Classifying network abnormalities into faults and attacks in IoT-based cyber physical systems using machine learning

Georgios Tertytchny[a,b,*], Nicolas Nicolaou[a,b,c] and Maria K. Michael[a,b]

[a]Electrical and Computer Engineering Department, University Of Cyprus
[b]KIOS Research and Innovation Centre Of Excellence, University Of Cyprus
[c]Algolysis Ltd

ARTICLE INFO

Keywords:
Machine Learning (ML)
Cyber Physical Systems (CPS)
Internet Of Things (IoT)
Cyber Security
Fault Diagnosis
Energy Aware Smart Home (EASH)

ABSTRACT

Cyber Physical Systems (CPS) integrate physical processes with electronic computing devices and digital communication channels. Their proper operation might be affected by two main sources of abnormality, security attacks and failures. The topics of fault diagnosis and security attack analysis in CPS have been studied extensively in a stand-alone manner. However, considering the co-existence of both sources of abnormality, faults and attacks, in a system and being able to differentiate among them, is an important and timely problem not yet addressed adequately. In this work, we study the internal communication environment of an Energy Aware Smart Home (EASH) system. More specifically, we formally define the problem of differentiating between component failures and network attacks in EASH, based on their effect on the communication behaviour. We formally show the correlation between such abnormality sources and provide a machine learning based framework for the differentiation problem. Our framework is evaluated using a simulation as well as a real-time testbed environment, demonstrating a promising accuracy in classification of over 85%. Based on the obtained experimental results, we also provide a detailed analysis on the considered classes and features used in the proposed approach, which can further improve the classification accuracy.

1. Introduction

Cyber Physical Systems (CPS) are combination of computational, networking and physical processes. CPS can be used to model a plethora of systems, including intelligent critical infrastructures. In fact, the extensive integration of CPS in critical infrastructures elevated their role in ensuring economic development [1, 2] and, hence, their security and resilience has become of utmost importance in all aspects of modern life. Two main causes of abnormality that may affect the proper operation of CPS are *security attacks* and *component failures*. As CPS play a vital role for the day-to-day operations of modern societies, they have become an appealing target for attacks by malicious actors [3, 4]. Their widespread use is leading to a significant increase in their attack surface. At the same time, just like any other physical control system, different components of CPS can suffer failures [5]. Both failures and attacks may lead to abnormal behaviour of the system, but their implications may differ greatly. Having the means of differentiating allows CPS operators to choose the proper recovery actions that minimise the negative effects of abnormal behaviour. Specifying the parameters that may lead to such differentiation is a complex task which requires the study of specific components in a CPS system, before reaching a holistic approach.

The proliferation of Internet of Things (IoT) technologies and their integration within CPS enables better monitoring, control and management of these systems. However, such systems are increasingly susceptible to component faults due to the nature of the integrated IoT devices, as well as security attacks as the underlying communication protocols used can introduce such new vulnerabilities to the system.

In this work, we focus on the communication network of such an IoT-based CPS, that is an Energy Aware Smart Home System (EASH). An example of an EASH is shown in Figure 1. In such systems, sensor/computing nodes measure and communicate the (instant) energy consumption of home electronic devices and appliances. These measurements are transmitted to a central node (coordinator), via common wireless and wired protocols, which stores and analyses the overall energy consumption in the house. From the central coordinator the energy consumption can then be reported to the energy utility by using the *Advanced Metering Infrastructure* (AMI), which enables a two-way communication between consumers (Smart Homes) and utilities (Smart Grids). This allows consumers to be aware of their home energy footprint (energy aware), and make decisions on how to minimise their energy consumption.

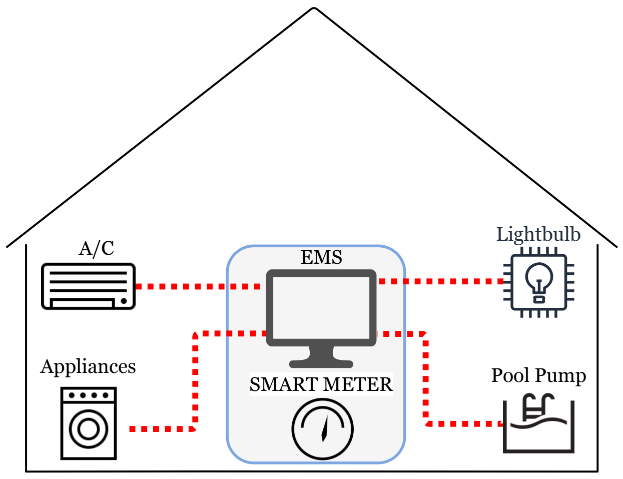


Figure 1: Energy Aware Smart Home System (EASH)

The normal behaviour of CPS can be altered by a failure that affects some of its components (nodes).Fault diagnosis in CPS has emerged as a challenging task due to the heterogeneity and large scale of the system, and because a faulty behaviour is a complicated and dynamic problem. Traditional solutions for fault diagnosis in CPS are based on operator's experience [6], while more recent approaches [7] utilise sensor and alarm

*Corresponding author
gterty01@ucy.ac.cy (G. Tertytchny); nicolas@algolysis.com (N. Nicolaou); mmichael@ucy.ac.cy (M.K. Michael)

data, characterising the new era of IoT. Such IoT solutions for fault diagnosis combine both machine learning approaches and human expertise. For instance, fault diagnosis in power and smart grid systems utilise artificial neural network which are adaptive systems inspired by biological systems. Common techniques under the artificial neural networks are Radial Basis Function (RBF) [8] and Support Vector Machines (SVM) [9]. Other approaches use reasoning [10] in order to eliminate hidden failures that can arise when a healthy system is triggered by a monitoring system for a failure situation.

Apart from fault diagnosis, security concerns for CPS have been studied in the area of systems fault detection, isolation, and recovery [11] where security attacks are modelled as extreme fault cases. CPS security has gained considerable attention in the recent years. Authors in [12] discuss the importance of CPS security using a simple case scenario where an attacker can compromise data transmitted between vehicles (e.g., by sending tampered distance information) leading to several accidents with catastrophic consequences. Hence, unique characteristics and vulnerabilities in CPS require new, more appropriate, detection and identification techniques. As mentioned in [13], traditional IT security methods are not applicable for CPS security. Vulnerabilities encountered in the communication protocols used for the transmission of measurements and control packets increase the possibility of attacks that can affect the physical plants [4]. Existing techniques for CPS security are typically categorised based on the security triad of confidentiality, integrity, and availability. A security goal is also related with the appropriated mitigation mechanisms that aim to protect a CPS system described by a specific system model from an adversary, with its respective model [14].

Research in cyber security and fault diagnosis, affecting systems such as EASH [15], is mainly focused on detection mechanisms [16, 17]. Such mechanisms are able to identify when a system's behaviour deviates from normal, but are unable to identify the source of abnormality. The normal behaviour of the system can be described either using a state estimator anomaly detection [18, 16], or historical data used in intrusion detection mechanisms [17, 19]. But, while existing studies deal with the detection and identification of attacks in such systems [20], work taking into account both attacks and faults is limited.

The differentiation problem between faults and attacks has been studied in different environments and systems, such as water infrastructures and smart grids. The work in [5] takes into consideration both fault diagnosis and network anomaly detection in the context of water infrastructure systems, by combining historical process data with adaptive control techniques. Authors in [21], present an approach that uses state fault diagnosis matrices for cyber attack detection and fault diagnosis in power systems. Also, a detection scheme that takes into consideration both attacks and faults is proposed in [22]. Here, the proposed framework utilises Kalman filters in order to estimate the variables of a wide range of state processes in a determined model. Using these variables and system readings they try to detect attacks and faults affecting the smart grid.

In this work we investigate how to differentiate component faults (which affect the normal behaviour of the network) from network attacks by examining channel characteristics in an EASH system, using supervised machine learning. The main contributions of this work are:

- We provide a theoretical analysis where we propose the differentiation operator for the problem of differentiating faults and attacks in an EASH system. We then show the correlation between these two classes of abnormalities, based on their effect on the communication channel.
- We give a general Machine Learning (ML) based framework that utilises data collected from the communication channels of EASH scenarios.
- We evaluate the proposed framework for a number of different classification algorithms using both a *simulation framework* as well as a *real-time testbed*.
- We provide a detailed feature and class based analysis for the cases we consider in the evaluation of the proposed ML framework, for both evaluation environments.

Overall, in this work we show that the use of supervised machine learning algorithms is a promising technique for the differentiation problem of faults and attacks, as for the cases examined we achieved to differentiate with an accuracy over 85%. Furthermore, our feature analysis shows that an appropriate selection of the features can lead to improved results.

The rest of the paper is organised as follows. Section 2 provides a precise problem formulation for differentiating between faults and attacks, the abnormality sources we examine in this work. In Section 3 we establish a theoretical relation between the fault and attack classes, based on the preceding problem formulation. The proposed methodology for the Machine Learning (ML) based framework for the differentiation problem is presented in Section 4. Sections 5 and 6 describe the simulation and real-time testbed evaluation platforms developed and corresponding results, respectively. An analysis on the classification results obtained in both evaluation environments on a per class and per feature basis is provided in Section 7. Finally, Section 8 concludes this work and discusses future work.

2. Problem Formulation

Our goal is to design a methodology for differentiating of faults and attacks in an EASH system. We model the system as a graph $G = (V, E)$ composed of a set V of m peripheral monitoring computing devices (nodes) $\{q_1, \dots, q_m\}$, and a central hub (coordinator node) q_c. Two nodes $q_i, q_j \in V$ communicate by exchanging messages via synchronous, reliable, communication channels if $(q_i, q_j) \in E$. We use $e_{i,j}$ as a shorthand to denote the channel between q_i and q_j, given that $(q_i, q_j) \in E$. Each $q_i \in V \setminus \{q_c\}$ communicates directly to the coordinator node q_c. Thus, a star topology exists where $(q_i, q_c) \in E, \forall q_i \in \{q_1, \dots, q_m\}$. For each $q_i \in \{q_1, \dots, q_m\}$, we define a single binary connectivity variable v to indicate the direct connection of q_i to the coordinator:

$$q_i.v = \begin{cases} 1 & \text{if } (q_i, q_c) \in E \\ 0 & \text{otherwise} \end{cases}$$

Thus node q_i state is given by:

$$\sigma^i = \{(q_i.b_j) : b_j \in B_i\} \bigcup \{(q_i.v)\}$$

Similarly, the state of each channel $e_{i,j}$ denoted by $\sigma^{e_{i,j}}$, is defined over a set of state variables, $C_{i,j}$. Using the snapshot of the state of the channel, or a sequence of state snapshots, we derive a set of measurements, that describe the channel conditions and performance metrics (e.g., delay). Thus channel $e_{i,j}$ state is given by:

$$\sigma^{e_{i,j}} = \{(e_{i,j}.y_k) : y_k \in C_{i,j}\}$$

The state of the system σ is a vector that contains the state of every node $q_i \in V$ and every channel $e_{i,j}$ for $(q_i, q_j) \in E$. Thus system state σ is given by:

$$\sigma = \{\sigma^{e_{i,j}} : (q_i, q_j) \in E\} \bigcup \{\sigma^i_t : q_i \in V\}$$

Each node $q_i \in V$ implements a set of actions. When an action α occurs at a node q_i it causes the state of q_i, and thus the state of the system, to change. An *execution* ξ of the system is an alternative sequence of states and actions, where actions send and receive events that may change the state of the channel or the state of a node in the system. A triple $\langle \sigma, \alpha, \sigma' \rangle$ is called an *execution step*, if the occurrence of α when the system is on state σ, yields the state σ' in an execution ξ. We assume that the system executes in *discrete time units* making one execution step in each time unit. Thus, σ_0 is the initial state of the system at time $t = 0$, and σ_t denotes the state of the system at time t in an execution ξ. We assume that each execution starts with the initial state and ends with a state. If σ_t is the ending state of an execution ξ then we say that t is the length of ξ and is denoted by $T(\xi) = t$. We say that an execution ξ' *extends* an execution ξ if the first state of ξ' is the last state of ξ. Let $(q_i.b_j)_t$ denote the value of the local variable b_j of node q_i at a time unit t. Similarly, we denote by $(q_i.v)_t$ the binary connectivity variable of node q_i and $(e_{i,j}.y_k)_t$ the value of the channel characteristic y_k at time t. Given this notation we can define the states of each node, channel and system state over t:

$$\sigma^i_t = \{(q_i.b_j)_t : b_j \in B_i\} \bigcup \{(q_i.v)_t\} \quad (1)$$

$$\sigma^{e_{i,j}}_t = \{(e_{i,j}.y_k)_t : y_k \in C_{i,j}\} \quad (2)$$

$$\sigma_t = \{\sigma^{e_{i,j}}_t : (q_i, q_j) \in E\} \bigcup \{\sigma^i_t : q_i \in V\} \quad (3)$$

For each state variable $q_i.b_j \in B_i$, we denote by $L(q_i.b_j)_t$ and $U(q_i.b_j)_t$, the *expected low and upper bounds* respectively, of the value of $q_i.b_j$, at a time unit t with respect to the previous state of the system σ_{t-1}. Using, these bounds we can now define *normality* and *abnormality* as follows:

Definition 1. *A state σ^i_t of node q_i in an execution ξ at time t is* normal *iff:*

$$\forall_j [(q_i.b_j \in B_i, (q_i.b_j)_t \in [L(q_i.b_j)_t, U(q_i.b_j)_t]]$$

$$\wedge (q_i.v)_t = 1$$

In other words a state of a node is normal if all the state variables are within the expected bounds, and the node is connected and communicating with the central node (i.e. $(q_i.v)_t = 1$). On the contrary in an abnormal state such conditions may not hold as explained in Definition 2.

