with the attack values. For the SCADA, it will manipulate the tag values with the concealment values.
- Simple Denial of Service (DoS) attack interrupts the flow of CIP messages containing data between PLCs. This attack first performs an ARP Spoofing attack into the target and then stops forwarding the CIP messages. This will cause the PLCs to be unable to update their cache with new system state information, possibly taking wrong control action decisions.

As described a bit by Alcaraz et al. [39] survey, our MITM attacks launched to the virtual testbed compromise legitimate devices and consequently interfere with communication channels. The MITM performs and causes the following actions:

- It launches routing attacks to play with the DT traffic from the physical space.
- It creates deviations that could deteriorate the Quality of Service.
- Injects false data and modifies control packets.

The simple DoS used in this thesis is a particular type of selective forwarding attack. The targeted PLC doesn't receive new system state information since the attacker stops forwarding CIP packets. Indeed, as explained by Khan et al. [50], in one form of the selective forwarding attack, the malicious nodes can selectively drop the packets coming from a particular node or a group of nodes. This behavior causes a DoS attack.

## 5.3 Attack Detector Implementation

In this section, we are going to describe the implementation details of all the Attack Detector subcomponents, visible in Figure 4.1.

### 5.3.1 Preprocessor

The Preprocessor is composed of two subcomponents: IPAL Transcriber and Physical Preprocessor. IPAL Transcriber allows the conversion of every packet inside a PCAP file into a JSON-formatted record composed of the following fields:

- Meta Data: packet's timestamp, length, a unique Identifier (ID), and a label to say if the packet is a malicious one.
- Addressing Information: source and the destination of a single packet represented as an IP-port combination, that can be extended, e.g., by adding Modbus's unit identifier field, to further disambiguate devices. The destination can remain empty for broadcast protocols.

- Message Identification: protocol, generic activity (one between requests, commands, and their respective answers), and the "responds to" field that lists the IDs of all IPAL packets a given message is a response to.
- Process Data: collects all process variables and their current values taken from the packet's payload.

In this way, our subcomponent preprocesses the PCAP files containing all the exchanged packets in the ICS to a format accepted by our NIDSs and that is equal for all the industrial network protocols.

Concerning the PIDS input, we must preprocess the information about the SCADA physical processes. Indeed, DHALSIM stores this information inside a CSV file which contains, for each iteration, all the actual values of all the field devices. However, since our PIDS takes in input a JSON file formatted in a specific way, the Physical Preprocessor must convert the CSV into a common JSON format consisting of the following fields:

- Meta Data: a timestamp, an ID, and a label to say if the record is a malicious one.
- Process Data: name of the actuator/sensor and its current value. It can be repeated a number of times equal to the number of sensors and actuators inside the ICS topology.

### 5.3.2 Network Detectors

A way to detect the occurrence of previously described cyber-attacks is to monitor the network to notice something anomalous. The two detection techniques included in the Attack Detector module to understand what traffic can be considered normal and what instead is suspicious are IAM and IAR.

As explained by Lin et al. [30] and as we saw in section 2.3.1, due to the use of request-response communication in polling, SCADA traffic exhibits stable and predictable communication patterns identifiable using a timing-based anomaly detection that monitors statistical attributes of traffic periodicity. Specifically, IAM and IAR use sampling distribution of the mean and the range to model the inter-arrival times of repeated messages. This method has been widely used for its *easiness* and *efficacy* in statistical process control areas to monitor the stability of processes.

Using the IPAL transcriber in the preprocessing phase all the network packets are converted to a JSON format which is *independent of the protocol.* In this way, it is possible to create the mean and range models of inter-arrival times events setting detection thresholds as previously described in section 2.3.1 and explained in the following. For the IAM, the mean model is computed in the following way:

- Calcule $\bar{T}$, that is the mean $\mu$ of the subpopulation $T$ with $m$ inter-arrival times, and it is estimated using equation 2.8, where $t_i$ is the $i$-th inter-arrival time.
- Calculate $S_T$, that is the standard deviation $\sigma$ of the subpopulation $T$ with $m$ inter-arrival times, and it is estimated using equation 2.9, where $t_i$ is the $i$-th inter-arrival time.

- Use CLT as in equation 2.4, that states the mean model of the inter-arrival times tends toward being equal to the mean $\mu$ of the subpopulation $T$ when the sample size increases.
- Use CLT as in equation 2.5, that states the standard deviation when the sample size increases of the inter-arrival times tend toward being equal to the standard deviation $\sigma$ of the subpopulation $T$ divided by the sample size $W$.
- Set the detection thresholds UL as $\mu_{\bar{X}} + N\sigma_{\bar{X}}$, while LL as $\mu_{\bar{X}} - N\sigma_{\bar{X}}$, where $N$ is a performance parameter called threshold level.

For the IAR, the range model is computed in this way:

- Calcule the sample ranges as in equation 2.6 for each sample window with size $W$.
- Compute the center of the sampling distribution of the sample range $\bar{R}$ with equation 2.7.
- Applying the equation 2.9 with $\bar{R}$ instead of $\bar{T}$, it is possible to estimate the standard deviation $\sigma$ of the subpopulation $T$ with size $m$.
- UL is set to $\bar{R} + N\sigma_R$, where $N$ is the threshold level, while we set LL as the smallest event inter-arrival time in the learning period.

During the training phase, the UL-LL interval is computed for each window composed of the packets with few *common packet fields*. Indeed, in Listing 1, it is possible to see that UL and LL are computed using $\mu$ and $\sigma$ related to the sample mean of the differences between consecutive messages timestamps related to specific IPs, type of communication, i.e., interrogate or inform, and field device. For example in line 11, UL, LL, $\mu$ and $\sigma$ are computed only for all the packets inside a window of length two with source IP 10.0.2.1 that interrogates 192.168.1.1 to know the value of the T42 sensor. Notice that it is not true to say inter-arrival methods consider the timestamps differences between the interrogate-inform pair of messages to build the sample mean and sample range model, as one wrongly can think, but they consider the packets with the same IPs, type of communication, and targeted field device.

During the detection phase, the module uses a *sliding window* which has the same window size as the sample size $W$. The module calculates the sample mean and sample range in each window containing the packets with the same source and destination IPs, the same type of communication, and the same involved field device as the ones found by the models. The Network Detector launches an alarm if the mean or the standard deviation is outside the UL-LL interval.

For example, assume to use the Linux epoch timestamps format, and consider UL = 6.2, LL = 3.8 and the $W = 3$ consecutive timestamps of all the interrogate messages from `10.0.1.1` to `192.168.1.1` that ask for the value of the sensor T42 are 1688139274.232246, 1688139277.448214 and 1688139287.73056. The differences between the couples of consecutive messages are 3.215968 and 10.282346, so their average is 6.749157 and the standard deviation is 3.581114615. In the case of IAM, $6.749157 > UL = 6.2$, while for IAR we have that $3.581114615 < LL = 3.8$, and this means that all the two detectors find an attack.

In this way, IAM and IAR check for *temporal variations* related to the transmitted messages of the same type in order to detect anomalies inside the ICS.

```json
{
    "_name": "inter-arrival-mean",
    "settings": {
        "_type": "inter-arrival-mean",
        ...
        "N": 5,
        "W": 2,
        ...
    },
    "mean_model": {
        "10.0.2.1-192.168.1.1-interrogate-76-T42": {
            "ll": 0.047261748369725964,
            "ul": 4.330687277659088,
            "mu": 2.188974513014407,
            "sigma": 0.4283425529289362
        },
        "192.168.1.1-10.0.2.1-inform-76-T42": {
            "ll": 0.060901387488802694,
            "ul": 4.317092888411967,
            "mu": 2.188997137950385,
            "sigma": 0.4256191500923165
        },
        "10.0.1.1-192.168.1.1-interrogate-76-T42": {
            "ll": 2.3608530834133963,
            "ul": 7.165241443373931,
            "mu": 4.763047263393664,
            "sigma": 0.48043883599605347
        },
        "192.168.1.1-10.0.1.1-inform-76-T42": {
            "ll": 2.389048313560807,
            "ul": 7.137232639572137,
            "mu": 4.7631404765664715,
            "sigma": 0.47481843260113293
        }
    }
}
```

**Listing 1:** JSON example of the UL-LL interval computed for each time window by the training of IAM using window size 2 and threshold level 5.

### 5.3.3 Physical Detectors

The implemented DIDS comprise also four PIDSs evaluated and implemented by Wolsing et al. [5] in their work. They are also used inside the Attack Detector module:

- The MinMax approach detects whether a sensor's/actuator's current value exceeds the range observed in the training data and raises an alarm if any observation falls outside that range (± error margin). This approach is motivated by the intuition that process values of industrial systems relate to physical measurements or setpoints and thus usually obey certain limits.
- The GT approach detects whether a sensor's/actuator's slope exceeds the minimum and maximum observed during training (± error margin). While MinMax observes global changes, more subtle attacks occurring within these limits may remain unnoticed. Hence, the GT approach assumes that ICSs have continual character, i.e., physical values such as temperatures cannot change at arbitrary speed.
- Focusing on another temporal aspect, the ST approach detects whether a sensor/actuator remains static, i.e., does not change its value, for a shorter or longer time than seen during training (± error margin). This approach is motivated by the observation that some attacks may freeze a sensor/actuator such as a pressure relief valve. Since a steady state is difficult to define for noisy sensor data, Steadytime takes only process values into account if the number of distinct values during training is sufficiently small ($\leq 10$).
- Specifically targeting the occurrence of values, the HM approach tracks their distribution within a fixed-sized window and tests whether it is in line with a histogram seen during training (± error margin). The underlying intuition expects a similar distribution of reoccurring values between process cycles. This approach can detect the existence and absence of frequent value changes. The histograms are created by counting the number of times each distinct value appears in a sliding window. We merge them into a single histogram that covers each value's minimum and maximum occurrences across all distinct fixed-sized windows. The window size should match the duration of a process cycle, which could be automatically determined in an additional run over the dataset before training the histograms. Like Steadytime, Histogram only applies for process values with a few distinct values ($\leq 10$), as comparing two histograms value-by-value is unfeasible for noisy sensor data.

### 5.3.4 Combiner

All the outputs of the detectors included in our DIDS need to be considered in a certain way. It is here that the Combiner joins the game.

Given some weights, the Combiner first normalizes them in order to be between 0 and 1 using the preprocessing tool offered by sklearn, then it assigns them to each detector. These weights help to sort by importance each detector: the higher they are, the more important the output of the assigned detectors are. After each detector inside the DIDS ends its detection process, if it finds an anomalous behavior, its weight is considered inside a summation. So, this summation contains the value of *all the weights assigned to the detectors that find an anomalous behavior.*

Finally, the value of this sum is *compared with a given threshold*, and if it is greater, then the DIDS identify an attack, otherwise, nothing happens.

## 5.4 Attack Identifier Implementation

Now, we are going to discuss the implementation of all the Attack Identifier subcomponents, visible in Figure 4.1.

### 5.4.1 Preprocessor

This subcomponent has the role of preprocessing the network traffic so it is possible to input some features to the Attack Classifier module. It makes use of CICFlowMeter [40] and of the sklearn class called Pipeline [45].

Formerly known as ISCXFlowMeter, CICFlowMeter is an Ethernet traffic Bi-flow generator and analyzer for anomaly detection. In this project, it is used to create a CSV file containing *statistical features of bidirectional network traffic flows*. Each bidirectional flow is composed of forward (source to destination) and backward (destination to source) directions during the exchange of network packets [40]. In this way, using CICFlowMeter it is possible to extract more than 80 network traffic analysis features: duration of the flow, total packets in the forward direction, total packets in the backward direction, the total size of the packet in the forward direction, the total size of the packet in the backward direction, the mean size of packet in forward, and so on, that are calculated separately in the forward and backward directions. These features can be the input of a ML model.

Then, labeling and feature selection processes are needed since we give these features in input to a supervised ML model. Later, the Preprocessor splits the samples into train and test sets. Since the support of samples labeled as under an attack is low, the Preprocessor uses SMOTE to synthesize new examples from the existing ones, as explained in the subsection 2.2.1.