Definition 2. *A state σ^i_t of node q_i in an execution ξ is* abnormal *at time t if at least one of the following holds:*

- abnormal node behaviour *iff $\exists j$ $q_i.b_j \in B_i$ such that:*

$$(q_i.b_j)_t < L(q_i.b_j)_t \vee (q_i.b_j)_t > U(q_i.b_j)_t$$

- abnormal node connectivity *iff:*

$$(q_i.v)_t = 0$$

By Definition 2, an abnormal state may include variables with unexpected values, and/or the direct connection of the node with the coordinator may be interrupted (i.e., $(q_i.v)_t = 0$). We assume that connectivity interruption may be caused by an external event (e.g, man-in-the-middle attack, or interruption attack) without necessarily the sending node be aware of this. We assume that an interrupted edge is removed from the set E. Following Definitions 1 and 2 we say that the system state σ_t is *normal* if $\forall q_i \in V$, σ^i_t is normal. Similarly we characterised a system state σ_t as *abnormal* if $\exists q_i$ such that σ^i_t is either an *abnormal node behaviour* or *abnormal node connectivity*. We define an execution ξ as *Normal-execution* if $\forall \sigma_t \in \xi$ σ_t is normal, for $0 \leq t \leq T(\xi)$. Similarly we say that an execution ξ is *ψ-execution*, for $\psi \in [Fault, Attack]$, if $\exists \sigma_t \in \xi$ such that σ_t is an abnormal state, resulting from a fault or attack respectively, for $0 \leq t \leq T(\xi)$. We call an action α as abnormal, if it is applied on a normal state and yields an abnormal state. That is, α is abnormal if the step $\langle \sigma_{t-1}, \alpha, \sigma_t \rangle$ appears in a *ψ-execution*, for $\psi \in [Fault, Attack]$, while σ_{t-1} is normal and σ_t is abnormal.

Definition 3. *Class state snapshot, denoted as w^{ψ}_t, is the system state σ_t, of class ψ, in an execution ξ, time $t \leq T(\xi)$, and $\psi \in [Normal, Attack, Fault]$.*

So given an algorithm A, we can define the differentiation operator, and thus the differentiation problem, as follows.

Problem 2.1 (Class Differentiation)**.** *Specify a* differentiation operator $\bigoplus_A$ *using a (machine learning) algorithm A such that, given two class state snapshot vectors w^{ψ}_t, $w^{\psi^*}_{t^*}$ from executions ξ and ξ^* (not necessarily different):*

$$w^{\psi}_t \bigoplus_A w^{\psi^*}_{t^*} = \begin{cases} T & \textit{If } \psi \neq \psi^* \\ F & \textit{otherwise} \end{cases}$$

3. On the Complexity of Fault and Attack Class Differentiation

This section examines how difficult it is to differentiate between fault and attack classes. In particular, we focus on a subset of fault-executions where faults may affect the state of the communication channels (i.e., by introducing abnormal messages). In the rest of the section, we show that this set of fault-executions is a proper subset of attack-executions where attacks gain access to the communication channels and alter the state of the channel. Our result, suggests that no algorithm running at the receiving end of a channel may differentiate attacks that fall in the fault spectrum, but differentiation is possible beyond boundary. We believe that results presented here may be extended in system parameters other than channel abnormality but this is not within the scope of this work. We assume that channels operate in a uniform way thus for the rest of the section we only focus on a single channel $e_{i,j}$.

Let $MSG^{e_{i,j}}$ refer to the buffer used for the communication between (q_i, q_j). $|MSG^{e_{i,j}}|$ refers to the size of the buffer and represents the total number of bits appended in the channel. $|MSG^{e_{i,j}}_t|$ refers to the size of message buffer at a time unit t.

We assume that each message m sent from a node q_i, includes a set of state variables of q_i we denote by $m.B_{i,m} \subseteq B_i$. A message may contain extra payload we denote by $m.data$.

Definition 4. We define a message m sent by a node q_i as a *normal* message if $\forall b \in m.B_{i,m}$, $L(q_i.b_j) < b < U(q_i.b_j)$; otherwise m is *abnormal*.

By Definition 4, a message is abnormal if it contains a value for a local variable of the sender, that is outside the expected boundaries. Given that definition we can now define integrity violating actions.

Definition 5. An action α_v is called an *integrity violating action* in an execution ξ, if the following hold (i) step $\langle \sigma_t, \alpha_v, \sigma_{t+1} \rangle$ appears in ξ, (ii) σ_t^i is *normal* and σ_{t+1}^i is *abnormal* of state variable $q_i.b \in B_i$, and (iii) the step $\langle \sigma_{t'}, send(m)_{i,*}, \sigma_{t'+1} \rangle$ appears in ξ, s.t. $t' > t$, this is the first step that contains a send action by q_i after state σ_{t+1} in ξ and m is abnormal on variable $q_i.b \in m.B_{i,m}$.

In particular, integrity violation actions are external actions that may lead to the generation of messages containing state variables with values outside the expected boundaries.

During the communication between two nodes q_i, q_j, let m_s denote the message transmitted from the sender q_i, and m_r the corresponding message received at the receiver q_j. To transmit a message from q_i to q_j, the sending node q_i, pushes m_s in the buffer $MSG^{e_{i,j}}$ and the receiving node q_j pops m_r from the buffer. Each channel $e_{i,j}$ exposes three different actions: $send(m)_{i,j}$, $receive(m)_{i,j}$ or $interrupt()_{i,j}$.

- $send(m)_{i,j}$ changes the state $\sigma_t^{e_{i,j}}$ of the channel $e_{i,j}$, by adding the message m in the buffer $MSG_{t+1}^{e_{i,j}}$. Formally, in an execution step $\langle \sigma_t, send(m)_{i,j}, \sigma_{t+1} \rangle$, the channel state $\sigma_{t+1}^{e_{i,j}}$ differs from $\sigma_t^{e_{i,j}}$ by setting:

$$MSG_{t+1}^{e_{i,j}} = MSG_t^{e_{i,j}} \cup \{m\}$$

- $receive(m)_{i,j}$ changes the state $\sigma_t^{e_{i,j}}$ of the channel $e_{i,j}$, by removing the message m from the buffer $MSG_{t+1}^{e_{i,j}}$. Formally, in an execution step $\langle \sigma_t, receive(m)_{i,j}, \sigma_{t+1} \rangle$, the channel state $\sigma_{t+1}^{e_{i,j}}$ differs from $\sigma_t^{e_{i,j}}$ by setting:

$$MSG_{t+1}^{e_{i,j}} = MSG_t^{e_{i,j}} \setminus \{m\}$$

- $interrupt()_{i,j}$ is an action performed by the environment (i.e. by an external physical event or an adversary), allowing an external entity to *obtain* or *alter* the contents of any single message $m \in MSG^{e_{i,j}}$. Therefore, in an execution step $\langle \sigma_t, interrupt()_{i,j}, \sigma_{t+1} \rangle$, either:
  - Inspect: $MSG_{t+1}^{e_{i,j}} = MSG_t^{e_{i,j}}$, or
  - Remove: $\exists m \in MSG_t^{e_{i,j}}$, s.t. $m \notin MSG_{t+1}^{e_{i,j}}$, or
  - Modify: $MSG_t^{e_{i,j}} \setminus \{m\} = MSG_{t+1}^{e_{i,j}} \setminus \{m'\} \wedge m \neq m'$

In other words, send actions append messages in the channel's buffer, and receive actions pop messages from the same buffer. Furthermore, interrupt actions are external events that allow a third party to gain access over the communication channel. Think of a man-in-the-middle attack, where the attacker may gain access to the channel and alter the communication. Having access on the channel the third party may modify, remove or just read any single message in the channel's buffer. A sequence of interrupt actions may change multiple messages in the buffer.

Given the channel actions, we can now define fault and attack spaces in terms of the existence of abnormal messages in a channel buffer.

Definition 6. A fault space $\mathcal{F}$ denote the set of all actions, denoted by f, that are integrity violation actions on a node q_i and lead to generation of *abnormal* messages.

Definition 7. An attack space $\mathcal{A}$ denote the set of all actions, denoted by a, that are $interrupt()_{i,j}$ actions, and can change *normal* messages to *abnormal*.

Given the definition of Faults $\mathcal{F}$ (Definition 6) and Attack $\mathcal{A}$ space (Definition 7) the following lemma shows that any fault can be emulated by an attack. In a nutshell, Lemma 1 shows that a receiver cannot distinguish two executions, such that, one contains a receive action that delivers an abnormal message generated by a fault $f \in \mathcal{F}$, and one that contains a receive action that delivers an abnormal message generated by an attack $a \in \mathcal{A}$ that simulates f.

Lemma 1. *For any pair of nodes q_i, q_j, if ξ' is an execution with a fault action $f_i \in \mathcal{F}$ on q_i which is followed by a pair of $send(m)_{i,j}$ and $receive(m)_{i,j}$ actions, and ξ'' an execution with an attack action $a_{i,j} \in \mathcal{A}$ that may generate message m, then the node q_j is not able to differentiate between ξ' and ξ''.*

Proof. **Execution Construction:** Let ξ be a normal execution such that $T(\xi) = t$, ending in state σ_t. We now construct the two executions ξ' and ξ'' as follows. Let without loss of generality [1], execution ξ' be a sequence:

$$\sigma_t, f_i, \sigma_{t+1}, send(m')_{i,j}, \sigma_{t+2}, receive(m')_{i,j}, \sigma_{t+3}$$

Similarly, let ξ'' be a sequence:

$$\sigma_t, send(m)_{i,j}, \sigma'_{t+1}, a_{i,j}, \sigma'_{t+2}, receive(m'')_{i,j}, \sigma'_{t+3}$$

Attack $a_{i,j}$ is a modify action and we assume that attacker has the knowledge for the fault simulation.

By contradiction. We assume that the receiver q_j is able to differentiate executions ξ' from ξ''. Due to the fact that receiver q_j has knowledge of ending states of both executions σ_{t+3}, σ'_{t+3}), differentiation is possible at q_j iff $\sigma^j_{t+3} \neq \sigma^{j\,\prime}_{t+3}$. Notice that as the actions f and $send(m)_{i,j}$ are actions that modify the state of q_i and of the channel $e_{i,j}$ respectively, the state of q_j remains the same in both states σ_t and σ_{t+1} in both executions ξ' and ξ''. The similar observation holds for states σ_{t+1} and σ_{t+2}. In particular the actions $send(m')_{i,j}$ and a respectively do not change the state of q_j, and hence $\sigma^j_{t+1} = \sigma^j_{t+2}$ in both executions ξ' and ξ''. As we assumed that action a may

[1]Note that any additional intermediary actions in the two executions can result in the same state transitions for q_j and thus q_j will reach the same state before receiving the message from q_i in both executions ξ' and ξ''.

emulate the fault f, then it may replace m with the message m' in the channel's buffer. Therefore $m'' = m'$ and the same $receive(m')_{i,j}$ occur in both ξ' and ξ''. This action affects the state of the channel $e_{i,j}$, as it removes message m' from the channel's state, and the state of q_j as it processes message m'. Since however $\sigma^j_{t+2} = \sigma^{j'}_{t+2}$ and the same action $receive(m')_{i,j}$ occurs in both executions, then q_j must transit in the same state $\sigma^j_{t+3} = \sigma^{j'}_{t+3}$. This however contradicts our initial assumption and completes our proof. □

Theorem 1. *If every fault $f \in \mathcal{F}$ can be emulated by a respective attack $a \in \mathcal{A}$ having the knowledge of fault simulation on the receiver, then there exists no operator $\bigoplus_A$ that may differentiate the two, at the receiver.*

Proof. The Theorem follows from Lemma 1. In particular, if an attack may emulate the message sent during a fault, then the receiver cannot differentiate the execution that contains the fault from the execution that contains the attack. Therefore any operator that uses the state of q_j (or just messages received) will provide the same result in both executions and hence will not be able to differentiate the fault from the attack. □

The next lemmas (Lemma 2, Lemma 3) show that executions that contain the occurrence of an attack a that performs an inspect on the messages of the buffer, they differ from executions that contain any fault $f \in \mathcal{F}$.

Lemma 2. *If an execution ξ' contains an integrity violating action $f_i \in \mathcal{F}$ on a node q_i, then every node q_j that receives the message send by the first $send(m)_{i,j}$ action followed f can differentiate ξ' from any normal execution ξ.*

Proof. The Lemma follows from Definition of an integrity violating action. In particular, followed action f, q_i sends an abnormal message m' via action $send(m')_{i,j}$ to q_j. The channel $e_{i,j}$ will eventually deliver m' to q_j when action of the $receive(m')_{i,j}$ appears in the execution ξ'. However any normal execution ξ that is similar to ξ', but does not contain f, will contain a $receive(m)_{i,j}$ action in ξ s.t. $m \neq m'$.

To derive contradiction lets assume that q_j cannot differentiate between ξ and ξ'. As in Lemma 1, we can show that the state of q_j is the same in both ξ' and ξ before the receive action occurs. So we need to examine the state of q_j after the occurrence of the $receive(*)_{i,j}$ action. There are two cases to consider: (a) the state of q_j changes when receiving a message, or (b) the state of q_j remains the same. If (b) is true then q_j will not change its state no matter how may and what messages it received from q_i. Thus, in this case is identical to the case where no communication link exists between the two nodes and hence in either case q_j could not be used to detect any fault on q_i. If now (a) is true, as $m \neq m'$ then q_j will reach a state $\sigma^{j'}$ in ξ' and σ^j in ξ such that $\sigma^{j'} \neq \sigma^j$. Thus, q_j will be able to differentiate ξ' from ξ contradicting our initial assumption. □

In a similar manner we can show that a node will not be able to differentiate a normal execution from an execution that contains an inspect attack.

Lemma 3. *If an execution ξ' contains an inspect attack $a_{i,j} \in \mathcal{A}$ on a channel $e_{i,j}$, then no node q_j may differentiate ξ' from any normal execution ξ.*

Proof. In order to proof this lemma consider that the two executions ξ and ξ' are identical up to the occurrence of the inspect attack action $a_{i,j}$ in ξ'. Let σ_t be the state immediately before $a_{i,j}$ in ξ' and before the $receive(m)_{i,j}$ action in ξ. Since action does not change the state of the channel (as it does not modify the messages in the buffer), then the state of the channel will be $\sigma^{e_{i,j}}_{t+1} = \sigma^{e_{i,j}}_t$. Hence if the channel included m in $receive(m)_{i,j}$ then it will include m in the $receive(m)_{i,j}$ in ξ' as well. Therefore, q_j will receive the same message in both executions. As q_j was in the same state in both executions before the occurrence of $receive(m)_{i,j}$ action then receiving the same message will lead q_j in the same state in the two executions. Thus, q_j will not be able to differentiate ξ from ξ'. □

Theorem 2. *There exist attack $a \in \mathcal{A}$ that may lead the system to a state σ such that σ does not appear in any execution that contains an integrity violating fault $f \in \mathcal{F}$.*

Proof. The proof of the Theorem follows from Lemmas 2 and 3. Notice that there exist an attack category, i.e. inspect attacks, that does not cause any state distinction at the receiving node (Lemma 3), while any integrity violating faults lead to distinguishable executions, (Lemma 2). □

4. Proposed ML-based Framework

This section presents the overall proposed Machine Learning (ML) based framework for differentiation between faults and attacks. According to the results presented in Section 3, it is difficult to differentiate attacks that resemble the behaviour of node faults at the receiving ends, as their effect on the communication channel is the same. Have we monitor the channel state however, we could potentially capture the state transition actions invoked by the attackers in order to generate a number of attacks. Thus, to overcome the limitation of differentiating attacks from faults on the receiving end, we choose to directly monitor the channel measurements. We then fit those measurements (in the form of feature datasets) in machine learning models with the aim to differentiate the two abnormality classes using the channel characteristics (and hence channel state).

The overall methodology, as illustrated in Figure 2, comprises of three phases. In the first phase, the system is modelled for the normal, faulty and attack classes. Consequently, in the second phase, a set of execution scenarios is performed in order to generate datasets which describe the system's behaviour under normal, faulty and attack classes. The datasets generated are used during the third phase, for the evaluation of different supervised machine learning algorithms for classification purposes in terms of the differentiation task.

The proposed framework is generic and accommodates the study of different classes of faults and attacks in various experimental setups, as well as the evaluation of the generated datasets over different supervised machine learning algorithms. Moreover, by focusing only in the characteristics of the communication channel, this framework is independent of device characteristics used in any IoT enabled CPS system.

This section focuses in this generic framework and presents in more detail the abnormality classes we consider in this work, the various ML classification algorithms studied, and the evaluation metrics used for the evaluation of the considered algorithms.

4.1. Execution Classes

Faults in systems like EASH, often appear in the sensing nodes having an impact on the communication channels. On the other hand, attacks are often seen to interfere with the communication channels, while also affecting the state variables (e.g., routing tables) of processing nodes. Therefore, the fault and attack types we choose: (i) resemble common abnormalities, and (ii) have a similar effect on the performance of the network in order to challenge the differentiation task. To do so, we directly monitor network and the state of the channel in the receiving end, to derive the features to be used. We consider the following execution classes:

- **Normal Class:** Includes *Normal-executions*, where the node exhibit normal behaviour by capturing measurements and transmitting packets to the central node with no attacks or faults.

- **Fault Class:** Includes *Fault-executions* where a system state σ_t is a *faulty state* if $\exists q_i \in V$ with σ_t^i being an *abnormal node behaviour* state, but not an *abnormal node connectivity* state (see Definition 2).

  In other words, a node may experience a fault that affects its local state variables (e.g., measurements, routing tables), but its connectivity to the central node is not interrupted. Such failures may also cause routing failures (due to routing table corruption) or packet drops (due to improper generation of the local packets). The normal operation of sensor nodes is affected by two types of faults that could lead to the degradation of their performance [23]. These are: *functionality* faults and *data* faults. For *data* faults the sensing readings of a node are altered abnormally, but it still communicate and behave according to its specification. On the other hand, during *functionality* faults, a node may diverge from its algorithm specification. This typically results in crash of individual nodes, packet loss, routing failure and network participation. The set of faulty classes (forming set F) is given by the following three classes, that are representative classes of *functionality* faults; F1: Low Energy Failure, F2: Routing Failure and F3: Packet Dropped Failure. Note that all three classes refer to faults at the nodes and not at the communication channels. Hence, the connectivity between the nodes is not affected, and thus there is no *abnormal node connectivity* as defined in Definition 2. Below we provide a short explanation of each fault class.

  Low Energy Failure (F1): This fault class contains executions where failures may reduce the power supply on a sending node. A sender experiencing this failure does not operate normally, as the amount of energy required for the packet generation and transmission is below a certain threshold. This may result to either corrupted messages sent in the channel, or the injection of faulty values in the state variables. In any case, a local state variable out of bounds, will result to an *abnormal node behaviour* state at the sending end.

  Routing Failure (F2): This fault class contains executions where failures may lead to the corruption of routing tables at the senders. Such corruption of the routing table leads to an *abnormal node behaviour* state: packets are generated correctly at the sender but transmitted inconsistently. Such failure may result in excessive delays in the communication, between the sender and the receiver.

  Packet Dropped Failure (F3): This fault class contains executions with failures causing packet corruption due to noise in the signal transmitter of the sender. In particular, a sending node affected by an external noise may not generate and transmit packets normally. A noise may cause the Signal To Noise Ratio (SNR), i.e. the ratio of the transmitting quality over the external noise, diverge from its expected bounds at the sender, leading to an *abnormal node behaviour* state. Thus, packets transmitted might not be readable by the receiver and result into some of them to be dropped.

- **Attack Class:** Includes *Attack-executions* where the system state σ_t is an *attack state* if $\exists q_i \in V$ and σ_t^i being both an *abnormal node behaviour* and *abnormal node connectivity* state. (see Definition 2).

  Man In The Middle attack (MITM) posses such characteristics, as the attacker tries to modify local variables of the communicating parties in order to gain access on the direct connectivity between them. Common attacks performed by MITM are message modification, replay and sink hole. Thus, the set of attack classes (forming set A) is given by the following three classes: A1: Replay Attack, A2: Sink Hole Attack and A3: Message Modification Attack. Those attack classes affect the direct connectivity between the sender and the receiver resulting to an *abnormal node connectivity* state at the sender. Moreover, the local variable that denotes the destination of packets at both ends is modified resulted to an *abnormal node behaviour* state. According to the attack class, the attacker can either decide to simply retransmit, modify or drop packets captured. The AIC triad (Availability, Integrity and Confidentiality) of security is affected by the attack classes considered. Availability, ensures that information is available when needed, integrity maintains and ensures accuracy and completeness of data, and confidentiality protects the information from unauthorised access. MITM, considered in all the three classes below, affects the confidentiality, while each of the attack class can also affect integrity, availability or both. Below we discuss each attack class, separately:

  Replay Attack (A1): contains executions with attacks where the attacker, that records the packets between the sender and the receiver (victims of the attack), retransmit them in a random order. By replaying the packets after a specific amount of time (delay), the receiver will either receive the packets in sequential order but with a delay, or receive the packets out of order and thus will drop them. This attack class also affects the availability of the information, but not the integrity as it leaves the content of the packets unmodified.

  Sink Hole Attack (A2): describes the attack class in which the attacker records the packets between the sender and the receiver (victims of the attack), but does not retransmit them. Thus, the attacker appears in the network as a sink-hole point, where packets are lost. This attack class affects mainly the

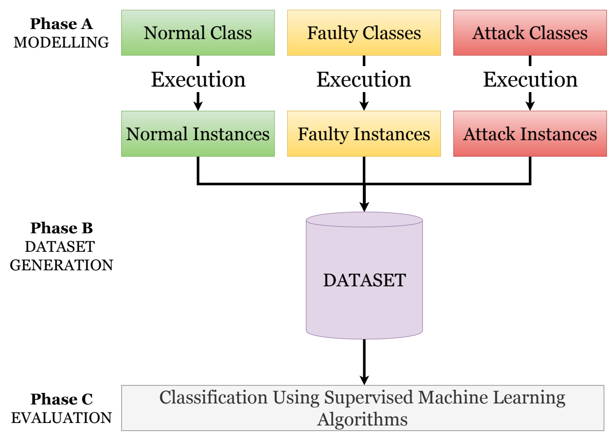
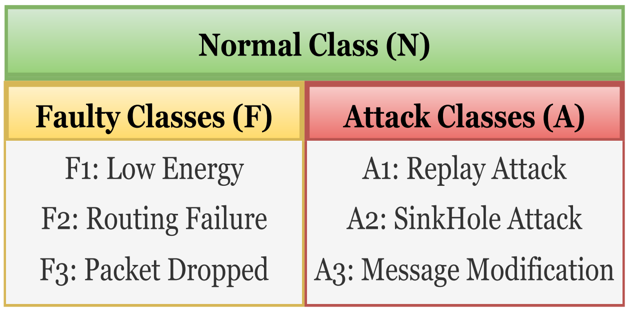


Figure 2: Flowchart of the proposed ML-based Framework



Normal Class (N)	
Faulty Classes (F)	Attack Classes (A)
F1: Low Energy	A1: Replay Attack
F2: Routing Failure	A2: SinkHole Attack
F3: Packet Dropped	A3: Message Modification

Figure 3: Execution Classes Considered

availability of the information, similar as if the packets are being dropped.