After that, the obtained data are "cleaned" to correct or remove corrupt or inaccurate records from the dataset, so identifying incomplete, incorrect, inaccurate, or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. To do all the required operations sequentially, we use Pipelines. They are useful for transforming and training data quickly. It is a way to codify and automate the workflow using multiple sequential steps that do everything from data extraction and preprocessing to model training and deployment. We used the component Pipeline offered by the sklearn library to do that. The purpose of the Pipeline is to assemble several steps that can be cross-validated together while setting different parameters [45].

### 5.4.2 Attack Classifier

The Attack Classifier module contains a RF model described in section 2.2. It takes in input cleaned, preprocessed statistical features of network traffic flows obtained thanks to CICFlowMeter.

Each attack has its pattern of impact on the network traffic. The idea of this module is to *fingerprint each attack* and train a model in such a way that once the anomaly has been identified, we are also able to recognize the attack that generates it. Looking at these features, we found that the Decision Tree can do it by determining the conditions to fulfill to classify the bidirectional traffic flow features. Then, to avoid overfitting, we decided to use RF.

We take the RF classifier from the sklearn.ensemble module, which includes ensemble-based methods for classification, regression, and anomaly detection.

# Chapter 6

# Experiments

In this chapter, we are going to discuss the experimental setup used to conduct our experiments, an analysis of the network traffic and operational data values during an ongoing attack, and the final results of each detector and classifier trained, discussing them in the meantime. The optimal results and hyper-parameters are obtained by performing a *grid search* on each detector.

When we are going to discuss the results, we'll introduce the detection time of each detector. Before we proceed, it is necessary to clarify that we consider the attacks officially started according to the timestamps reported by the ground truth file. This is one of the files produced by DHALSIM. However, the labeling may not be accurate: there may be *side-operations* not related to the attack, which may take a while to start too. So, we have to little rely on the found detection time and see it as an *indicative approximation* useful for comparing the various detectors included in the DIDS.

## 6.1 Dataset

We used DHALSIM virtual testbed to create the network traffic data and operational data. As explained in section 5.1, there are a total of two datasets, one is called Anytown, and the other is KY15. They refer to different ICS architectures, KY15 is *more complex* than Anytown.

Notice that all the attacks launched on the testbed for each topology contain different targets and triggering times to make the attacks more variable so the models can generalize better. Now we are going to discuss the data used by NIDS and PIDS.

**Network Dataset** The network traffic data consists of one PCAP file for each network device interface, except for the central router due to testbed limitations.

For the Anytown topology, the generated PCAP files are eight since there are three PLCs, four routers, and one SCADA device; while for the KY15 topology, the PCAP files are fourteen (six PLCs, seven routers, and one SCADA device). These files are generated for each simulated scenario inside every topology and later are converted by the Preprocessor module into JSON files to give them in input to the detectors. The Listing 6.1 shows an example of a JSON file preprocessed from a message inside a PCAP file. Notice that for the training and testing, we only used effectively the

PCAP/JSON of the PLCs inside the network.

Since we place each NIDS between each PLC and its respective switch, as we can see from Figure 6.15, we train and test each NIDS using the collected traffic related to its reference PLC. The unsupervised NIDSs, that are IAM and IAR, are all trained using the JSON network traffic data obtained from the normal execution lasted 500 iterations without attacks.

**Physical Process Dataset** The physical operational data consists of one SCADA CSV file with the values of all actuators and sensors at each iteration for each topology and scenario.

The unsupervised PIDSs, that are MinMax, GT, HM, and ST, are all trained with the operational data in JSON format obtained from the normal execution lasted 500 iterations without attacks.

In the Listing 6.2 there is the JSON preprocessed from one record of operational data. This latter contains also the integer part of a field device value, indicated with the "_int" suffix, since ST and HM work on a number space of a maximum of ten values. As we can see, the JSON fields are the same as the one described in paragraph 4.1.

**Listing 6.1:** Preprocessed JSON from a PCAP

```
{
    "id": 106,
    "timestamp":
        1688396107.286207,
    "protocol": "cip",
    "malicious": true,
    "src": "10.0.2.1:47214",
    "dest": "10.0.4.1:44818",
    "length": 42,
    "crc": false,
    "type": 76,
    "activity":"interrogate",
    "responds to": [],
    "data": {
        "T42": null
    }
}
```

**Listing 6.2:** Preprocessed JSON from a CSV

```
{
    "id": 21,
    "timestamp":
        1688403303.574755,
    "malicious": true,
    "P78": 0.0,
    "P78_int": 0,
    "P79": 0.0,
    "P79_int": 0,
    "T41": 8.809667587280273,
    "T41_int": 8,
    "T42": 8.165063858032227,
    "T42_int": 8
}
```

**Test Data** We know that the NIDSs test set always comprises JSON files converted from the PCAPs of network traffic, while the PIDSs and BLSTM test set includes JSON files converted from the CSV file of operational data. Both operational data and network traffic inside the test set are obtained from:

- The normal execution lasted 50, 80, and 200 iterations without attacks.
- The Execution lasted 50 iterations in which the DoS attack starts at iteration 14 until iteration 36.
- The Execution lasted 50 iterations in which the MITM attack starts at iteration 15 until iteration 46.

- The Execution lasted 50 iterations in which the NITM attack starts at iteration 11 until iteration 38.
- The Execution lasted 50 iterations in which the CITM attack starts at iteration 14 until iteration 38.
- The Execution lasted 50 iterations in which the SITM attack starts at iteration 13 until iteration 38.
- The Execution lasted 50 iterations in which the device attack starts at iteration 3 until iteration 30.

**Attack Classifier Dataset** The attack classifier used, that is RF, takes in input the preprocessed statistical network traffic features inside a CSV format file to discriminate the attack looking at the network traffic. As discussed in the subsection 5.4.1, these features are extracted using CICFlowMeter. Indeed, it can extract more than 80 features, that later are cleaned and preprocessed using a standard ML Pipeline proposed by the tool scikit-learn [36].

Since the goal of the thesis is to prove the efficiency of DIDS even when there are few attack samples, we produce a specific type of dataset focused on this kind of attack detection. However, at the same time, from the point of view of the Attack Classification task, the dataset generated contains a lot more normal samples than attack samples. This causes that, for the Attack Classification task, there is a high-class imbalance that we try to solve using the SMOTE data augmentation method.

The training set contains 957036 samples, while the test set size is 64674, both with mixed MITM, DoS, device attacks, and normal samples. Since CITM, SITM, NITM and MITM at the end are all MITM attacks, only for the Attack Classification task we decided to fuse these attacks into one single acronym, that is MITM, and consider just this label during the training and testing of the model. Indeed, we are interested in understanding that a MITM is occurring, and not necessarily which specific implementation of MITM is in progress.

## 6.2 Preliminary Data Analysis

The cyber-attacks described in section 5.2 cause some “anomalies” inside the network and in the processes themself. We now see some of these strange behaviors on the Anytown infrastructure.

At the process level, we can see in Figure 6.1 for example, the application of the device attack on the P79 actuator. P79 is a pump, and it was closed earlier than expected so the control rule is not respected. This can cause the level of water inside a tank to be lower than the desired value. In a real scenario, a device attack can be a huge economic loss considering for example a big Water Distribution System, because it can risk not having enough water to distribute.

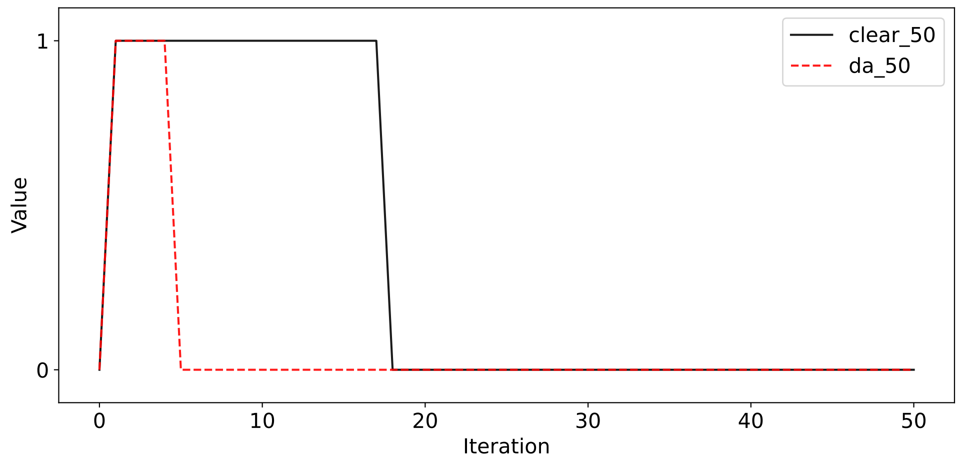

**Figure 6.1:** P79 actuator values during ICS normal operation (black) and under device attack (red).

Figure 6.2 shows the NITM attacking the PLC3 directly connected to the T42 sensor. The sensor value rises to ten at a certain iteration and stays there for a certain amount of time. After the attack, the values are a bit decreased with respect to the ones of the ICS during normal operation.

The NITM can change the payload of all the CIP packets directed towards the other control devices, differently from the standard MITM, which only modifies the payload of specific CIP packets directed towards other control devices. The attacker is placed in the middle between the PLC3 and the related switch connected to it, so he/she can sniff and spoof the messages.

The attacker's goal is to confuse the control devices making them believe that all the tanks connected to PLC3 (in this case only T42) reach their maximum water level. In this way, the control rules will lower the water level of the tanks by closing the pumps. The consequence is that the control rules are broken and the real water level becomes lower than the desired one, as we can see after the attack execution in Figure 6.2. As said before, in a real scenario it can be a huge economic loss.

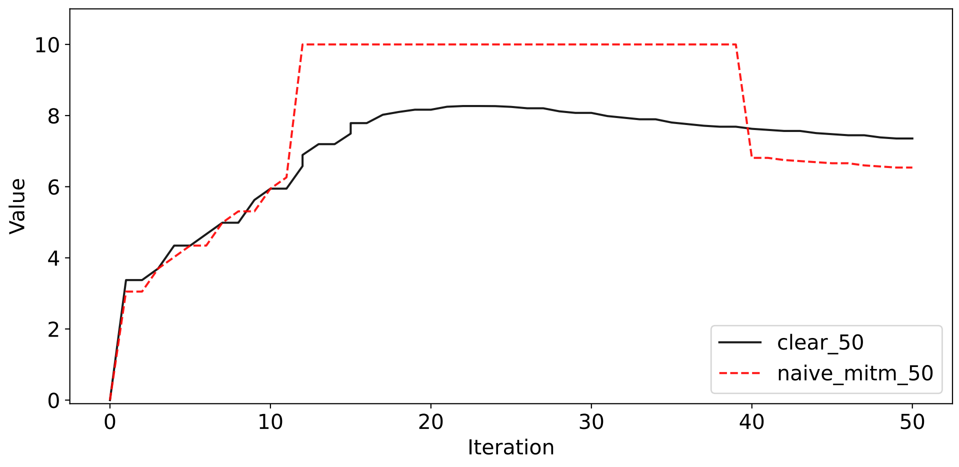

**Figure 6.2:** T42 sensor values during ICS normal operation (black) and under NITM attack (red).

Figure 6.3 shows the MITM in action on the T41 sensor. The sensor value drops suddenly to zero at a certain iteration and stays there for a certain amount of time. After the attack, the values are a bit increased with respect to the ones of the ICS during normal operation.

The purpose of the attack is to confuse the control devices, that will believe the tank T42 water level is the minimum. In this way, the control rules will increase it, putting the real water level above the normal one, as visible in Figure 6.3 after the attack is completed. This can cause serious problems, like a water flood if there is so much water inside the tank that exceeds the limit and if there are no devices, like our DIDS, able to prevent it. In a real scenario, it can be a great economic loss.

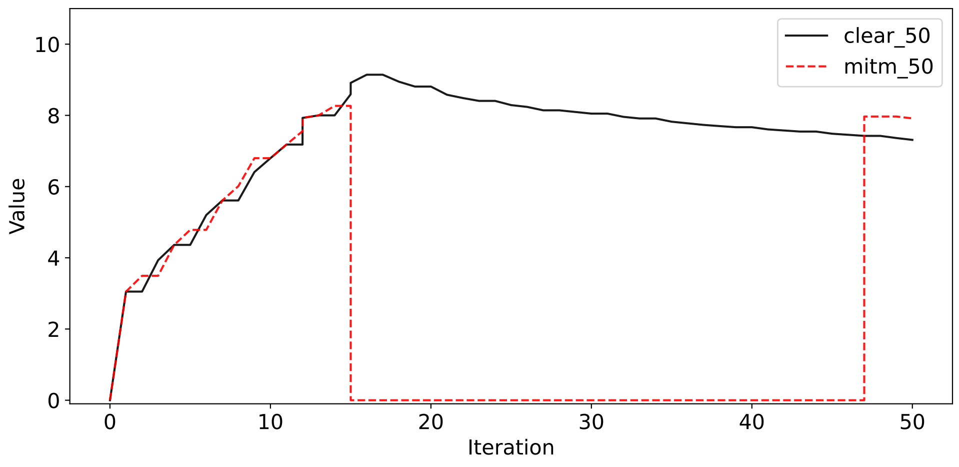

**Figure 6.3:** T41 sensor values during ICS normal operation (black) and under MITM attack (red).

The Figure 6.4 shows the SITM attacking the T41 sensor. The effects are almost equal to the previous MITM attack. Indeed, as before, the sensor value drops to zero at a certain iteration and stays there for a certain amount of time. After the attack, the values are a bit increased with respect to the ones of the ICS during normal operation.

The SITM causes the attacker to launch a CIP server and then serve the PLC connected to T41 using that server. It will create a new TCP connection and ENIP session between the attacker and the victim. In this way, the attacker can respond to the CIP requests coming from the PLC, sniffing and spoofing messages. The consequences are the same as the previous MITM attack.

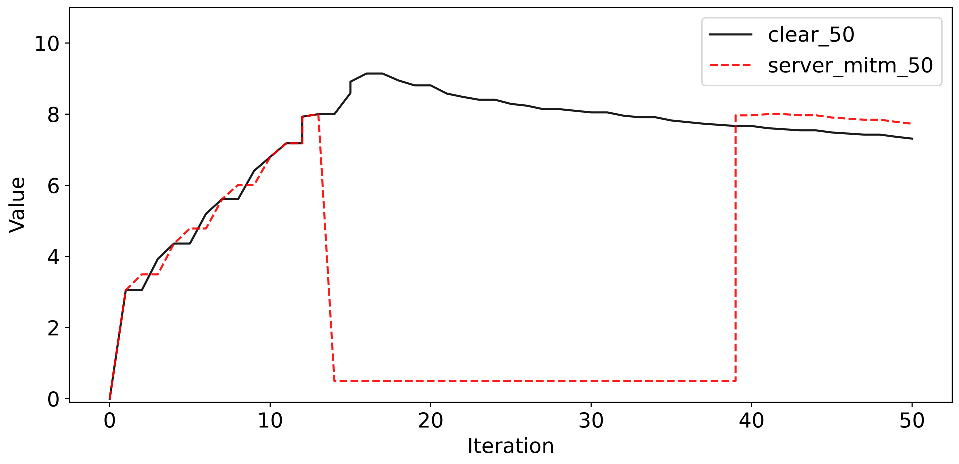

**Figure 6.4:** T41 sensor values during ICS normal operation (black) and under SITM attack (red).

In Figure 6.5 instead, we can see the application of the CITM on the T41 sensor. The sensor value drops gradually to zero at a certain iteration and stays there for a short amount of time. After the attack, the values return to be equal to the ones of the ICS during normal operation, differently from the previous MITM attacks. Probably, it depends on the short attack duration which doesn't destabilize too much the sensor value.

In Concealment Man-in-the-Middle (CITM), for a short time, the attacker will differentiate between traffic with a specific PLC as a destination and traffic with a SCADA server as a destination. If the destination is a PLC, the attacker will modify the packet payload field values with the configured attack values. If the destination is a SCADA server, the attacker will modify the CIP packet payload field values with the configured concealment values. In this way, the attacker can remain undetected for a bit of time.

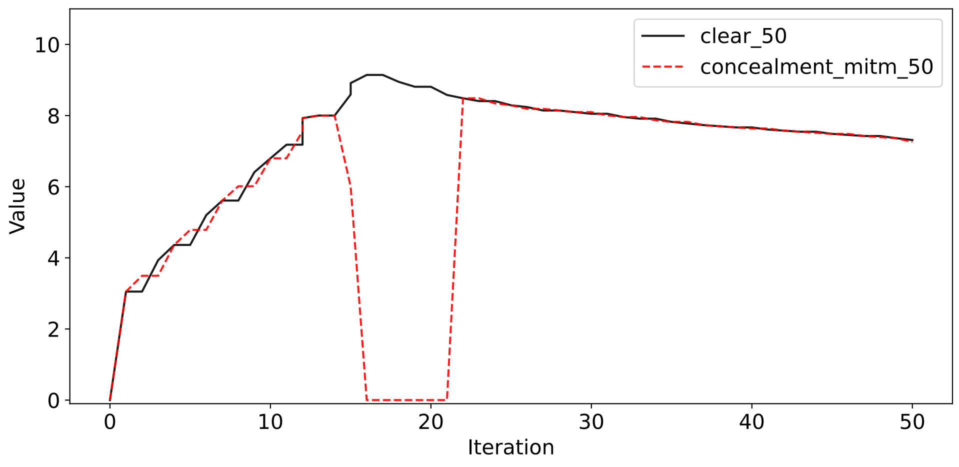

**Figure 6.5:** T41 sensor values during ICS normal operation (black) and under CITM attack (red).

Sometimes the attacks don't show a concrete effect on the physical process data, but instead are more "visible" on the network, or vice versa. That's the case of the DoS attack. In Figure 6.6 we can see the absence of repercussions on the T42 sensor during a DoS on the PLC3, directly connected to T42.

The Denial of Service (DoS) attack used by the attacker interrupts the flow of CIP messages containing data between PLCs. This attack first performs an ARP Spoofing attack into the target and then stops forwarding the CIP messages. This will cause the PLCs to be unable to update their cache with new system state information, possibly taking wrong control action decisions that could lead to wrong actions on the water level inside the tanks. It can cause a substantial economic loss in a real scenario.

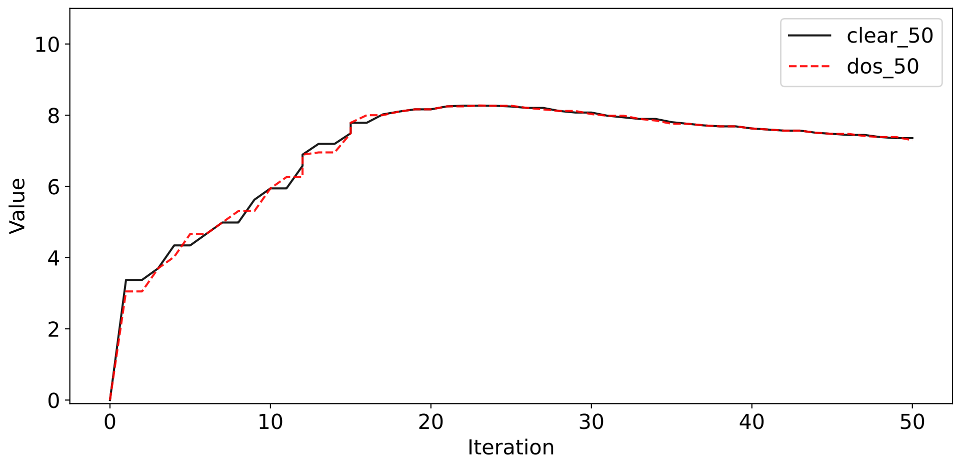

**Figure 6.6:** T42 sensor values during ICS normal operation (black) and under DoS attack (red).

Discussing about the network side, Figures from 6.7 to 6.14 show the number of packets in a time window of 5 seconds detected inside PLC1, PLC2 and PLC3 under normal ICS operation, NITM, MITM, CITM, SITM and DoS attack. As we can see, the number of packets is really variable. However, during an attack at a certain timestamp, it becomes lower than the number of packets during normal operation. This happens since the MITM attacks slow the network speed, so the packets are sent and received slowly, while DoS attack performs an ARP Spoofing attack into the target that stops forwarding the CIP messages, and this causes that fewer packets are effectively received and then sent.

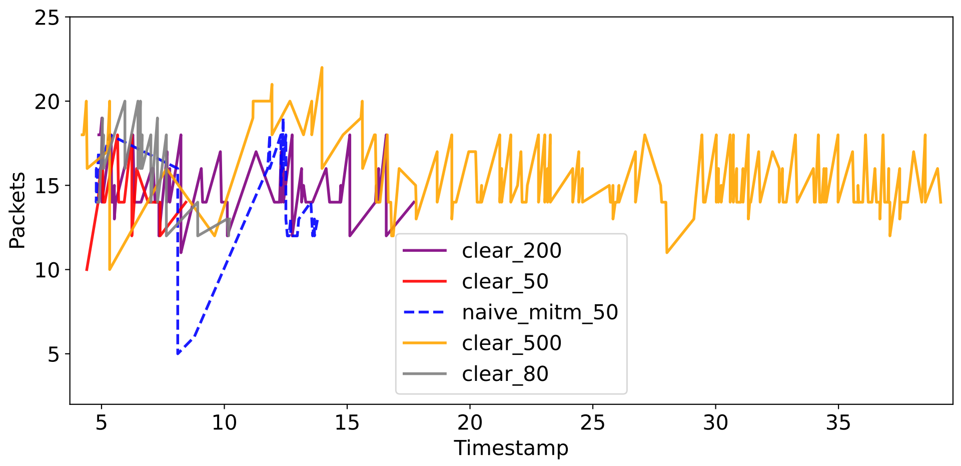

**Figure 6.7:** PLC1 number of packets in a time window of 5 seconds during ICS normal operation and under NITM attack (blue).

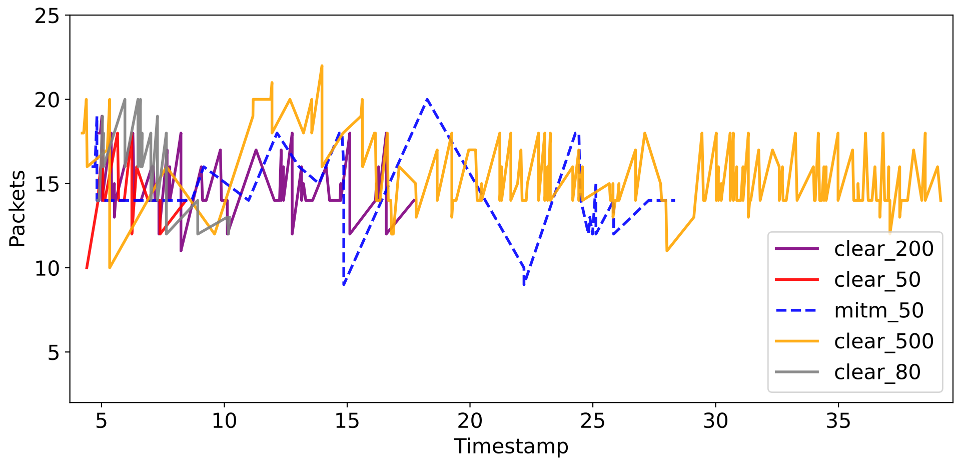

**Figure 6.8:** PLC1 number of packets in a time window of 5 seconds during ICS normal operation and under MITM attack (blue).