Message Modification Attack (A3): is the attack class in which the attacker modifies the packets recorded. The attacker may introduce fraudulent or inconsistent data resulting in a payload change. Modified packets are then transmitted to their destination. Such attacks target the integrity of the packets at the receiver, as their content differs from the content generated at the sender.

Figure 3 summarises all execution classes considered in this work.

4.2. ML Classification Algorithms

We perform an exploration on different ML algorithms for classification, focusing on the ones considered more appropriate for the underlying problem. Before we discuss the selected algorithms, we provide a review on the most commonly used supervised machine learning algorithms. The classification algorithms categories discussed below are derived from the separation made in [24].

4.2.1. Brief Overview of ML Classification Approaches

Decision trees[25], belong to the logic based category classification algorithms that use inductive logic for building predicted models. In practice, tree based classification models are trees that are built during training by sorting feature values. Nodes of this tree are the features, used for classification and instances are classified based on sorting from the root node to leafs based on the attribute values of the nodes.

The next category of classifiers is Artificial Neural Networks (ANN). ANN are described in term of three fundamental aspects, their input, activation functions of different layers, the overall architecture and the weights that relate each layer of the network. In the training phase the weights of the network are computed and adjusted based on the representation of the input data. These instances are repeatedly exposed to the net and in that way the network learns from the instances. Multilayered Perceptron (MLP) is a commonly used classification technique of ANN. Perceptrons are nodes that computes the sum of weighted inputs and output gets its value based on a threshold. If this sum is above the threshold then the output value is one, otherwise it is zero. A more recent classification algorithm technique is Support Vector Machines (SVM). SVM [26] similarly to ANN aim to build a hyper plane that separates two data classes by maximising the margin and thereby creating the largest possible distance between the separating hyper plane and the instances. This technique aims to minimise the overall generalisation error. Another category involves statistical learning algorithms. Their main characteristic is that they have an explicit underlying probability model that gives a probability for each instance to which class it belongs. In such a way, the ultimate target of those algorithms is the computation of the probabilities for each instance with respect of its class. Common ways used for estimating the probabilities distribution using the instances and the predicted class are maximum entropy and discriminant analysis.

The most representative technique in statistical learning algorithms is Bayesian Networks (BN) [27] which are graphical models for probability relationships among a set of features. This graphical model is a directed acyclic graph where nodes are the features and the edges represent a relation between features. Thus, the learning task in BN is based on the determination of the structure of the network as well as the and the of the parameters for each class. These parameters are encoded into tables for each feature. Instance-based learning is another subcategory of statistical learning algorithms. The key characteristic of this category is the lazy-learning procedure as they delay the induction until classification is performed. The most common algorithms in this category are nearest neighbour algorithm that can predict the class of a given instance based on nearest classified instance that was previously classified. The advantage of these algorithms is reduced training time. On the other hand, they require more time to classify instances during the classification phase.

4.2.2. Selected Classification Algorithms

Algorithm selection for evaluation was performed based on our observations of the described characteristics of the collected datasets. These characteristics show that the datasets used are: (i) static and (ii) contain nominal attributes. Particularly, static datasets cannot be used in online learning algorithms. Moreover, the same algorithms where used in both testbed and simulation experiments for comparison reasons, despite the fact that the dataset generated in real-time testbed is larger than the one derived via simulation. Given our datasets characteristics, we use and evaluate the following supervised classification algorithms:

- J48: is a tree classification algorithm that uses as part of training of the model a pruned or unpruned C4.5 decision tree for classification [28]. J48 provides a fast decision tree that deals with missing values and perform well over nominal datasets.

- Naive Bayes (NB): is a Bayesian network with a single parent (unobserved node) connteced to children nodes being the observed nodes [29]. The strong assumption made for this type of networks is that children nodes are independent of the parent and the estimation is done with respect to the probabilistic model for features of an unknown class. NB requires less data and therefore fits well with our datasets. Moreover, as we have static datasets, this classifier seems to be a more reasonable choice for our evaluation procedure.

- Multilayered Perceptron (MLP): is an ANN classifier that is trained using Back Propagation (BP) algorithm that aims to find the parameters (in weights) for a set of inputs. A multilayer neural network consists of large number of units (neurons), joined together in connections which forward information from input layers to the output layers [30]. The classification class is encoded at the output layers, which are consisted of neurons, representing the predicted class of an instance.

- Multinomial Logistic Regression (MLG): is an ANN supervised classifier that builds a model during training which classify instances using multinomial logistic regression of a ridge estimator [31]. Specifically, MLG is trained by solving the problem of maximising the likelihood estimator (MLE) for the ridge estimator. Finding this value, allows the classifier to associate features describing instances with their respectively target class.

The execution classes are used to generate values for descriptive measurements of the network's behaviour. For the evaluation we used WEKA (Waikato Environment for Knowledge Analysis) [32] tool. This is a widely used tool that enjoys widespread acceptance in both academia and industry.

4.3. Selected Algorithms Execution Parameters

For the machine learning classification algorithms selected, after extensive experimentation we choose a set of execution parameters values based on their performance. Specifically, for each selected parameter value, we focus on the ones providing the best results in terms of accuracy, for the particular problem considered.

For the sake of reproducability of our experiments we report all considered execution parameters in Appendix A. In this appendix section we provide a brief description of the execution parameters for the selected classification algorithms, along with the values selected for the experimental setup.

4.4. Evaluation Metrics

A classifier's evaluation, is typically performed on prediction accuracy. The prediction accuracy is calculated after the splitting of a dataset into training and testing sets. The training set is used to build a classifier model which learns from the dataset instances. The testing set is used for estimating the performance of the classifier. The predicted accuracy is calculated as the percentage of the correct prediction divided by the total number of predictions. This metric can also be used for comparison between different classifiers with respect to specific cases. Consequently, it is easy to compute an average value to describe the overall prediction performance of a classifier. In our approach, each generated dataset (simulation-based and testbed-based) is splitted into 75% for the training set and 25% for the testing set, as this is derived after a sensitivity analysis on training set percentage split on accuracy of classification. The following evaluation metrics are used:

Table 1: Classification Matrix Example

	Predicted Class	
Actual Class	**Yes**	**No**
Yes	a	b
No	c	d

- *Observed accuracy* (p_o): counts the number of correct predictions over the total set of predictions performed by the classifier. This is a percentage metric (%). By splitting our dataset into train and test set, the use of accuracy metric is necessary.

- *Kappa Statistic Value* (κ): a metric that takes into consideration unbalancing in datasets, and is a normalized value between 0 and 1. In our case, unbalancing is observed due to larger number of normal instances in the datasets, as normal behaviour is the one captured more often. This value is computed as shown below, where p_e is the *expected accuracy*: $\kappa = \frac{p_o - p_e}{1 - p_e}$

- *Precision (pr):* Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. False positives are cases the model incorrectly labels as positive that are actually negative. Precision expresses the proportion of the data points our model says was relevant actually were relevant.

- *Recall (re):* Recall is defined as the number of true positives divided by the number of true positives plus the number of false negatives. False negatives are cases the model incorrectly labels as negative that are actually positive. Recall expresses the ability to find all relevant instances in a dataset.

To explain how Observed Accuracy (p_o), Expected Accuracy (p_e), Precision (pr) and Recall (re) are computed we use a simple example with two classes, as given in Table 1. Values a,d represent the correct positive and negative predictions (True Positive, True Negative), respectively and values c, b represent the incorrect positive and negative predictions (False Positive, False Negative), respectively. The computation of Observed Accuracy (p_o), Expected Accuracy (p_e), Precision (pr) and Recall (re) is given by the following equations:

$$Precision : pr = \frac{a}{a+c}$$
$$Recall : re = \frac{a}{a+b}$$
$$Observed\_Accuracy : p_o = \frac{a+d}{a+b+c+d}$$
$$Expected\_Accuracy : p_e = p_{yes} + p_{no}$$
$$p_{yes} = \frac{a+b}{a+b+c+d} \cdot \frac{a+c}{a+b+c+d}$$
$$p_{no} = \frac{c+d}{a+b+c+d} \cdot \frac{b+d}{a+b+c+d}$$

Table 2: Sensitivity Analysis Matrix on Training Set Percentage Split on Accuracy of the Classification (Simulation)

Training	**CLASSIFICATION ALGORITHM (A)**				
Set (%)	**J48**	**NB**	**MLP**	**MLG**	**AVERAGE**
65	87.78	87.57	88.02	85.30	87.17
70	88.55	85.83	88.03	85.50	86.98
75	89.62	88.20	88.69	86.38	**88.22**
80	88.10	87.25	88.72	86.50	87.64

Table 3: Observed Accuracy (p_o) & kappa value (k) Evaluation Matrix (Simulation Cases)

		CLASSIFICATION ALGORITHM (A)			
		J48 (p_o, k)	NB (p_o, k)	MLP (p_o, k)	MLG (p_o, k)
CASE	Case #1	(82.76, 0.72)	(89.66, 0.83)	(80.00, 0.72)	(75.86, 0.61)
	Case #2	(100.0, 1.00)	(94.40, 0.89)	(94.40, 0.89)	(94.40, 0.89)
	Case #3	(86.11, 0.82)	(80.55, 0.75)	(91.67, 0.89)	(88.89, 0.86)
	Average	**(89.62, 0.85)**	(88.20, 0.82)	(88.69, 0.83)	(86.38, 0.79)

Table 4: Precision (pr) & Recall (re) Evaluation Matrix (Simulation Cases)

		CLASSIFICATION ALGORITHM (A)			
		J48 (pr, re)	NB (pr, re)	MLP (pr, re)	MLG (pr, re)
CASE	Case #1	(0.837, 0.828)	(0.960, 0.897)	(0.793, 0.793)	(0.761, 0.759)
	Case #2	(1.000, 1.000)	(0.972, 0.944)	(0.972, 0.944)	(0.972, 0.944)
	Case #3	(0.880, 0.861)	(0.819, 0.806)	(0.935, 0.917)	(0.903, 0.889)
	Average	(0.906, 0.896)	(0.917, 0.882)	(0.900, 0.885)	(0.879, 0.864)

Table 5: Differentiation Results for J48 & Case #1 (Simulation Case)

$\bigoplus_{J48}$	w_t^N	w_t^{F1}	w_t^{F2}	w_t^{F3}
w_t^N	-	T	T	T
w_t^{F1}	T	-	T	F
w_t^{F2}	T	T	-	T
w_t^{F3}	T	F	T	-

5. Simulation Implementation & Results

The EASH system model was simulated using the OPNET Simulator. For communication purposes, the ZigBee protocol was used. In order to form our star-topology, we use ZigBee End Devices (ZED) for the peripheral nodes and a ZigBee Coordinator (ZC) for the coordinating node. Our star-topology consists of peripheral sensing nodes, each one responsible for the monitoring of a specific appliance in the system. The coordinator is responsible for the initialisation of the network and for the reception of the measurements by the peripheral nodes. After the initialisation is performed in all the simulates scenarios nodes are configured to transmit their measurements every minute. The different scenarios we simulate are associated with the attack and fault executions of the classes as those are described in Section 4.1. In particular, F1 was simulated by lowering a single node's energy below a transmission threshold; F2 was simulated by changing the destination of each packet; F3 was simulated by adding noise to the packet rendering the data load invalid and causing the receiver to drop the packets. Attack classes were simulated by entering a middle node interrupting communication between a single peripheral node and the central coordinator. In A1 the attacker is re-transmitting (without modifying) the packets received from the transmitting node. In A2, the packets to be transmitted are dropped, and in A3 the payload of the packet is increased before re-transmitting to the central coordinator. A detailed analysis of the experimental setup for our simulation toolchain given in Appendix B.

From the various simulated scenarios we extracted a total of twenty five network characteristics (s.t., throughput, packet delays, etc.) to generate instances each associated with some execution class (i.e., normal, fault, or attack). The set of these instances yield our complete dataset. A total of 144 instances were generated after the execution of the simulated scenarios.

In order to choose the percentage split between training and testing dataset we performed a sensitivity analysis on training set size and accuracy. We examined percentages of 65%, 70%, 75% and 80% for training datasets, as we aim to keep a split that will not lead to overfitting of our models. Results are presented in Table 2. We observe that the training set percentage that derives the highest average accuracy (**88.22%**) is 75%. Based on this observation the evaluation is performed using this percentage as part of training data, leading the rest of 25% used as testing data. The experimental cases for evaluation considered in simulation experimental environment are: Case # 1: Normal Vs Every Fault (N vs F1 F2 F3), Case # 2: Normal Vs Every Attack (N vs A1 A2 A3), Case # 3: Normal Vs Every Attack Vs Every Fault (N vs F1 F2 F3 A1 A2 A3). Classification results are presented in Table 3. For each ML algorithm considered, we list the observed accuracy and kappa value as a pair (p_o, k).

Additionally, precision and recall metrics for same algorithms and evaluation cases are presented in Table 4 as pairs (pr, re), as well as their average performance value, in the last row. We can observe that results are promising as in all four algorithms the average accuracy is above 86%. On average accuracy, algorithm J48 slightly outperforms the other algorithms while Precision and Recall average values for it are 0.906 and 0.896, respectively. NB, is the algorithm with a highest average precision value of 0.917, but still this is due to a higher value in Case #1, in the rest of the metrics J48 outperforms the other algorithms.

We evaluated the class differentiation operator for each algorithm, as defined in Definition 2.1. Table 5 presents such results for the J48 algorithm and Case #1. A T appears in a cell if the algorithm can differentiate the two classes corresponding on the row and column of the cell, and F if the algorithm failed to differentiate the two classes. Differentiation does not apply for states of the same class (in the diagonal, denoted by -). For the J48 classifier and Case #1, all instances were classified correctly except for the Low Energy Fault (F1) instances which were misclassified as Packet Dropped Failure (F3) instances and vice-versa. In a similar manner, we have constructed tables for each of the algorithms and cases considered.

6. Testbed Implementation & Results

In addition to the simulation framework, we have also developed a real-time testbed, consisting of real sensing units. In particular, we used three Raspberry Pi 3 Model B+ connected via Bluetooth to the three Sensor Tag CC2650 for sensing humidity, temperature values (peripheral nodes), a Macbook Pro for collecting the transmitted values and running a Wireshark

Table 6: Sensitivity Analysis Matrix on Training Set Percentage Split on Accuracy of the Classification (Testbed)

Training	CLASSIFICATION ALGORITHM (A)				
Set (%)	**J48**	**NB**	**MLP**	**MLG**	**AVERAGE**
65	89.25	89.12	92.14	87.69	89.55
70	89.10	89.20	92.23	86.82	89.34
75	89.64	89.58	92.76	87.33	**89.83**
80	88.14	89.02	92.32	86.61	89.02

Table 7: Observed Accuracy (p_o) & kappa value (k) Evaluation Matrix (Testbed Cases)

		CLASSIFICATION ALGORITHM (A)			
		J48 (p_o, k)	NB (p_o, k)	MLP (p_o, k)	MLG (p_o, k)
CASE	Case #1	(96.36, 0.94)	(92.73, 0.87)	(98.18, 0.97)	(96.37, 0.93)
	Case #2	(95.83, 0.93)	(97.92, 0.96)	(97.91, 0.96)	(91.66, 0.84)
	Case #3	(76.71, 0.68)	(78.08, 0.70)	(82.19, 0.75)	(73.97, 0.64)
	Average	(89.64, 0.85)	(89.58, 0.84)	**(92.76, 0.89)**	(87.33, 0.80)

Table 8: Precision (pr) & Recall (re) Evaluation Matrix (Testbed Cases)

		CLASSIFICATION ALGORITHM (A)			
		J48 (pr, re)	NB (pr, re)	MLP (pr, re)	MLG (pr, re)
CASE	Case #1	(0.963, 0.964)	(0.929, 0.928)	(0.979, 0.978)	(0.963, 0.964)
	Case #2	(0.974, 0.959)	(0.976, 0.975)	(0.973, 0.975)	(0.933, 0.918)
	Case #3	(0.785, 0.769)	(0.804, 0.782)	(0.911, 0.823)	(0.715, 0.735)
	Average	(0.907, 0.897)	(0.903, 0.895)	(0.954, 0.925)	(0.870, 0.872)

Table 9: Differentiation Results for J48 & Case #1 (TestBed Case)

$\bigoplus_{J48}$	w_t^N	w_t^{F1}	w_t^{F3}
w_t^N	-	T	T
w_t^{F1}	T	-	F
w_t^{F3}	T	F	-

sniffing tool (central node), and an Ubuntu PC to issue the man in the middle attacks (attacker). We formed a star-based topology, as it was introduced in section 2.

For the real time testbed, the abnormality classes implemented for faults and attacks are: Low Energy Failure (F1) and Packet Dropped Failure (F3) for the set of faulty classes (F) and Sink Hole Attack (A2), Message Modification (A3) for the set of attack classes (A). The normal class and scenario were implemented by having the peripheral nodes (formed by RPis) to receive humidity and temperature from the sensors and then generate TCP packets for that measurements in order to be transmitted to the central node.

The dataset is generated after an inspect is performed on the extracted TCP packets captured by the Wireshark network protocol analyser. These are the packets exchanged between the peripheral nodes and the coordinator node. For the instances generation we only keep, as comma separated values (csv), parameters of TCP communication which had values with a large range of variation. Each analysed packet with its related descriptive characteristics values was associated with one of the execution classes we considered in this work. Low Energy Failure (F1) was emulated by interrupting a single peripheral node's communication by cutting off its power. Packet Dropped Failure (F3) was emulated by setting a threshold on the packet generation procedure indicating the quality of packets being generated. If the value of a generated packet was above the threshold, then the packet was transmitted; otherwise the packet was dropped. To implement the attack classes, we initiated a Man In the Middle Attack (MITM), where the attacker interrupts the communication between a peripheral and the central node by receiving the packets of their communication. This attack was implemented using the Address Resolution Protocol (ARP) spoofing (poisoning) method that allows the attacker to alter the ARP tables on the communicating nodes by sending (spoofed) ARP messages. The purpose of this attack is to associate the attacker MAC address with the IP address of another host.

In our implementation the attacker associated its MAC address with the default gateway. This attack allows the attacker to intercept the data frames on the network, to modify the traffic, or stop all traffic entirely. By intercepting the channel between a peripheral and the central node we emulate the behaviour of A2, by stopping all the traffic in the network, and of A3, by modifying the measurements in the packets transmitted. Using Wireshark, we collected a total of twenty four characteristics (e.g, time sent, origin, destination, protocol etc.) for each message sent, forming comma separated instances for our dataset. Each instance is associated with the execution class emulated at the time the packet was captured. The total amount of instances we collected for the real time dataset was 589.

Similarly, as in the case of simulation environment dataset, we performed a sensitivity analysis of the training set size on the accuracy of the classification. The same percentages for training data of 65%, 70%, 75% and 80% were considered. Results are presented in Table 6. We observe that in this evaluation the average accuracy is approximately the same among the various percentages considered. Even though, 75% for training data shows the highest average accuracy (**89.83%**). As a result, we use 75% of the dataset as training data and the rest 25% of the dataset as testing data, similarly as in the simulation evaluation. Experiments were performed for the following cases: Case # 1: Normal Vs Every Fault (N vs F1 F3), Case # 2: Normal Vs Every Attack (N vs A2 A3), Case # 3: Normal Vs Every Attack Vs Every Fault (N vs F1 F3 A2 A3). Similarly to the simulation environment, we generate the evaluation matrix using the same supervised ML classification algorithms the cases mentioned above, and the same evaluation metrics.

Table 7 lists the observed accuracy and kappa value as a pair (p_o, k) and Table 8 lists the precision and recall value as pairs (pr, re). On the average, all algorithms have an accuracy of above 87%. Moreover, the MLP classifier outperforms every other algorithm, on average. In terms of accuracy, kappa value, precision and recall the average performance values for MLP are 92.76%, 0.89, 0.954 and 0.925, respectively. MLP takes advantage of the larger dataset which can be generated in the testbed environment to train a better model. This also occurs for some of the other classifiers in some of the cases, when comparing the simulation results with the testbed results.

We also evaluate the class differentiation operator for each algorithm and a representative table is presented in Table 9. Based on this table we observe that F1 and F3 instances are still misclassicified (as it was the case in the simulation environment).

7. Classification Results Analysis

This section provides experimental analysis of classification results obtained form the previous sections. This analysis is splitted into two parts. The first part is used for the similarity analysis of abnormality classes (faults, attacks) and the second part we conducted a feature based analysis. The analysis is performed on both experimental datasets (simulated-based and testbed-based). Experimental classes considered for this analysis are those in common in both datasets. These are: Normal (N), Low Energy Failure (F1), Packet Dropped Failure (F3), Replay Attack (A2) and the third attack of Message Modification Attack (A3).

Specifically, in the first part of this analysis (Section 7.1), we aim to discover the abnormality classes classified in a similar way, for a model trained to differentiate between the normal and a single abnormal class. Abnormality classes that are classified in similar way indicate pairs that require additional information (in terms of their described features), in order to be correctly classified. The second part of the section (Section 7.2) aims to give an inside on the classification performed by a specific algorithm (MLP), by ranking the datasets features according to their significant in the classification. To do so, we perform a features ranking evaluation using two common features ranking schemes of Gain Ratio and Information Gain.

7.1. Similarity Analysis Of Abnormality Classes

In this first part of our analysis we study how pairs of abnormality classes (faults and attacks) can be classified in a similar way by an algorithm due to common features they share. Our experimentation methodology has as follows. We prepare a training set consisted of normal and a faulty class F_i instances only. As previously noted, the faulty classes we consider for this analysis are F1 & F3. Using this training set we then train a model using MLP. Evaluation, is performed by using only attack instances from A2 & A3. The model has to classify those attack instances in one of the available trained classes Normal or Abnormal denoted by the trained faulty class.

Trained data instances are distributed in classification areas as specified by the trained model. Experimental results are presented in Figure 4, where the classification areas are specified by normal, faulty and attack instances. The blue (placed in the bottom left area of figure) and red (placed in the top right area of figure) (marked with symbol 'x') form the classification areas of the normal and faulty classes (resp.) in each scenario. The instances denoted by the "square" symbol show how the attack class instances are classified. If in a particular scenario the attack instances are classified as normal or as a fault instances in both experimental settings (simulation and testbed – see Figure 4(a)) then either: (a) the attack instances are classified under the normal class and hence can be differentiated from fault instances in both settings; or, (b) the attack instances are classified under the fault class and thus the attack class cannot be differentiated from the fault class. Finally, when the attack instances are classified differently in the two settings for a particular scenario (i.e. as normal class in the simulation setting and as attack class in the testbed setting – see Figure 4(d)), then we conduct further tests to determine the features that may allow the class differentiation.

NF1 models (Figures 4a,4b): We observe that both attack class instances (A2 & A3) for both datasets are classified in the classification area of fault F1. As a result, we conclude that the feature set we considered was not sufficient to differentiate the two attack classes from F1.