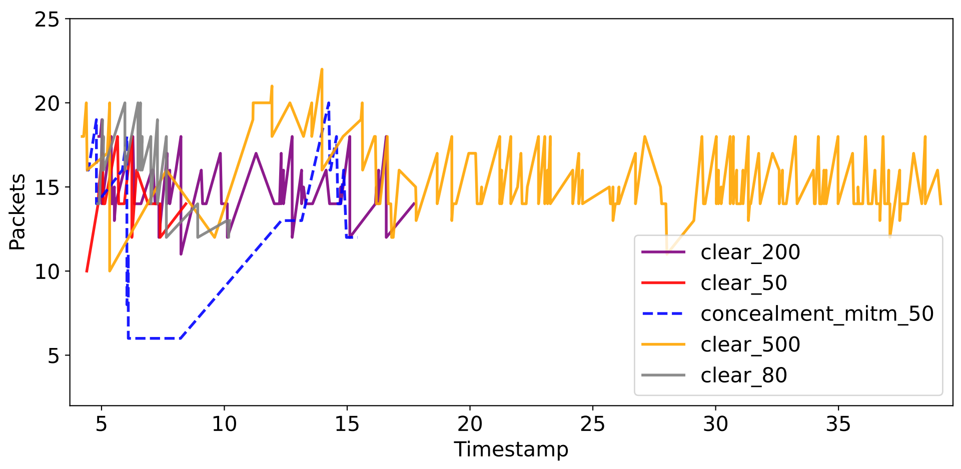

**Figure 6.9:** PLC1 number of packets in a time window of 5 seconds during ICS normal operation and under CITM attack (blue).

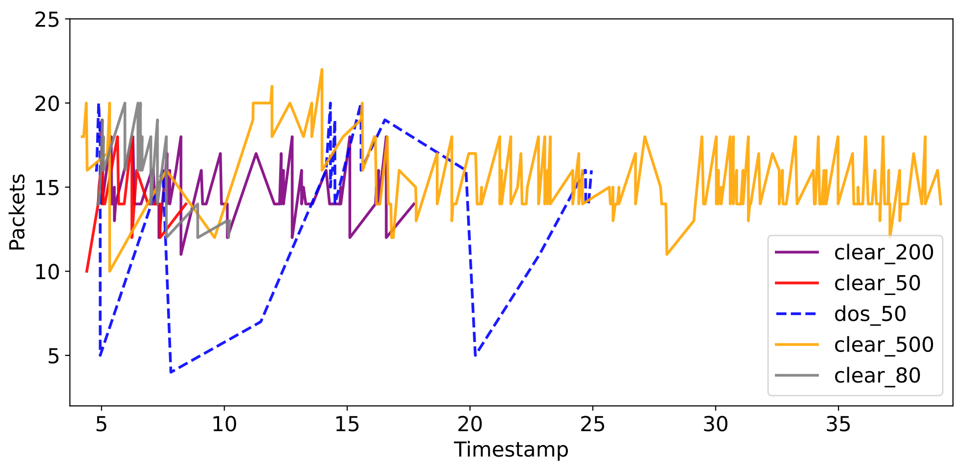

**Figure 6.10:** PLC1 number of packets in a time window of 5 seconds during ICS normal operation and under DoS attack (blue).

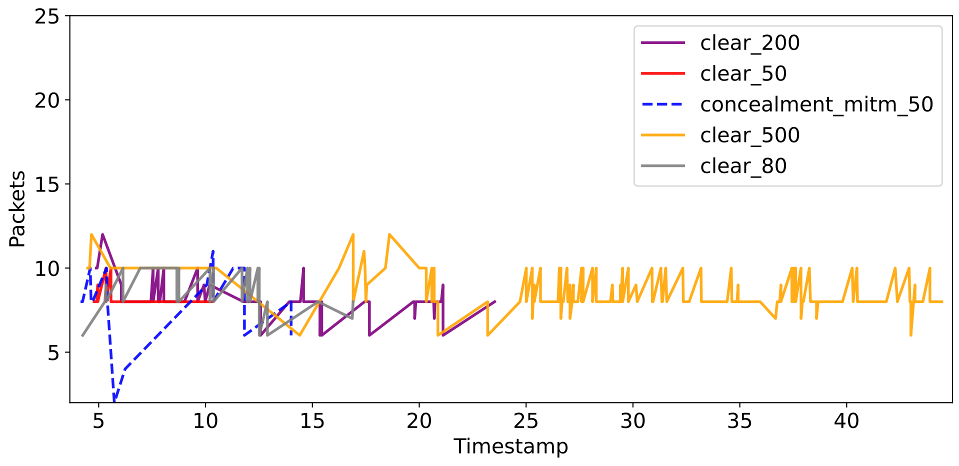

**Figure 6.11:** PLC2 number of packets in a time window of 5 seconds during ICS normal operation and under CITM attack (blue).

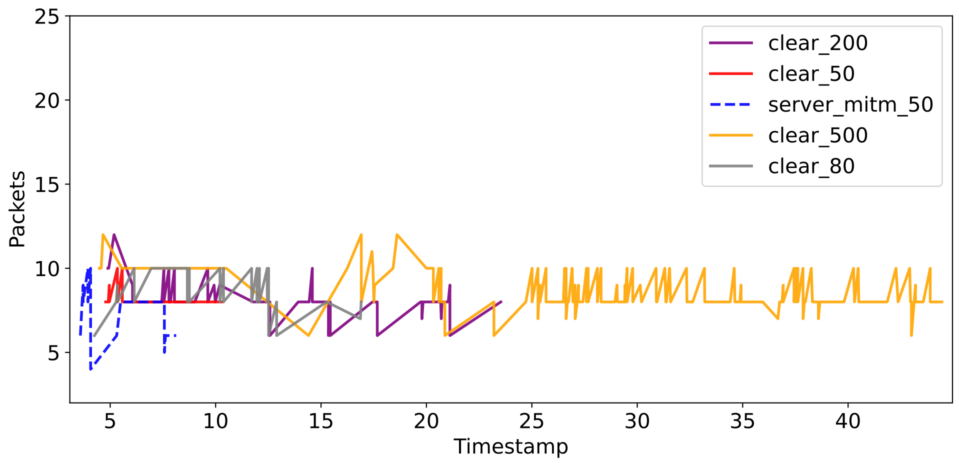

**Figure 6.12:** PLC2 number of packets in a time window of 5 seconds during ICS normal operation and under SITM attack (blue).

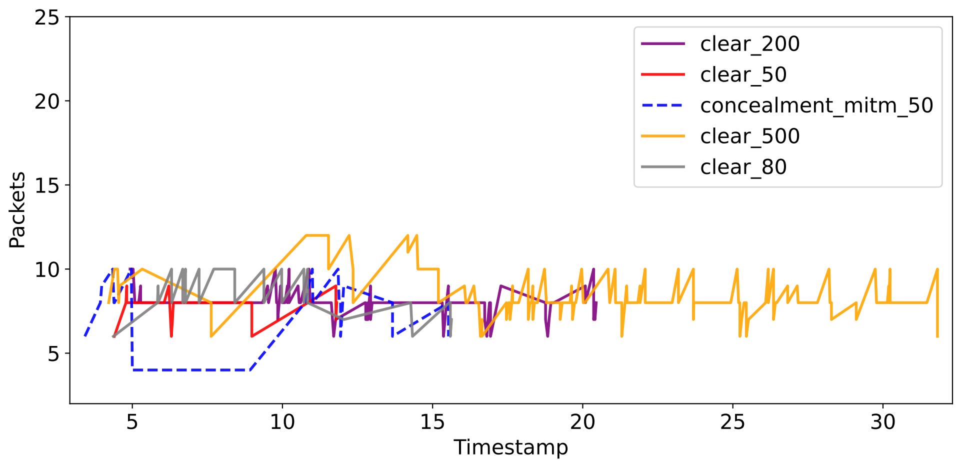

**Figure 6.13:** PLC3 number of packets in a time window of 5 seconds during ICS normal operation and under CITM attack (blue).

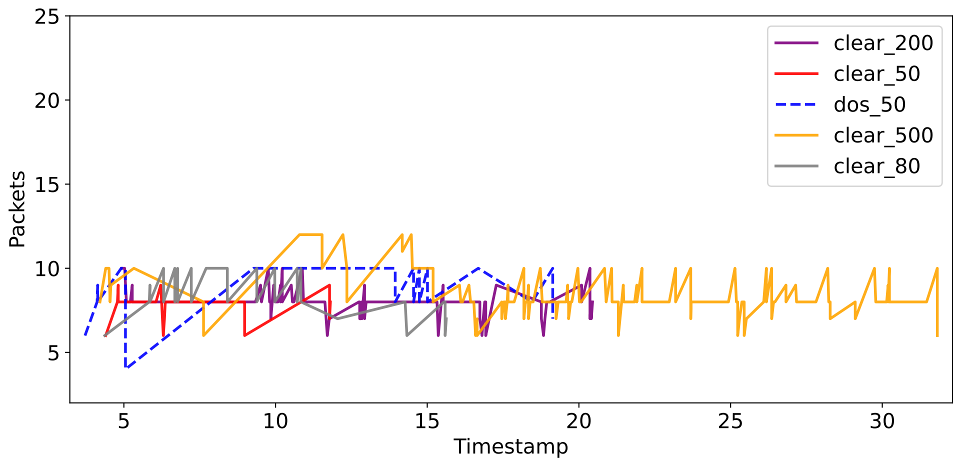

**Figure 6.14:** PLC3 number of packets in a time window of 5 seconds during ICS normal operation and under DoS attack (blue).

We can conclude from this analysis that some attacks, like the device attack, impact more the physical level than the network level, while other attacks, like DoS, impact more the network level than the physical level. This strengthens our thesis: a DIDS that considers both the data sources is better than an IIDS only specialized in the detection of the attacks using one single source of data.

## 6.3 Experimental Setup

The DIDS and the Attack Classifier module need to be set up in a certain way to work properly on our specific environment and dataset. So, to find the optimal parameters for our DIDS, we perform parameter tuning using a particular type of grid search for each detector inside the DIDS.

**DIDS Grid Seach** The grid search performed on each detector contained in the DIDS tries to maximize the number of correctly classified scenarios. At the same time, this grid search considers only the parameter settings that correctly do not classify as under attack at least two out of three normal testbed executions not under attack. This is done because of the low number of normal scenarios, indeed three out of nine scenarios are not under attack inside each dataset.

Suppose not using the condition on the number of scenarios with no attacks correctly ignored. The absence of this condition makes a dumb detector that always finds the presence of an attack, even if there aren't any, have better performance than another detector that correctly doesn't identify attacks during normal scenarios. However, it detects fewer attacks than the previous detector. So, to not have dumb detectors inside the DIDS, we insert these two conditions inside our grid search.

**Network Detectors Configuration** One issue that we addressed is the following: Where do the NIDSs need to be placed inside the ICS network? As we can see from Figure 6.15, we decided to place the NIDSs between the PLCs and the switches, to get all the packets directed towards and from them. Indeed, by testing each network detector on different positions, we found that more attacks are detected when the DIDSs are placed between the switches and the PLCs with respect to when they are placed between other devices, like between the switches and the routers or between the switch and the SCADA device.

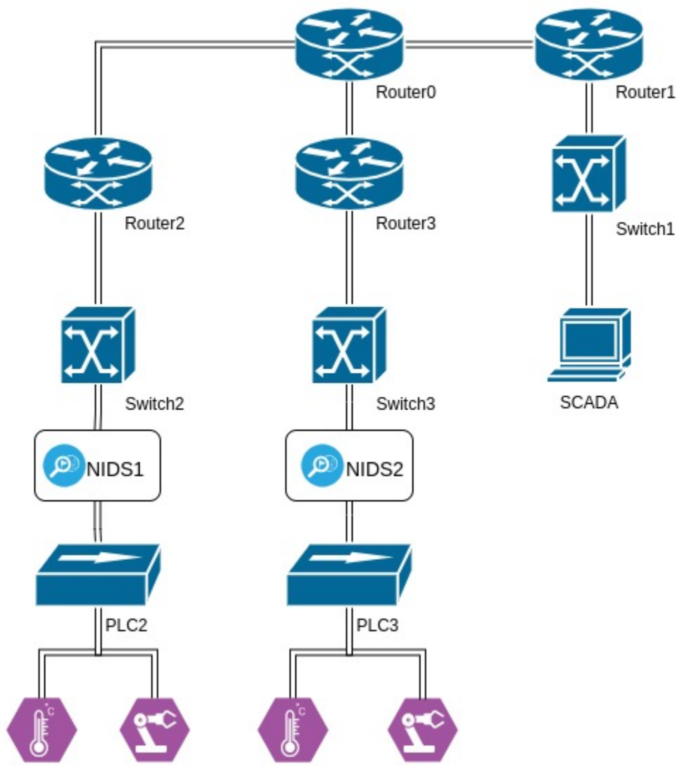

**Figure 6.15:** NIDSs location inside the ICS network.

The parameters selected and discovered using grid search on the NIDSs are summarized inside Table 6.1 grouped for the dataset, IIDS and, when necessary, for PLC. There are two tunable parameters in IAM and IAR, sample/window size (W) and threshold level (N).

**Physical Detectors Configuration** The parameters selected and discovered using grid search on the PIDSs instead are summarized inside Table 6.2 grouped for the dataset and IIDS. The PIDSs have one tunable parameter, that is the Error Margin (EM), with an exception for the HM detector, which uses the window size (WH) too.

**Combiner Configuration** The combiner, described in the subsection 5.3.4, uses the weights manually assigned to each detector based on the relevance of their detection, to decide whether there is an attack or not. These weights are normalized to be between zero and one, and they can be seen in Table 6.4. Each weight is considered inside the summation of the weights only if the respective IIDS detects an attack, that is when this latter detects at least one malicious record/message. Then, the summation is compared with a threshold. If it is higher than the threshold, the combiner launches an alarm.

We found that using 0.7 as the threshold for Anytown, while 1.1 for KY15 is a good choice that can avoid launching false alarms in the presence of noise, at the same time detecting all the proposed attacks.

**Table 6.1:** Parameters selected for IAM and IAR on all the PLCs and datasets.

| Dataset → | Anytown | | KY15 | |
|---|---|---|---|---|
| IDS / Params. | IAM | IAR | IAM | IAR |
| PLC1-W | 26 | 20 | 8 | 100 |
| PLC1-N | 1 | 2 | 19 | 17 |
| PLC2-W | 3 | 25 | 8 | 130 |
| PLC2-N | 2 | 2 | 19 | 15 |
| PLC3-W | 5 | 40 | 8 | 100 |
| PLC3-N | 2 | 3 | 36 | 60 |
| PLC4-W | - | - | 8 | 130 |
| PLC4-N | - | - | 19 | 15 |
| PLC5-W | - | - | 8 | 8 |
| PLC5-N | - | - | 19 | 12 |
| PLC6-W | - | - | 8 | 90 |
| PLC6-N | - | - | 19 | 15 |

**Table 6.2:** Parameters selected for the Physical detectors on both datasets

| IDS / Params. | MinMax | ST | GT | HM |
|---|---|---|---|---|
| EM | 0.4 | 0.5 | 0.06 | 9.0 |
| WH | - | - | - | 21 |

**Table 6.3:** Parameters selected for RF model on both the datasets

| NoE | Criterion | MS |
|---|---|---|
| 76 | gini | 2 |

**Table 6.4:** Weights selected for the Combiner of the PIDSs and NIDSs for each dataset

| IAM | IAR | MinMax | ST | GT | HM |
|---|---|---|---|---|---|
| 0.14 | 0.14 | 0.58 | 0.29 | 0.44 | 0.57 |

**Attack Classifier Configuration** The optimal configuration for the Attack Classifier used, that is the RF, is found performing parameter tuning with grid search that maximizes the Macro-averaged F1 score (6.9). Indeed, since the dataset used for Attack Classification is imbalanced, the accuracy is not a reliable statistic in this case. At the same time, the Macro-averaged F1 score considers both the Precision and Recall statistics of each class, independently by its support.

Table 6.3 shows the parameters found thanks to the grid search on the RF model. Between the parameters used for RF classifier, the most relevant, and the ones that are tuned in our grid search, are the Number of Estimators (NoE), Criterion, and

Minimum number of samples required to Split a node (MS).

## 6.4 Evaluation Metrics

Classic metrics, like f1 score, precision, and so on, are not exhaustive to represent the results when they have to deal with our DIDS. This is because we want to identify whether the attack was detected or not, how late, and how many normal executions are falsely detected as attacks.

Regards the Attack Classifier component, we used all the standard metrics usually used to define its performance. However, due to the imbalanced dataset, we found some metrics more useful than others. In particular, the most useful metrics are Precision, Recall, F1 score, Macro-averaged Precision, Macro-averaged Recall, and Macro-averaged F1 score. Indeed, these last metrics evaluate the performance of the RF model considering that some classes have a low support inside the dataset.

For this reason, we propose new metrics for evaluating the PIDSs and NIDSs and use the standard classification metrics to show the results of the RF model. All the metrics are detailed in the following paragraph.

**Attack Detector Metrics** The following metrics are used to evaluate the attack detector modules. In general, we are interested in understanding if an attack is detected instead of how many malicious instances (e.g., packets or sensor measurements) are identified. This is because we believe that the primary goal of a detector is to flag ongoing attacks instead of classifying process instances:

- True Detection (TD) measures how many attack scenarios are correctly detected.
- False Detection (FD) measures how many normal scenarios are identified as attack scenarios.
- True NOrmal (TNO) measures how many normal scenarios are correctly not detected as an attack.
- False NOrmal (FSO) measures how many attack scenarios are not detected.
- Detection Delay (DD) measures the attack detection delay in terms of iteration. It may be positive or negative if it is detected after or before the first "true" label assigned by the labeling process. In section 5.1 we saw that an iteration for the Anytown topology is more or less 3.65 seconds, while for the KY15 topology is about 18.18 seconds, so DD can be used to measure approximately the detection time.
- Mean Detection Delay (MDD) measures the mean DD of all attack scenarios in a dataset.
- Standard deviation of the Detection Delays (SDD) measures the DD standard deviation of all attack scenarios in a dataset.
- Best Detection Delay (BDD) refers to the best DD found between all attack scenarios in a dataset.
- Worst Detection Delay (WDD) refers to the worst DD found between all attack scenarios in a dataset.

**Table 6.5:** Parameters used in the metrics.

| Classification | Description |
|---|---|
| True Positive$_{class}$ (TP$_{class}$) | Number of class samples predicted correctly |
| True Positive$_{class}$ (TN$_{class}$) | Number of non-class samples predicted correctly |
| False Positive$_{class}$ (FP$_{class}$) | Number of non-class samples predicted wrongly |
| False Negative$_{class}$ (FN$_{class}$) | Number of class samples predicted wrongly |

**Attack Identifier Metrics** When we are going to present the Attack Classification results in section 6.8, we use different standard metrics to evaluate the performance of the RF model. In this section, we briefly summarise them. For the sake of completeness, Table 6.5 clarifies the terms used in the metrics. When these measures are applied to a specific class, we use the subscript style to specify the class, e.g. $TP_{MITM}$ to indicate the number of MITMs attack samples predicted correctly.

· Accuracy: the total number of class and non-class samples that have been classified correctly. This metric is the same for all the classes.

$$Accuracy_{class} = \frac{TP_{class} + TN_{class}}{TP_{class} + TN_{class} + FP_{class} + FN_{class}}. \tag{6.1}$$

· Precision: represents the ratio of class samples correctly classified among the total number of class samples.

$$Precision_{class} = \frac{TP_{class}}{TP_{class} + FP_{class}}. \tag{6.2}$$

· Recall: also known as sensitivity, represents the ratio of class samples correctly identified among the total number of class samples. Both Precision and Recall are therefore based on relevance.

$$Recall_{class} = \frac{TP_{class}}{TP_{class} + FN_{class}}. \tag{6.3}$$

· F1 Score: the harmonic mean of the Precision and Recall.

$$F1_{class} = \frac{2 \cdot Precision_{class} \cdot Recall_{class}}{Precision_{class} + Recall_{class}}. \tag{6.4}$$

· Macro-averaged Precision: calculated as an average of Precisions of all classes. So, all classes equally contribute to the final averaged metric:

$$\frac{\sum_{j=1}^{k} Precisions_j}{k}. \tag{6.5}$$

· Weighted-averaged Precision: it is also calculated based on Precision per class but takes into account the number of samples of each class, indicated as $N_j$ for class $j$, in the data. So, each class's contribution to the average is weighted by its size:

$$\frac{\sum_{j=1}^{k} N_j * Precisions_j}{\sum_{j=1}^{k} N_j}. \tag{6.6}$$

· Macro-averaged Recall: as before, but considering the Recall:

$$\frac{\sum_{j=1}^{k} Recall_j}{k}. \tag{6.7}$$

· Weighted-averaged Recall: as before, but considering the Recall:

$$\frac{\sum_{j=1}^{k} N_j * Recall_j}{\sum_{j=1}^{k} N_j}. \tag{6.8}$$

· Macro-averaged F1 score: as before, but considering the F1 score:

$$\frac{\sum_{j=1}^{k} F1_j}{k}. \tag{6.9}$$

· Weighted-averaged F1 score: as before, but considering the F1 score:

$$\frac{\sum_{j=1}^{k} N_j * F1_j}{\sum_{j=1}^{k} N_j}. \tag{6.10}$$

## 6.5 Network Detector Results

To have a better comparison between our Physical and Network detectors, we start to analyze the results of each technique and then see the results of the combination of the two detectors, showing the correctly identified and not identified attacks on the datasets by each detector using the metrics explained in section 6.4. Tables 6.6 and 6.7 show IAM and IAR results on Anytown and KY15 topologies respectively.

**Anytown** Both IAM and IAR have really good results on Anytown, with at most one FSO or FD, as visible in Table 6.6. Clearly, the most difficult attack to detect is device one, since the attacker physically tampers an actuator, not producing too much anomalous traffic. However, IAR and IAM, when configured on specific PLCs, seem able to detect also that attack. When IAM is placed between PLC2 and its related switch, it has the maximum TDs and TNOs.

**KY15** On KY15 topology instead, the TDs are lower than Anytown, as visible in Table 6.7. NITM and Device attacks are the most difficult to detect: just one IAR can detect one of these attacks, and one another IAR placed in a different position can detect the other attack. Intuitively, DoS is detected, jointly with MITM. However, all the detectors have on average three TDs, with the IAR placed before PLC5 having four TDs at most, even if it has one FD.

The normal traffic with 50 iterations is FD by five detectors, as visible in Table 6.7. In general, all the detectors when applied on the KY15 topology are less accurate, maybe because it is a more complex topology than Anytown and so the anomalous network traffic is more noisy and concentrated in a few zones with respect to Anytown architecture. Indeed, if an attack targets a specific PLC or field device, most of the consequences on the network are visible just by the subnetwork of the device targeted by the attack, and so by the specific NIDS placed in that subnetwork.

At first sight, it seems that the inter-arrivals have better performance on the Anytown topology than the KY15 topology.

**Table 6.6:** Correctly detected attacks and ignored normal executions by IAM and IAR applied on each PLC inside the Anytown topology. The prefix "C" stays for a normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation.

| DS / IDS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| PLC1-IAM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | 5/6 |
| PLC1-IAR | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| PLC2-IAM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| PLC2-IAR | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| PLC3-IAM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | 5/6 |
| PLC3-IAR | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | 5/6 |

**Table 6.7:** Correctly detected attacks and ignored normal executions by IAM and IAR applied on each PLC inside the KY15 topology. The prefix "C" stays for a normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation.

| DS / IDS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| PLC1-IAM | ✓ | ✓ | ✓ | ✓ | x | x | x | ✓ | x | 2/6 |
| PLC1-IAR | x | ✓ | ✓ | ✓ | x | x | x | x | x | 1/6 |
| PLC2-IAM | ✓ | ✓ | ✓ | ✓ | x | ✓ | x | ✓ | x | 3/6 |
| PLC2-IAR | x | ✓ | ✓ | ✓ | x | ✓ | x | x | x | 2/6 |
| PLC3-IAM | ✓ | ✓ | ✓ | x | x | x | x | ✓ | x | 1/6 |
| PLC3-IAR | ✓ | x | ✓ | x | x | ✓ | x | ✓ | x | 2/6 |
| PLC4-IAM | ✓ | ✓ | ✓ | ✓ | x | x | x | ✓ | x | 2/6 |
| PLC4-IAR | x | ✓ | ✓ | ✓ | ✓ | x | x | x | x | 2/6 |
| PLC5-IAM | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | x | 3/6 |
| PLC5-IAR | x | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | 4/6 |
| PLC6-IAM | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | x | 3/6 |
| PLC6-IAR | x | ✓ | ✓ | ✓ | x | x | ✓ | x | x | 2/6 |

**Detection time** Tables 6.8 and 6.9 contain the Mean Detection Delay (MDD), Standard deviation of the Detection Delays (SDD), Best Detection Delay (BDD) and Worst Detection Delay (WDD) for each NIDS and PLC on Anytown and KY15 topologies, respectively.