NF3 models (Figures 4c,4d): In this case we can observe that attack class instances (A2 & A3) are classified in different classification areas over simulation and testbed settings. In particular, instances of classes A2 and A3 are classified under the F3 fault class in the simulation and testbed settings respectively. By closer examination, it appears that A2 instances could not be differentiated due to lack of TCP features such as sequence number (SEQ). On the other hand, A3 instances could not be differentiated due to lack of features describing global conditions in the network such us Throughput and Delay. So we re-evaluated the two scenarios by re-training that models with less or more features.

NF3A2 Re evaluation: Figure 5 presents results on re evaluation of NF3 model for classifying A2 instances from the simulation dataset by: (i) adding SEQ feature, (ii) removing global features and (iii) keeping only the ten most significant features. In all three experiments, results are improved as A2 instances are differentiated from F3 and classified in normal classification area.

NF3A3 Re evaluation: Figure 6 shows results on re-training of NF3 model for classifying A3 instances from the testbed dataset by adding: (i) Delay feature, (ii) Throughput Feature and (iii) Delay and Throughput Feature. In all three experiments, results are not improved as A2 instances are not differentiated by F3 and classified to that classification area. Thus, extra features might need to be considered in order to reach a differentiation between those classes (F3 & A3). Similar experiments can be done over the same experimental settings by using either a different classification algorithm or by adding or removing datasets features.

7.2. Feature Based Analysis

In this section, we perform an evaluation on the significance of features in respect to the classification performed over the evaluation cases of our datasets in both settings, using the MLP classification algorithm. Before we step into the obtained results, we provide a description of the feature ranking schemes used for evaluation, and how this was conducted. the feature ranking schemes used are Gain Ratio and Information Gain. Both of them, rank the significance of a feature by computing how much it reduce the entropy of the predicted class. The range of entropy value is based on the number of classes under which the evaluation is performed. By having, the ranking value from both schemes we compute an average for each feature and we sort this average value in order to derived the five most significant features of the evaluation cases for both datasets.

Experimental cases used for evaluation are: Case # 1: Normal Vs Every Fault (N vs F1 F3), Case # 2: Normal Vs Every Attack (N vs A2 A3), Case # 3: Normal Vs Every Attack Vs Every Fault (N vs F1 F3 A2 A3). Figure **??** shows this ranking charts performed over the experimental cases and datasets, while top five features extracted per case and dataset are reported in Table 10. Based on each feature characteristic and following the ranking charts presented in Figure **??** the following remarks can be conducted, on the way the differentiation

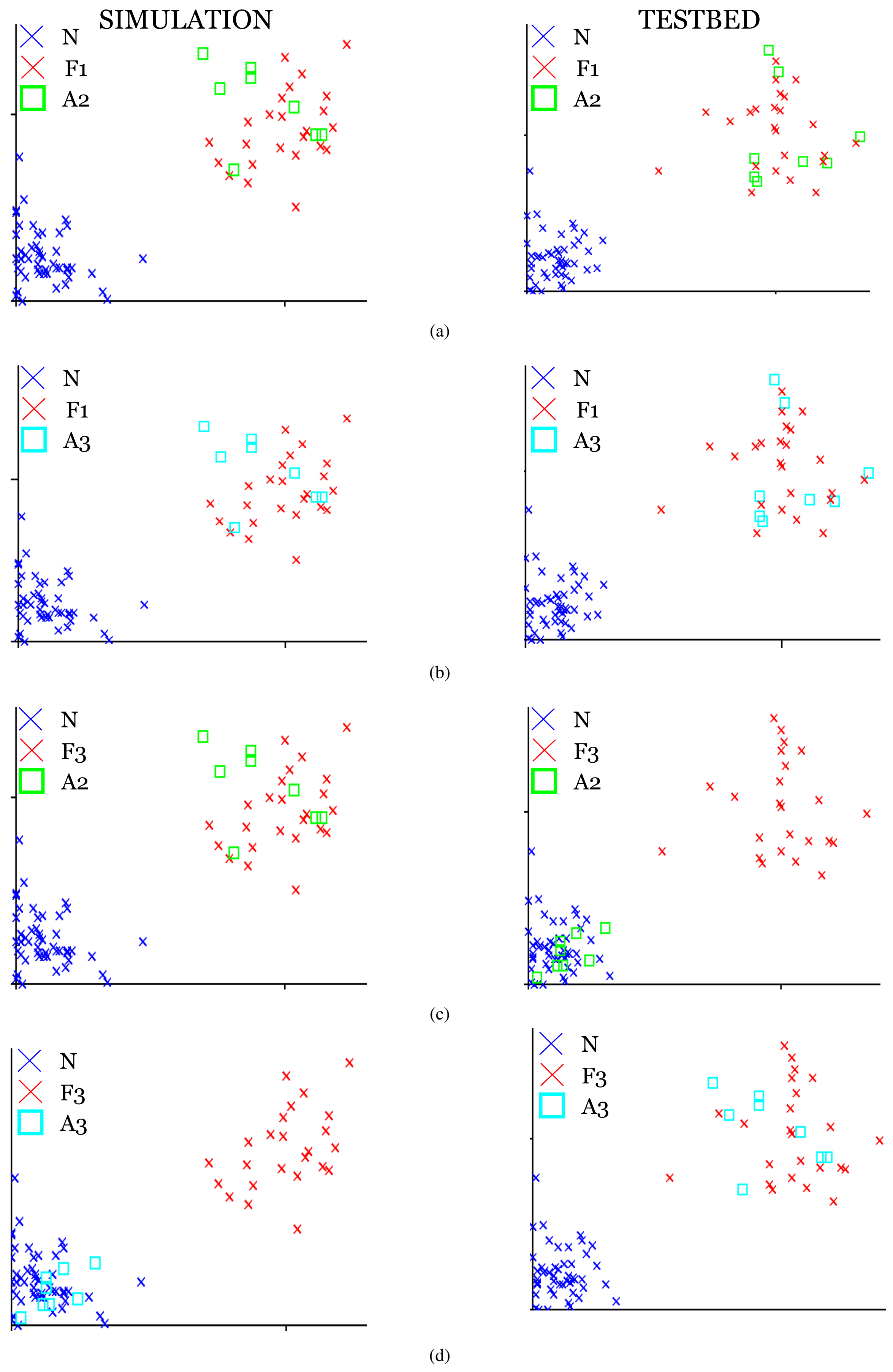
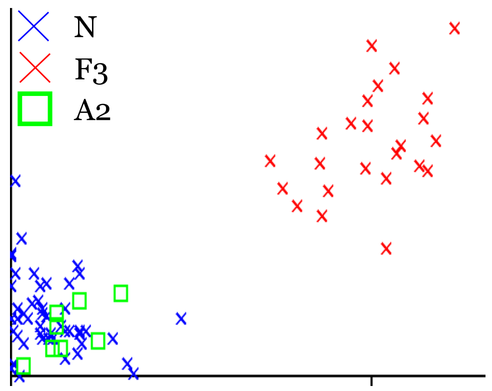
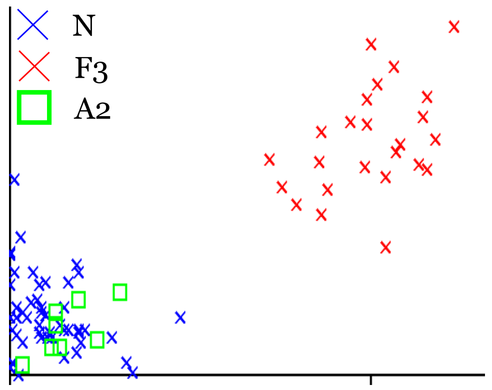


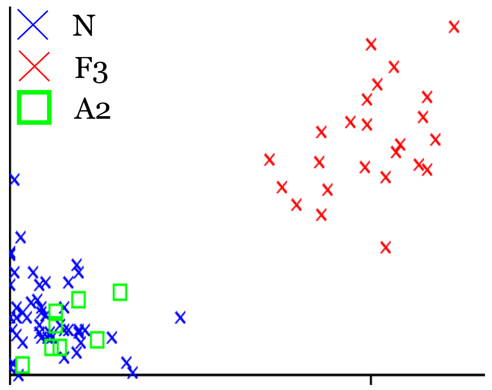
Figure 4: Simulation (Left Column) & Testbed (Right Column) Classification Areas for classifying attack instances on models trained with Normal and a Fault class. (a) Classifying A2 instances on N & F1 model (b) Classifying A3 instances on N & F1 model (c) Classifying A2 instances on N & F3 model (d) Classifying A3 instances on N & F3 model



(i)

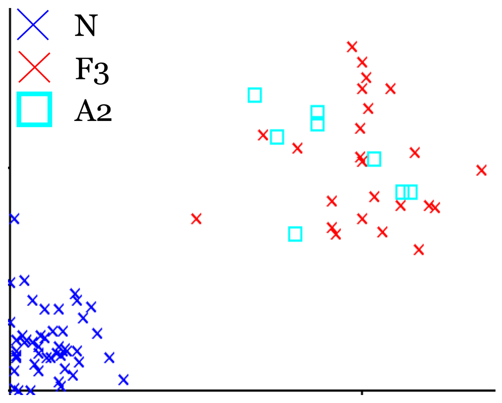


(ii)

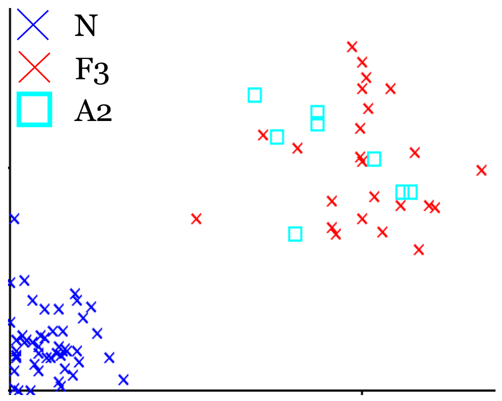


(iii)

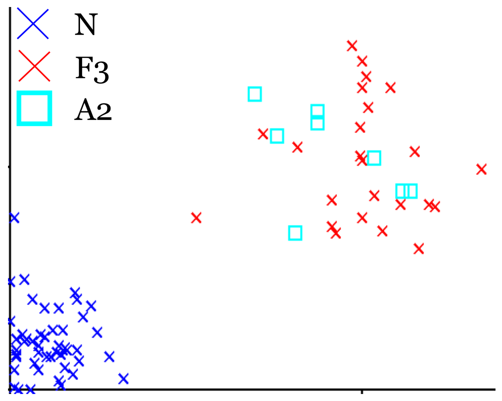
Figure 5: Classification Results on reevaluation of NF3 model (Simulation Dataset) Figure 4c, with (i) SEQ feature addition (ii) Global Features Removal (iii) TOP 10 Features



(i)



(ii)



(iii)

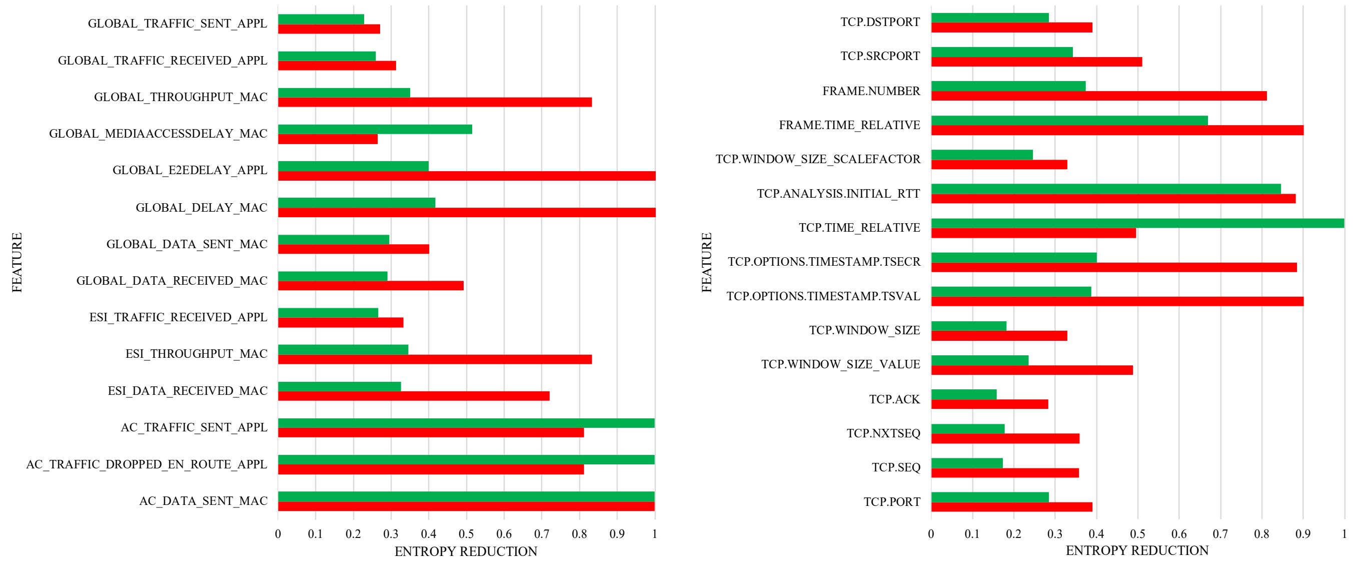
Figure 6: Classification Results on reevaluation of NF3 model (Testbed Dataset) Figure 4d, (i) Delay (ii) Throughput (iii) Delay & Throughput

task is performed for the evaluation cases of both datasets:

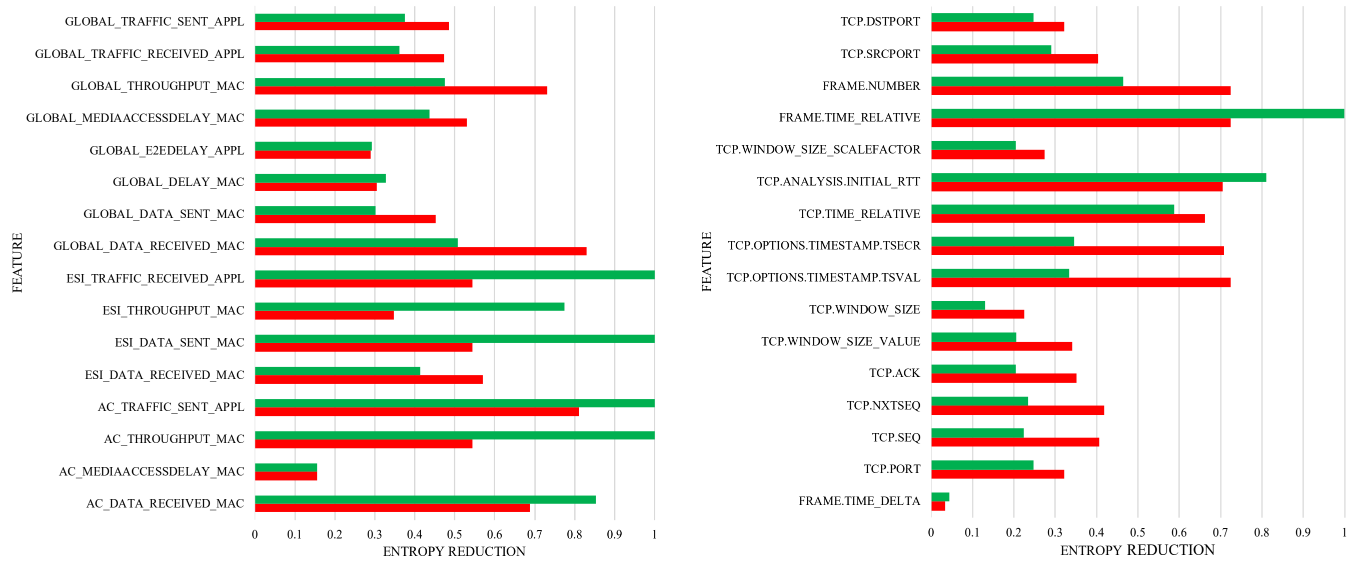
- Case #1 (Figure 7a): For simulation dataset instances, differentiation between normal and faulty classes is performed by focusing on features such as traffic generated, sent and dropped data, while also on channel characteristics like overall delay in the network. For the Testbed dataset differentiation is performed based on time-based features such as the relative time of each frame and the round trip time, similar to the delay feature of simulation dataset.

- Case #2 (Figure 7b): For simulation dataset instances, differentiation between normal and attack classes is performed on features related to the traffic generated, sent and received by the nodes of the network. For the testbed dataset instances, differentiation task is derived by examining time related features such as round trip time and relative time, as well as the number of frames, similar to the throughput feature of simulation dataset.

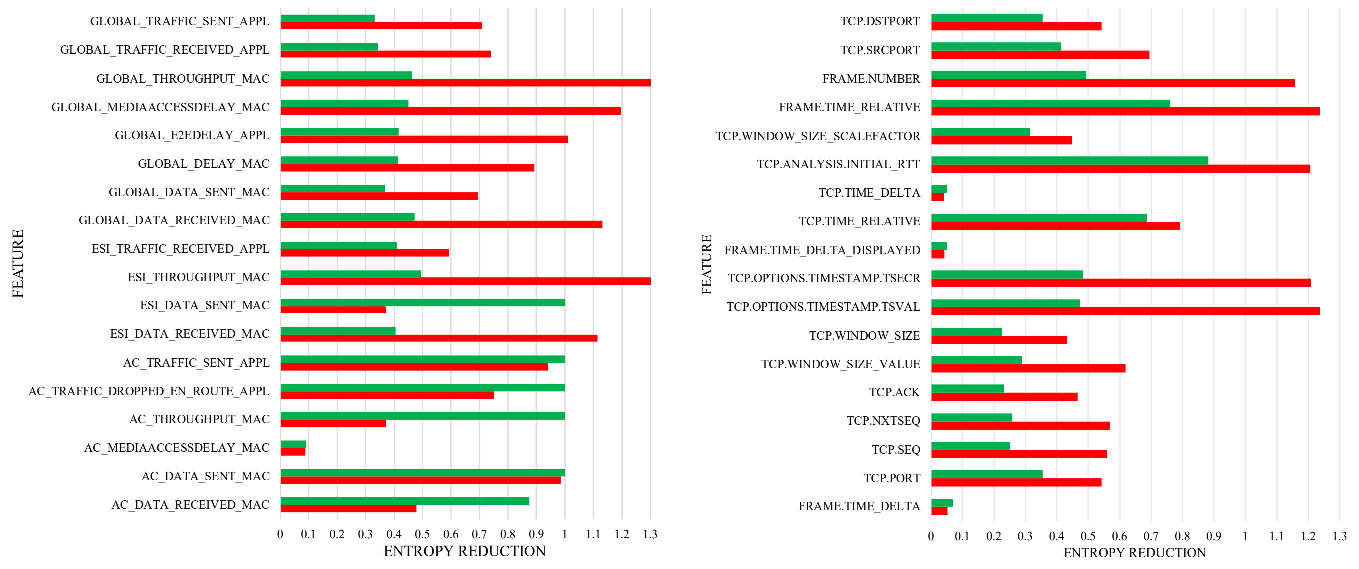
- Case #3 (Figure 7c): For simulation dataset instances, the differentiation performed over normal, attack and the fault classes is mainly performed by focusing on node features such as the traffic generated, packets received and dropped. For the Testbed dataset, differentiation can is performed by examining time related features such as round trip time, relative time, and frame number, similar to traffic generated feature of the simulation dataset.



(a)



(b)



(c) Features Ranking using Gain Ratio (Red/Bar Below) & Information Gain (Green/Bar Above) Over Cases For Simulation Dataset (SD Left Column) & Testbed Dataset (TD Right Column). (a) Case #1 (Normal Vs Faults), (b) Case #2 (Normal Vs Attacks), (c) Case #3 (Normal Vs Faults Vs Attacks)

Table 10: TOP 5 Features Per Dataset Case

Simulation Dataset (SD)		Testbed Dataset (TD)	
Feature	Average Value	Feature	Average Value
Case #1			
AC DATA SENT MAC	1	TCP ANALYSIS INITIAL RTT	0.8645
AC TRAFFIC DROPPED	0.90565	FRAME TIME RELATIVE	0.7855
AC TRAFFIC SENT APPL	0.90565	TCP TIME RELATIVE	0.7475
GLOBAL DELAY MAC	0.74495	TSVAL	0.644
GLOBAL E2EDELAY APPL	0.71985	TSECR	0.643
Case #2			
AC TRAFFIC SENT APPL	0.9055	TIME RELATIVE	0.86245
AC THROUGHPUT MAC	0.772	TCP ANALYSIS INITIAL RTT	0.75755
ESI DATA SENT MAC	0.772	TCP TIME RELATIVE	0.62555
ESI TRAFFIC RECEIVED APPL	0.772	FRAME NUMBER	0.59465
AC DATA RECEIVED MAC	0.77	TSVAL	0.52905
Case #3			
AC DATA SENT MAC	0.9926	TCP ANALYSIS INITIAL RTT	1.0442
ESI THROUGHPUT MAC	0.97135	FRAME TIME RELATIVE	0.9989
AC TRAFFIC SENT APPL	0.97015	TSVAL	0.8555
GLOBAL THROUGHPUT MAC	0.9382	TSECR	0.8461
AC TRAFFIC DROPPED	0.8748	FRAME NUMBER	0.8257

We noticed that some traffic features (e.g. THROUGHPUT, FRAME NUMBER) and delay features (e.g. DELAY, RTT) are significant and contribute to the differentiation task in the same cases and for both datasets (Simulation & Testbed). There are also features, such as Traffic Dropped in simulation dataset and Relative Time in testbed dataset, that are significant for the differentiation task for the one dataset but not the other. This was expected as the conditions of the experiments differ from simulation to testbed.

There are also features, such as Traffic Dropped in simulation dataset and Relative Time in testbed dataset, that are significant for the differentiation task for the one dataset but not the other. This was expected as the conditions of the experiments differ from simulation to testbed. Finally based on the top five features per case and dataset, given in Table 10, we perform a dimensional reduction on the number of the features of both datasets in order to reevaluate the classification performance of the algorithms (J48, MLG, NB, MLP), in terms of their observed accuracy. Observed accuracy was significantly improved in all the four algorithms. For the most important case, (Case #3), the accuracy was improved for simulation dataset into 98.00% for all classification algorithms while for the same case and testbed dataset the accuracy was improved to 87.0748% for J48 and MLP and to 86.395% for NB and MLG.

8. Conclusion

This paper presents results on the differentiation task between abnormalities affecting an EASH system. The abnormalities examined in this work can be generated by either fault components or network attacks. The relation between these abnormality classes and their effects on the communication channel of the system was studied and a differentiation operator has been defined. We used ML-based approaches to achieve differentiation through classification. Our obtained results show that the use of supervised machine learning algorithms is a promising approach as it successfully achieves to differentiate between faulty and attack classes with high accuracy rate. Misclassifications for cases with similar impact on the network where further analysed, in both experimental settings (simulation and real-time testbed), for the two abnormal classes as well as the considered features. Our analysis suggests that classification results can be further improved from the addition/removal of features from the descriptive datasets.

Acknowledgements

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) and from of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

References

[1] L. Shi, Q. Dai, and Y. Ni, "Cyber–physical interactions in power systems: A review of models, methods, and applications," *Electric Power Systems Research*, vol. 163, pp. 396–412, 2018.

[2] G. Wen, W. Yu, X. Yu, and J. Lü, "Complex cyber-physical networks: From cybersecurity to security control," *Journal of Systems Science and Complexity*, vol. 30, no. 1, pp. 46–67, 2017.

[3] J. R. Lindsay, "Stuxnet and the limits of cyber warfare," *Security Studies*, vol. 22, no. 3, pp. 365–404, 2013.

[4] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011.

[5] E. E. Miciolino, R. Setola, G. Bernieri, S. Panzieri, F. Pascucci, and M. M. Polycarpou, "Fault diagnosis and network anomaly detection in water infrastructures," *IEEE Design & Test*, vol. 34, no. 4, pp. 44–51, 2017.

[6] G. Cardoso, J. G. Rolim, and H. H. Zurn, "Identifying the primary fault section after contingencies in bulk power systems," *IEEE Transactions on Power Delivery*, vol. 23, no. 3, pp. 1335–1342, 2008.