In the case of Anytown, the WDD is 46 when IAR is mounted on PLC2, the BDD instead is -5 obtained by various NIDSs. In general, the worst and best MDD is 8, by IAR on PLC3, and -2.4, by IAM on PLC3, as visible by Tables 6.8, respectively. The SDD is between 1.5 and 17.56. Some attacks are detected before they are launched. This happens because some attacks, especially MITM, do some side operations on the network during a pre-attacking phase with variable duration according to the topology, and these are captured by our detectors.

**Table 6.8:** IAM and IAR DD metrics for each PLC on Anytown topology.

| DS / IDS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| PLC1-IAM | 12 | 15.39 | -1 | 35 |
| PLC1-IAR | 1.83 | 10.02 | -5 | 24 |
| PLC2-IAM | 3.67 | 13.63 | -5 | 34 |
| PLC2-IAR | 6.83 | 17.56 | -3 | 46 |
| PLC3-IAM | -2.4 | 1.5 | -5 | -1 |
| PLC3-IAR | 8 | 1.67 | 5 | 10 |

**Table 6.9:** IAM and IAR DD metrics for each PLC on KY15.

| DS / IDS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| PLC1-IAM | -6.5 | 8.5 | -15 | 2 |
| PLC1-IAR | -15 | 0 | -15 | -15 |
| PLC2-IAM | -4.67 | 7.41 | -15 | 2 |
| PLC2-IAR | -8 | 7 | -15 | -1 |
| PLC3-IAM | 19 | 0 | 19 | 19 |
| PLC3-IAR | 26 | 4 | 22 | 30 |
| PLC4-IAM | -6.5 | 8.5 | -15 | 2 |
| PLC4-IAR | 4 | 19.0 | -15 | 23 |
| PLC5-IAM | -3.67 | 8.01 | -15 | 2 |
| PLC5-IAR | 4 | 16.32 | -15 | 30 |
| PLC6-IAM | -3.67 | 8.01 | -15 | 2 |
| PLC6-IAR | -6.5 | 8.5 | -15 | 2 |

Concearning KY15 topology, Table 6.9 shows DD statistics. In terms of iterations, KY15 BDD and WDD are -15 and 30, respectively. The best and worst mean Mean Detection Delay (MDD) are -8 and 26, while the SDD is variable between 0 and 19 in general. The fastest detector seems to be IAR on PLC2. Notice that Table 6.9 shows the KY15 BDD is pretty much the same for almost all the NIDSs: -15. This is the DD related to the MITM attack, and we suppose that it is detected before it is launched because of the side operations on the network during the pre-attack phase, that don't pass unnoticed by the network detectors.

## 6.6 Phyisical Detector Results

Regarding the Physical detectors, Table 6.10 and 6.11 shows the PIDSs results on Anytown and KY15 topologies respectively, applying the model parameters described in section 6.3. At first glance, it seems that KY15 and Anytown are very similar in terms of results.

**Anytown** Specifically, Table 6.10 regards Anytown topology and shows that in this case the most difficult attack to detect isn't only the Device one, but the DoS attack too, since DoS operates more at the network traffic level, while a physical detector can only see the consequences. The best detector seems to be ST, even if it has one FD.

**KY15** The results on the KY15 topology visible in Table 6.11 show that ST becomes the worst detector, followed by HM. MinMax and GT instead have the maximum TD and TNO. The most detected attacks are the Device one and SITM.

In general, the PIDS detectors detect on average 4/6 attacks considering both Anytown and KY15 topologies.

**Table 6.10:** Correctly detected attacks and ignored normal executions by Physical detectors on Anytown topology. The prefix "C" stays for a normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation.

| IDS \ DS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| MinMax | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | 4/6 |
| ST | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| GT | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | 4/6 |
| HM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | 4/6 |

**Table 6.11:** Correctly detected attacks and ignored normal executions by Physical detectors on KY15 topology. The prefix "C" stays for normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation.

| IDS \ DS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| MinMax | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| ST | ✓ | ✓ | ✓ | ✓ | x | ✓ | x | x | ✓ | 3/6 |
| GT | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| HM | ✓ | ✓ | ✓ | x | x | ✓ | x | x | ✓ | 4/6 |

**Table 6.12:** PIDSs DD metrics on Anytown.

| IDS \ DS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| MinMax | 0 | 0.71 | -1 | 0 |
| ST | 14 | 14.52 | 0 | 34 |
| GT | -0.25 | 0.43 | -1 | 0 |
| HM | 0 | 0.71 | -1 | 1 |

**Table 6.13:** PIDSs DD metrics on KY15.

| IDS \ DS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| MinMax | -0.33 | 0.47 | -1 | 0 |
| ST | -0.33 | 0.47 | -1 | 0 |
| GT | 7.83 | 10.64 | -1 | 26 |
| HM | -0.33 | 0.47 | -1 | 0 |

**Detection Time** Tables 6.12 and 6.13 contain the MDD, SDD, BDD and WDD for each PIDS on Anytown and KY15 topologies, respectively.

ST in a case is the slowest PIDS on Anytown topology, requiring 34 DD as visible by Table 6.12. Except ST, all the other detectors on Anytown show a similar DD metrics, with an MDD and SDD around 0. On KY15 instead, they show the same detection time statistics except for GT, which in one case detects an attack after 26 iterations, as visible by Table 6.13

## 6.7 DIDS vs BLSTM Results

To make a better analysis of the DIDS results, we compare them with the ones obtained from the implementation of BLSTM detector implemented by Wolsing et al. [31]. We chose BLSTM since, by what Wolsing et al. [5] report in their paper, BLSTM is one of the detectors with the best performance on the dataset they used. Wolsing et al. [31] implementation of BLSTM uses the components offered by the TensorFlow library [49].

Since BLSTM is a supervised model, it has been trained on a mix of normal simulator executions without and with attacks. These latters were launched two times during the simulation, producing a total of twelve attacks. This dataset generation process is applied for both KY15 and Anytown topologies. Notice that BLSTM takes in input the preprocessed physical data of the physical processes obtained by the SCADA device, and so it is considered a PIDS.

Moving to discuss the DIDS, combining all the previous results, the final DIDS is able to reach the maximum TD and the maximum TNO. Indeed, assigning the weights to all the IIDS with the parameters discussed in section 6.3, using the threshold 0.7 for Anytown and 1.1 for KY15, Tables 6.14 and 6.15 show the scores assigned by the combiner doing the weighted sum of the detections for each scenario and the final results of DIDS and BLSTM.

We can see that on Anytown topology BLSTM has one FD and one FSO, while on KY15 topology BLSTM has two FSO. So, the TD is five and four for Anytown and KY15 topologies respectively.

We proved that DIDS overcame BLSTM, one of the most used DL models based on RNN in the IDS field. We also show that in general, all the single detectors work well. However, they reach the maximum performances when combined into a single detector, so, based on the results, we recommend using them jointly. Indeed, the IAR and IAM are really good in detecting DoS attacks and the standard MITM (Tables 6.6, 6.7), while the PIDSs have good performances in detecting all the types of MITM and the Device attack (Tables 6.10, 6.11).

**Table 6.14:** Correctly detected attacks and ignored normal executions by BLSTM and our DIDS on Anytown topology, and the scores assigned by combination algorithm. The prefix "C" stays for a normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation. The threshold is 0.7.

| DS / IDS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| Score | 0.0 | 0.59 | 0.0 | 2.78 | 2.78 | 2.78 | 2.78 | 0.88 | 0.74 | - |
| DIDS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| BLSTM | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | 5/6 |

**Table 6.15:** Correctly detected attacks and ignored normal executions by BLSTM and our DIDS on KY15 topology, and the scores assigned by combination algorithm. The prefix "C" stays for a normal dataset with no attacks, while the numerical suffix is the number of iterations used. All the attacks run on a 50-iteration simulation. The threshold is 1.1.

| DS / IDS | C50 | C80 | C200 | MITM | NITM | SITM | CITM | DoS | Dev | Atks |
|---|---|---|---|---|---|---|---|---|---|---|
| Score | 0.74 | 0.15 | 0.0 | 3.37 | 1.18 | 2.34 | 1.62 | 2.21 | 2.04 | - |
| DIDS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 6/6 |
| BLSTM | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | x | 4/6 |

However, even if they complement each other, we decided to give low values to the weights of IAR and IAM, as visible by Table 6.4. Indeed, as we can see in Tables 6.6 and especially 6.7, they have a total of eight FD when the number of iterations is equal to twenty and eighty. While, as visible by Tables 6.10 and 6.11, ST is the only PIDS with one FD on the Anytown topology. Also, the network detectors are many more, especially in KY15 topology, and so an error of one NIDS could cause the echo of all the others. For this reason, we decided to give more "decisional power" to the PIDSs, and low weights to the NIDSs.

However, why did the NIDSs have this great number of FD? These can be explained by the variable frequency of transmission time. Indeed, all the networks can be subject to latency, and in a real scenario noise disturbance too. So, it may happen that these factors confuse our network detectors, which are based on inter-arrival times.

**Table 6.16:** BLSTM and DIDS DD metrics on Anytown.

| DS / IDS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| DIDS middle-case | 1.83 | 10.02 | -5 | 24 |
| DIDS worst-case | 12 | 15.39 | -1 | 35 |
| BLSTM | 15.4 | 4.84 | 12 | 25 |

**Table 6.17:** BLSTM and DIDS DD metrics on KY15.

| DS / IDS | MDD | SDD | BDD | WDD |
|---|---|---|---|---|
| DIDS middle-case | -0.33 | 0.47 | -1 | 0 |
| DIDS worst-case | 26 | 4 | 22 | 39 |
| BLSTM | 9.5 | 8.44 | 3 | 24 |

**Detection Time** Tables 6.16 and 6.17 show all the DD metrics related to BLSTM and DIDS. Since the Combiner module needs to wait for all the detectors to finish their execution in order to compute the weighted sum of all the detections, we reported in both the Tables the DIDS worst-case related to the MDD metric and DIDS middle-case related to the MDD metric. On Anytown, it seems that DIDS MDD and BDD are better than BLSTM in all cases, while the WDD related to the worst-case is much higher than the WDD of BLSTM, as visible in Table 6.16. So, we can conclude that on Anytown topology, DIDS is faster than BLSTM the majority of times in detecting the attacks.

On KY15 instead, Table 6.17 shows that all the DD metrics of DIDS worst-case are much greater in comparison to the ones of the BLSTM. The DIDS middle-case instead, shows that DD metrics are better than BLSTM and the previous topology too. However, it is not sufficient. So, in this topology, we have that the fastest method in detecting the attacks is BLSTM the majority of times.

However, it is reasonable to think that BLSTM is faster than our DIDS in general. Indeed, our DIDS uses from 10 to 16 detectors in the two topologies analyzed, and it may require also more NIDSs in more complex topologies. While the BLSTM doesn't require other detectors than itself. This causes the probability of late detections by the DIDS to increase a lot with the increasing number of detectors, as visible in Anytown and KY15 topology. Indeed, on Anytown, the DIDS has a faster detection on average than BLSTM, while on KY15 topology this is not true.

## 6.8 Attack Classification Results

The attack classification module has the goal of identifying the specific attack occurring, if any. To do this, we implemented a RF model that takes in input statistical features of bidirectional flows inside the ICS network traffic. Now we are going to discuss its results using the statistics described in section 6.4. We recall that, since CITM, SITM, NITM and MITM at the end are all MITM attacks, only for the Attack Classification task we decided to fuse these attacks into one single acronym, that is MITM, and consider just this label during the training and testing of the model. Indeed, we are interested in understanding that a MITM is occurring, and not necessarily which specific implementation of MITM is in progress.

In Figure 6.16 it is possible to see the confusion matrix of the test set related to the RF model trained with the parameters discussed in section 6.3. Maybe for the large support of MITM samples, normal examples are often misunderstood with them. Also, enough normal samples are classified as DoS samples and vice versa.

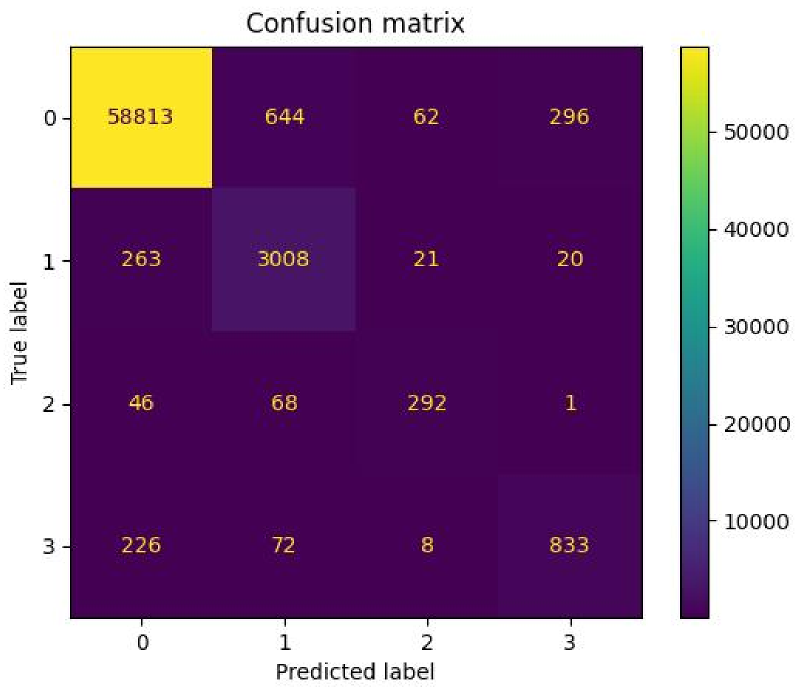

**Figure 6.16:** Confusion matrix related to the RF model. 0 refers to the normal sample, 1 refers to MITM, 2 refers to Device attack, 3 refers to DoS.

Table 6.18 shows instead the statistics obtained by the predictor. As we can see, normal samples have really good Precision, Recall, and F1 score, probably for the high support. Instead, between the three attacks, the MITM is the one with the highest Recall and F1 score, even if there isn't a big difference. Surprisingly, DoS have the lowest performance jointly with Device attack, around 0.73 and 0.82 for all the three metrics. This can be explained by the low support of DoS and Device attacks, and the labeling process not properly precise: indeed, we just label the packets as malicious when they are between the timestamp of the first and the last iterations related to the attacks. However, some packets between these may belong to the normal traffic flow, because reasonably even if there is an attack ongoing, not all the packets are considered malicious. So, better labeling should have been done. Nevertheless, this is out of the scope of this thesis, which is to prove the efficiency of DIDS in detecting attacks in an ICS. Indeed, better labeling would need to be created from scratch with ad-hoc techniques.

The Accuracy, as shown by Table 6.18, is 0.97, so really good. However, due to the high-class imbalance in the dataset, this statistic is not useful since it just says that a lot of normal samples are correctly classified, and a small part of attack samples too. The model shows around 0.83 Macro-average Precision, Macro-average Recall, and Macro-average F1 score, and 0.97 Weighted-average Precision, Weighted-average Recall, and Weighted-average F1 score, by what is reported in Table 6.18. However, these last three metrics are not so reliable in our case of the imbalanced dataset, since they consider too much the support of each class in the final result. So, the number of normal samples predominates the statistics, while the low support of the attack samples causes their statistical value to be little considered.

**Table 6.18:** Statistics related to the trained RF model. "Normal" refers to the normal data without attacks.

| Statistics → | Precision | Recall | F1 score | Support |
|---|---|---|---|---|
| Normal | 0.99 | 0.98 | 0.99 | 59815 |
| MITM | 0.78 | 0.89 | 0.83 | 3312 |
| Device atk | 0.82 | 0.75 | 0.78 | 407 |
| DoS | 0.73 | 0.76 | 0.74 | 1139 |
| Accuracy | - | - | 0.97 | 64673 |
| Macro avg | 0.83 | 0.85 | 0.84 | 64673 |
| Weighted avg | 0.97 | 0.97 | 0.97 | 64673 |

# Chapter 7

# Conclusions

In this work, a DIDS for ICS security is described and implemented, jointly with the performance assessment of its constituents unsupervised IIDSs, that are, IAM, IAR, MinMax, ST, GT, HM, their combination, and the supervised ML Attack Classifier model RF. The DIDS results are compared with the ones obtained by the supervised RNN algorithm called BLSTM. All the detectors and the classifier are tested on a dataset containing six kinds of attacks and three normal operation executions with no attacks running. The optimal parameters of the models are obtained using specific types of *grid search* algorithms focused on the task we want to perform. The performance of all the detectors has been measured in terms TD, FD, TNO, FSO, MDD, SDD, BDD, and WDD, while for what concern the RF model, Precision, Recall, F1 score, Accuracy, Macro-averaged and Weighted-averaged for Precision, Recall, and F1 score are used.

We showed that the detectors sometimes achieve good results when detecting attacks on network traffic and operational data of the field devices. However, the combination of these detectors shows *their potential*, and it is, therefore, a suitable choice for building an IIDS since it reaches the *optimal results on each dataset* even surpassing BLSTM, one of the detectors with the best performance in Wolsing et al. [5] work. So, DIDS turned out to be very useful in CI, where the consequences of a failure can be really serious. The unsupervised nature of our detectors allows us to detect with enough probability also non-targeted *zero-days attacks*, while their *transparency and non-complexity* make them alternative choices to more complex black box methods, like DNN and RNN. Furthermore, our DIDS *abstracts* each network message by converting it into a common format for each different protocol, in this way allowing its operation on a wide range of heterogeneous ICS network traffic.

Regarding the Attack Classification task, we show that the CICFlowMeter and RF model work reasonably well in order to have network traffic flow features and to classify them, respectively. However, the imbalanced dataset (partially solved using SMOTE) and the non-properly accurate labeling process probably caused some results to be a bit too lower than others. This doesn't mean that RF is not good, but that maybe it mistakes the classification of some classes related to the attacks because they are not 100% correctly labeled and/or because such classes have low support. So, there is a percentage of doubt on the RF *worst results*, which could improve or keep unchanged these letters.

Combining the final detection results with the output of the Attack Classifier model, it is possible not only to have a better understanding of the type of the ongoing attack, but also to have another reliable source that introduces an additional layer of security against malicious attacks.

By deploying the NIDSs between the PLCs and their respective switch we have got good results. Nevertheless, at the cost of having one detector for each PLC: this can cause *late detection*, since the number of detectors is proportional to the detection time, and it is *not economically cheap* for small manufacturing enterprises. So, further work should be done in order to test the results of just a single NIDS sniffing the traffic directed towards and from the central router (Router0 in Figure 6.15) instead. Before that, a labeling improvement task is needed, maybe considering the IPs involved in the attacks, accompanied by an attack data augmentation to solve the dataset imbalance in the Attack Classification task. Furthermore, since the tested attacks are only six, it may be useful to test other kinds of attacks. Also, conducting some stress tests injecting noises and latencies on the simulated control network and see if the detectors are able to ignore them or not could be a possible future work, since these issues may normally happen in a real scenario ICS.

# References

**[1]** *C. Alcaraz, “Secure interconnection of IT-OT networks in industry 4.0,” in Critical Infrastructure Security and Resilience. Cham, Switzerland: Springer, 2019, pp. 201–217.*

**[2]** *Margolin, J.: Outdated Computer System Exploited in Water Treatment Plant Hack (2021), accessed: 2022–04-24.* URL:
`www.abc7news.com/story/10328196/`

**[3]** *Kus, D., et al.: A False Sense of Security? ACM CPSS, revisiting the state of machine learning-based industrial intrusion detection. In (2022).*

**[4]** *Etalle, S.: From intrusion detection to software design. In: ESORICS (2017).*

**[5]** *Wolsing, K., Thiemt, L., Sloun, C.v., Wagner, E., Wehrle, K., Henze, M. (2022). Can Industrial Intrusion Detection Be SIMPLE?. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds) Computer Security – ESORICS 2022. ESORICS 2022. Lecture Notes in Computer Science, vol 13556. Springer, Cham.* URL:
`https://doi.org/10.1007/978-3-031-17143-7_28`

**[6]** *Conti, Mauro & Donadel, Denis & Turrin, Federico. (2021). A Survey on Industrial Control System Testbeds and Datasets for Security Research.*

**[7]** *Understanding Machine Learning: From Theory to Algorithms, 2014, Shai Shalev-Shwartz and Shai Ben-David.* URL:
`http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning`

**[8]** *Muhammad Azmi Umer, Khurum Nazir Junejo, Muhammad Taha Jilani, Aditya P. Mathur, Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations, International Journal of Critical Infrastructure Protection, Volume 38, 2022, 100516, ISSN 1874-5482.* URL:
`https://doi.org/10.1016/j.ijcip.2022.100516`

**[9]** *Pinto, A.; Herrera, L.-C.; Donoso, Y.; Gutierrez, J.A. Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure. Sensors 2023, 23, 2415..* URL:
`https://doi.org/10.3390/s23052415`

**[10]** *J. Giraldo et al., "A survey of physics-based attack detection in cyber-physical systems," ACM Comput. Surv., vol. 51, no. 4, 2018, Art. no. 76.*

**[11]** *Dina Hadžiosmanović, Robin Sommer, Emmanuele Zambon, and Pieter H. Hartel. 2014. Through the eye of the PLC: semantic security monitoring for industrial processes. In Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14). Association for Computing Machinery, New York, NY, USA, 126–135..* URL:
https://doi.org/10.1145/2664243.2664277

**[12]** *D. Mashima and A. A. Cárdenas, "Evaluating electricity theft detectors in smart grid networks," in Proc. Int. Workshop Recent Advances Intrusion Detection, 2012, pp. 210–229..*

**[13]** *S. Mishra, Y. Shoukry, N. Karamchandani, S. N. Diggavi, and P. Tabuada, "Secure state estimation against sensor attacks in the presence of noise," IEEE Trans. Control Netw. Syst., vol. 4, no. 1, pp. 49–59, Mar. 2017.*

**[14]** *C. Murguia and J. Ruths, "Characterization of a cusum modelbased sensor attack detector," in Proc. IEEE 55th Conf. Decis. Control, 2016, pp. 1303–1309.*

**[15]** *Y. Mo, S. Weerakkody, and B. Sinopoli, "Physical authentication of control systems: Designing watermarked control inputs to detect counterfeit sensor outputs," IEEE Control Syst. Magazine, vol. 35, no. 1, pp. 93–109, Feb. 2015.*

**[16]** *Nong Ye, Syed Masum Emran, Qiang Chen, Sean Vilbert, Multivariate statistical analysis of audit trails for host-based intrusion detection, Comput. IEEE Trans. 51 (7) (2002) 810–820.*

**[17]** *Eirini Anthi and others, A three-tiered intrusion detection system for industrial control systems, Journal of Cybersecurity, Volume 7, Issue 1, 2021, tyab006.* URL:
https://doi.org/10.1093/cybsec/tyab006

**[18]** *Duque Anton, Simon & Sinha, Sapna & Schotten, Hans. (2019). Anomaly-based Intrusion Detection in Industrial Data with SVM and Random Forests. 1-6. 10.23919/SOFTCOM.2019.8903672.*

**[19]** *Colelli, Riccardo & Magri, Filippo & Panzieri, Stefano & Pascucci, Federica. (2021). Anomaly-Based Intrusion Detection System for Cyber-Physical System Security. 428-434. 10.1109/MED51440.2021.9480182.*

**[20]** *Ankang Chu, Yingxu Lai, Jing Liu, and Clemente Galdi. 2019. Industrial Control Intrusion Detection Approach Based on Multiclassification GoogLeNet-LSTM Model. Sec. and Commun. Netw. 2019 (2019).* URL:
https://doi.org/10.1155/2019/6757685

**[21]** *Feng, Cheng & Li, Tingting & Chana, Deeph. (2017). Multi-level Anomaly Detection in Industrial Control Systems via Package Signatures and LSTM Networks. 10.1109/DSN.2017.34.*

**[22]** *Goh, Jonathan & Adepu, Sridhar & Tan, Yi Xiang Marcus & Lee, Zi. (2017). Anomaly Detection in Cyber Physical Systems Using Recurrent Neural Networks. 140-145. 10.1109/HASE.2017.36.*

**[23]** *Homayouni, Hajar & Ghosh, Sudipto & Ray, Indrakshi & Gondalia, Shlok & Duggan, Jerry & Kahn, Michael. (2020). An Autocorrelation-based LSTM-Autoencoder for Anomaly Detection on Time-Series Data. 5068-5077. 10.1109/BigData50022.2020.9378192.*

**[24]** *Khurum Nazir Junejo and Jonathan Goh. 2016. Behaviour-Based Attack Detection and Classification in Cyber Physical Systems Using Machine Learning. In Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security (CPSS '16). Association for Computing Machinery, New York, NY, USA, 34–43.* URL:
https://doi.org/10.1145/2899015.2899016

**[25]** *Aleksei Kharitonov and Axel Zimmermann. 2019. Intrusion Detection Using Growing Hierarchical Self-Organizing Maps and Comparison with other Intrusion Detection Techniques. In Proceedings of the 5th on Cyber-Physical System Security Workshop (CPSS '19). Association for Computing Machinery, New York, NY, USA, 13–23.* URL:
https://doi.org/10.1145/3327961.3329531

**[26]** *Perez, Rocio & Adamsky, Florian & Soua, Ridha & Engel, Thomas. (2018). Machine Learning for Reliable Network Attack Detection in SCADA Systems. 633-638. 10.1109/TrustCom/BigDataSE.2018.00094.*

**[27]** *Radoglou Grammatikis, Panagiotis & Sarigiannidis, Panagiotis & Efstathopoulos, George & Karipidis, Paris & Sarigiannidis, Antonios. (2020). DIDEROT: an intrusion detection and prevention system for DNP3-based SCADA systems. 1-8. 10.1145/3407023.3409314.*

**[28]** *Raman, Maitreyi & Somu, Nivethitha & Mathur, Aditya. 2019. Anomaly Detection in Critical Infrastructure Using Probabilistic Neural Network.*

**[30]** *Lin, CY., Nadjm-Tehrani, S., Asplund, M. (2018). Timing-Based Anomaly Detection in SCADA Networks. In: D'Agostino, G., Scala, A. (eds) Critical Information Infrastructures Security. CRITIS 2017. Lecture Notes in Computer Science(), vol 10707. Springer, Cham.* URL:
https://doi.org/10.1007/978-3-319-99843-5_5

**[31]** *Konrad Wolsing, Eric Wagner, Antoine Saillard, and Martin Henze. 2022. IPAL: Breaking up Silos of Protocol-dependent and Domain-specific Industrial Intrusion Detection Systems. In Proceedings of the 25th International Symposium on Research in*

*Attacks, Intrusions and Defenses (RAID '22). Association for Computing Machinery, New York, NY, USA, 510–525.*
https://doi.org/10.1145/3545948.3545968

**[33]** *Murillo, Andrés & Taormina, Riccardo & Tippenhauer, Nils Ole & Salaorni, Davide & Dijk, Robert & Jonker, Luc & Vos, Simcha & Weyns, Maarten & Galelli, Stefano. (2022). High-fidelity Cyber and Physical Simulation of Water Distribution Systems. Part 1: Models and Data.*

**[34]** *Nozomi Networks. June 2018. Advancing ICS Visibility and Cybersecurity with the Nozomi Networks Solution.*

**[35]** *Giuseppe Bernieri, Mauro Conti, and Federico Turrin. 2019. KingFisher: an Industrial Security Framework based on Variational Autoencoders. In Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML 2019). Association for Computing Machinery, New York, NY, USA, 7–12.* URL:
https://doi.org/10.1145/3362743.3362961

**[36]** *James Ho. Step by Step Tutorial of Sci-kit Learn Pipeline, Published in Towards Data Science.* URL:
https://towardsdatascience.com/step-by-step-tutorial-of-sci-kit-learn-pipeline-62402d5629

**[37]** *SMOTE for Imbalanced Classification with Python by Jason Brownlee on January 17, 2020 in Imbalanced Classification* URL:
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification

**[38]** *Chawla, Nitesh & Bowyer, Kevin & Hall, Lawrence & Kegelmeyer, W.. (2002). SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Intell. Res. (JAIR). 16. 321-357. 10.1613/jair.953.*

**[39]** *C. Alcaraz and J. Lopez, "Digital Twin: A Comprehensive Survey of Security Threats," in IEEE Communications Surveys & Tutorials, vol. 24, no. 3, pp. 1475-1503, third quarter 2022, doi: 10.1109/COMST.2022.3171465.*

**[40]** *CICFlowMeter (formerly ISCXFlowMeter) Applications.* URL:
https://www.unb.ca/cic/research/applications.html#CICFlowMeter

**[41]** *S. B. Mathieu Guillame-Bert and J. P. Josh Gordon, "Introducing TensorFlow decision forests." [Online].* URL:
https://blog.tensorflow.org/2021/05/introducing-tensorflow-decision-forests.html

**[42]** *Rossman, L. A. 2000. EPANET 2: Users manual. Cincinnati: Water Supply and Water Resources Division, National Risk Management Research Laboratory.*

**[43]** *Antonioli, D., and N. O. Tippenhauer. 2015. "MiniCPS: A toolkit for security research on CPS networks." In Proc., 1st ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy, CPS-SPC '15, 91–100. New York: Association for Computing Machinery.*

**[44]** *Lantz, B., B. Heller, and N. McKeown. 2010. "A network in a laptop: Rapid prototyping for software-defined networks." In Proc., 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX. New York: Association for Computing Machinery.*

**[45]** *Sklearn Pipeline.* URL:
https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

**[46]** *Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016.* URL:
https://www.deeplearningbook.org/

**[47]** *Introduction to Deep Learning, GeeksForGeeks* URL:
https://www.geeksforgeeks.org/introduction-deep-learning/

**[48]** *Su, Tongtong & Sun, Huazhi & Zhu, Jinqi & Wang, Sheng & Li, Yabo. (2020). BAT: Deep Learning Methods on Network Intrusion Detection using NSL-KDD dataset. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2972627.*

**[49]** *TensorFlow Website.* URL:
https://www.tensorflow.org/

**[50]** *Khan, Wazir & Xiang, Yang & Aalsalem, Mohammed & Arshad, Quratulain. (2012). The Selective Forwarding Attack in Sensor Networks: Detections and Countermeasures. International Journal of Wireless and Microwave Technologies. 2. 33-44. 10.5815/ijwmt.2012.02.06.*

**[51]** *M. Grieves, "Digital twin: Manufacturing excellence through VirtualFactory replication," Dassault Systèmes' DELMIA, Digital Twin White Paper, vol. 1, pp. 1–7, 2014. [Online].* URL:
https://www.3ds.com/fileadmin/PRODUCTS-SERVICES/DELMIA/PDF/Whitepaper/DELMIA-APRISO-Digita
pdf

# Acronyms and Abbreviations

**ANN** Artificial Neural Networks. 10, 11

**BDD** Best Detection Delay. 50, 53, 54, 56, 58, 61

**BLSTM** Bidirectional Long Short Term Memory. 2, 3, 19, 21, 37, 56–58, 61

**CI** Critical Infrastructure. 1, 5, 61

**CIP** Common Industrial Protocol. 6, 28, 29, 39, 41, 42

**CITM** Concealment Man-in-the-Middle. 29, 38, 41–46, 58

**CLT** Central Limit Theorem. 17, 18, 31

**CPS** Cyber-Physical Systems. 1, 7, 20

**DD** Detection Delay. 50, 54–58

**DHALSIM** Digital HydrAuLic SIMulator. 3, 26–28, 30, 36

**DIDS** Distributed Intrusion Detection System. 2, 3, 8, 16, 20, 21, 23–25, 33, 34, 36, 38, 40, 46, 47, 50, 56–59, 61

**DL** Deep Learning. 9, 10, 19, 56

**DNN** Deep Neural Networks. 21, 61

**DoS** Denial of Service. 7, 29, 37, 38, 42–44, 46, 52, 55, 56, 58, 59

**DT** Digital Twin. 2, 3, 7, 9, 26, 29

**FD** False Detection. 50, 52, 55–57, 61

**FSO** False NOrmal. 50, 52, 56, 61

**GT** Gradient. 25, 33, 37, 55, 56, 61

**HM** Histogram. 25, 33, 37, 48, 55, 61

**HMI** Human Machine Interface. 5, 6

**IAM** Inter-Arrival Mean. 3, 17, 25, 30–32, 37, 48, 49, 52–54, 56, 57, 61

**IAR** Inter-Arrival Range. 3, 17, 25, 30, 31, 37, 48, 49, 52–54, 56, 57, 61

**ICS** Industrial Control System. 1–5, 7, 8, 20, 24–28, 30, 31, 33, 36, 39–48, 58, 59, 61, 62

**IDS** Intrusion Detection System. 2, 7, 8, 20, 21, 56

**IIDS** Industrial Intrusion Detection System. 2–4, 7, 8, 20, 23, 25, 46, 48, 56, 61

**IPAL** Industrial Protocol Abstraction Layer. 3, 24, 29, 30

**IT** Information Technology. 1, 5, 20, 21

**LL** Lower Limitation. 17, 18, 31, 32

**LSTM** Long Short Term Memory. 15, 16, 19, 21

**MDD** Mean Detection Delay. 50, 53, 54, 56, 58, 61

**MinMax** Minimum and Maximum. 25, 33, 37, 55, 61

**MITM** Man-In-The-Middle. 7, 28, 29, 37–41, 43, 51–54, 56, 58, 59

**ML** Machine Learning. 2, 3, 8–10, 18, 20, 21, 23, 34, 38, 61

**MSE** Mean Squared Error. 12, 14

**NIDS** Network Intrusion Detection System. 8, 17, 20, 23, 25, 30, 36, 37, 47–50, 52–54, 57, 58, 62

**NITM** Naive Man-In-The-Middle. 28, 38–40, 43, 52, 58

**OT** Operational Technology. 1, 5, 6, 8, 20, 21

**PIDS** Physics-based Intrusion Detection System. 8, 20, 23–25, 30, 33, 36, 37, 48–50, 54–57

**PLC** Programmable Logic Controller. 1, 5, 6, 27–29, 36, 37, 39, 41–49, 52–54, 62

**RF** Random Forest. 2, 3, 8, 18, 19, 21, 25, 34, 35, 38, 49–51, 58–61

**RNN** Recurrent Neural Networks. 10, 12–15, 19, 56, 61

**SCADA** Supervisory Control and Data Acquisition. 1, 3, 5, 6, 20, 21, 28–30, 36, 37, 41, 47, 56

**SDD** Standard deviation of the Detection Delays. 50, 53, 54, 56, 61

**SITM** Server Man-In-The-Middle. 29, 38, 41, 43, 45, 55, 58

**SMOTE** Synthetic Minority Over-sampling TEchnique. 9, 34, 38, 61

**ST** SteadyTime. 25, 33, 37, 55–57, 61

**SVM** Support Vector Machine. 20, 21

**TD** True Detection. 50, 52, 55, 56, 61

**TNO** True NOrmal. 50, 52, 55, 56, 61

**UL** Upper Limitation. 17, 18, 31, 32

**VAE** Variational Autoencoder. 21

**WDD** Worst Detection Delay. 50, 53, 54, 56, 58, 61