[7] D. Kim, S. C. Han, Y. Lin, B. H. Kang, and S. Lee, "Rdr-based knowledge based system to the failure detection in industrial cyber physical systems," *Knowledge-Based Systems*, vol. 150, pp. 1–13, 2018.

[8] R. Mahanty and P. D. Gupta, "Application of rbf neural network to fault classification and location in transmission lines," *IEE Proceedings-Generation, Transmission and Distribution*, vol. 151, no. 2, pp. 201–212, 2004.

[9] D. Thukaram, H. Khincha, and H. Vijaynarasimha, "Artificial neural network and support vector machine approach for locating faults in radial distribution systems," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 710–721, 2005.

[10] A. D. Chhokra, N. Mahadevan, A. Dubey, S. Hasan, D. Balasubramanian, and G. Karsai, "Hierarchical reasoning about faults in cyber-physical energy systems using temporal causal diagrams," *System*, vol. 1, no. S1, p. P1, 2017.

[11] H. Kang, J. Cheng, I. Kim, and G. Vachtsevanos, "An application of fuzzy logic and dempster-shafer theory to failure detection and identification," in *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*. IEEE, 1991, pp. 1555–1560.

[12] C. Konstantinou, M. Maniatakos, F. Saqib, S. Hu, J. Plusquellic, and Y. Jin, "Cyber-physical systems: A security perspective," in *Test Symposium (ETS), 2015 20th IEEE European*. IEEE, 2015, pp. 1–8.

[13] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry *et al.*, "Challenges for securing cyber physical systems," in *Workshop on future directions in cyber-physical systems security*, vol. 5, 2009.

[14] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *2008 The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, 2008, pp. 495–500.

[15] N. Komninos, E. Philippou, and A. Pitsillides, "Survey in smart grid and smart home security: Issues, challenges and countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1933–1954, 2014.

[16] D. I. Urbina, J. Giraldo, A. A. Cardenas, J. Valente, M. Faisal, N. O. Tippenhauer, J. Ruths, R. Candell, and H. Sandberg, *Survey and new directions for physics-based attack detection in control systems*. US Department of Commerce, NIST, 2016.

[17] R. Mitchell and I.-R. Chen, "A survey of intrusion detection techniques for cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 55, 2014.

[18] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 2017, pp. 140–145.

[19] B. B. Zarpelao, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A

survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[20] F. Pasqualetti, F. Dörfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 11, pp. 2715–2729, 2013.

[21] Y. Fujita, T. Namerikawa, and K. Uchida, "Cyber attack detection and faults diagnosis in power networks by using state fault diagnosis matrix," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 398–403.

[22] K. Manandhar, X. Cao, F. Hu, and Y. Liu, "Detection of faults and attacks including false data injection attack in smart grid using kalman filter," *IEEE transactions on control of network systems*, vol. 1, no. 4, pp. 370–379, 2014.

[23] S. Guo, Z. Zhong, and T. He, "Find: faulty node detection for wireless sensor networks," in *Proceedings of the 7th ACM conference on embedded networked sensor systems*, 2009, pp. 253–266.

[24] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[25] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data mining and knowledge discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[26] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[27] F. Jensen, "An introduction to bayesian networks springer," *New York*, 1996.

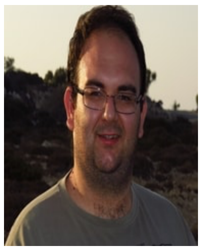
[28] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision tree analysis on j48 algorithm for data mining," *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 6, 2013.

[29] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp. 1310–1315.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[31] D. Böhning, "Multinomial logistic regression algorithm," *Annals of the institute of Statistical Mathematics*, vol. 44, no. 1, pp. 197–200, 1992.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.



Georgios Tertytchny is a PhD student at the Department of Electrical and Computer Engineering at the University of Cyprus. He is also a research assistant at the KIOS Research and Innovation Center of Excellence, also at the University of Cyprus. Georgios has a B.Sc and M.Sc in Computer Science form the Computer Science department of University of Cyprus. His research interests include but not limited to Cyber Physical Systems Security, Machine Learning, Data Mining, Model-based Systems Verification (Formal Methods) and Network Security.

Nicolas Nicolaou is a Senior Research Scientist and a co-founder of Algolysis Ltd (www.algolysis.com), an R&D startup which operates in Cyprus. His main research interests focus on the analysis, design and implementation of practical, robust and secure distributed and parallel algorithms, design and implementation of algorithms for consistent distributed storage systems, blockchains and distributed ledgers, ad-hoc mobile and sensor networks, and security evaluation and exploitation of critical computing equipment (including voting technologies). His research was published in top conferences and journals in the fields of distributed computing, networks and security. For his work he received funding from the Cyprus Research Promotion Foundation (2010-2011 and 2019-2021) and secured an Intra-European Marie-Curie Fellowship (2014-2016). Previously he hold positions as a Senior Researcher in the KIOS Research and Innovation Center of Excellence at the University of Cyprus (2017-2019), as a short-term scholar the CSAIL lab at the Massachusetts Institute of Technology (summer 2017), as a Marie Curie Fellow at IMDEA Networks Research Institute in Madrid (2014-2016) and as a Visiting Lecturer in various Universities in Cyprus and abroad.

Maria K. Michael is an Associate Professor at the Department of Electrical and Computer Engineering at the University of Cyprus. She is also a founding member and the Director of Education and Training at the KIOS Research and Innovation Center of Excellence, also at the University of Cyprus. Maria has a Ph.D. degree from the ECE Dept of Southern Illinois University, Carbondale-USA. Her research expertise falls in the areas of test and reliability of digital circuits and chip-level architectures, with emphasis on embedded and general-purpose multicore systems reliability and on-line testing, dynamic/intelligent parallel CAD algorithms for automatic testing and fault simulation, intelligent methods for design, test and fault tolerance, delay test and emerging fault models. Recent research interests expand to design and optimisation of embedded systems and other chip-level architectures, dynamic self-detecting and self-healing architectures, and dependability and security in the hardware backbone of cyber-physical systems. She has published numerous papers in high-caliber refereed journals and international conferences and she serves on steering, organising and program committees of several IEEE and ACM conferences in the areas of test and reliability. She is a co-recipient of a Best Paper Award of MSE 2009. She is a member of the IEEE and the ACM.

Table 11: J48 Parameters & Selected Values

#	Parameter	Use	Value
1	batchSize	The preferred number of instances to process if batch prediction is being performed	100
2	binarySplits	Whether to use binary splits on nominal attributes when building the tree	false
3	collapseTree	Whether parts are removed that do not reduce training error	true
4	confidenceFactor	The confidence factor used for pruning (smaller values incur more pruning)	0.15
5	debug	If set to true, classifier may output additional info to the console	false
6	doNotCheckCapabilities	If set, classifier capabilities are not checked before classifier is built (use with caution to reduce runtime)	false
7	doNotMakeSplitPointActualValue	If true, the split point is not relocated to an actual data value	false
8	minNumObj	The minimum number of instances per leaf	2
9	numDecimalPlaces	The number of decimal places to be used for the output of number in the model	2
10	numFolds	Determines the amount of data used for reduced-error prunning	3
11	reducedErrorPruning	Whether reduced - error pruning is used instead of C	false
12	saveInstanceData	Whether to save the training data for visualization	false
13	seed	The seed used for randomizing the data when reduced-error pruning is used	0
14	subtreeRaising	Whether to consider the subtree raising operation when pruning	true
15	unpruned	Whether prunning is performed	true
16	useLaplace	Whether counts at leaves are smoothed based on Laplace	false
17	useMDLcorrection	Whether MDL correction is used when finding splits on numeric attributes	false

A. Selected Algorithms Execution Parameters - Values

A.1. J48

Execution parameters for J48 are presented in Table 11. Below we explain the values selected for each parameter:

1. **batchSize:** As we are not using batch prediction, this value will be set to the default value of one hundred (100).

2. **binarySplits:** The process of binary splitting, grows the tree by considering one nominal value versus all other nominal values instead of considering a split on each nominal value individually. This results in a tree where there are only two branches from any node. This method is helpful in cases of large datasets and thus doesn't effect the performance of the algorithm in the case of our dataset and thus the value is set to false, meaning that we do not use it.

3. **collapseTree:** As the removal of parts that do not reduce the training error helps the performance of algorithm to build a smaller model with the same accuracy we set this value to be true.

4. **confidenceFactor:** This determines how aggressive the pruning process will be. The higher this value, the more confident you are that the data you are learning from is a good representation of all possible events, and therefore the less pruning that will occur. Smaller values induce more pruning. This significantly affects classifier performance. As the percentage split, for the training and testing instances is performed randomly the confidence in the learning instances is not high enough and thus we are using a small value of (0.15).

5. **debug:** This does not affect classifier performance thus we use the default value of false.

6. **doNotCheckCapabilities:** As the execution of our algorithms doesn't take much time due to dataset size, we use the false value where the capabilities are checked in advance.

7. **doNotMakeSplitPointActualValue:** We need the relocation to be done on the actual value of data and thus the value of this attribute is set to false.

8. **minNumObj:** This helps to control over fitting in our problem and the number defined by the tool to be used is two (2).

9. **numDecimalPlaces:** Changing this will not affect classifier performance, thus we are using the default value of two (2).

10. **numFolds:** This value determines how much of the data will be used for pruned. Default value of three (3) means that the one third of the data are used for pruned and the other two for growing the tree. A smaller value than three will increase the over fitting in the problem and thus the default value is selected to be used.

Table 12: NaiveBayes (NB) Parameters & Selected Values

#	Parameter	Use	Value
1	batchSize	The preferred number of instances to process if batch prediction is being performed	100
2	debug	If set to true, classifier may output additional info to the console	false
3	displayModelInOldFormat	Use old format for model output	false
4	doNotCheckCapabilities	If set, classifier capabilities are not checked before classifier is built (use with caution to reduce run time)	false
5	numDecimalPlaces	The number of decimal places to be used for the output of number in the model	2
6	useKernelEstimator	Use a kernel estimator for numeric attributes rather than a normal distribution	true
7	useSupervisedDiscretization	Use supervised discretization to convert numeric attributes to nominal ones	true

11. **reducedErrorPruning:** Reduced error pruning is an alternative algorithm for pruning that focuses on minimizing the statistical error of the tree, instead of the misclassification rate. The performance of the classification is not altered and thus we are not using it by setting its value to false.

12. **saveInstanceData:** Changing this will not affect classifier performance, thus we use the default value of false.

13. **seed:** This value is related with the reducedErrorPrunning value and if the value of reducedErrorPrunning is set to be false, this value needs to be zero (0).

14. **subtreeRaising:** This is a specific method of pruning whereby a whole set of branches further down the tree are moved up to replace branches that were grown above it. It should remain true.

15. **unpruned:** We have a small dataset and as we need to avoid, over fitting for our model pruning seems to be useful as it educe the risk of over fitting to the training data. That's why we set the value to be true.

16. **useLaplace:** Laplacian smoothing, add a certain number to all instances in order to eliminate circumstances that are statistically undesirable, such as encountering the number zero. This value is used in the case of predicting probabilities and thus in our case should be false.

17. **useMDLcorrection:** This does not affect classifier performance and thus we use the default value of false.

A.2. NaiveBayes (NB)

Execution parameters for NB are presented in Table 12. Below we explain the values selected for each parameter:

1. **batchSize:** As we are not using batch prediction, this value will be set to the default value of one hundred (100).

2. **debug:** This does not affect classifier performance and thus we use the default value of false.

3. **displayModelInOldFormat:** This does not affect classifier performance and thus we use the default value of false.

4. **doNotCheckCapabilities:** This does not affect classifier performance and thus we use the default value of false.

5. **numDecimalPlaces:** Changing this will not affect classifier performance, thus we are using the default value of two (2).

6. **useKernelEstimator:** In NaiveBayes classification algorithm, when the predictors are non binary there is a need to describe the prior probability other than simply counting. This can be done by two (2) different methods. The first one is by model prior probabilities according a normal distribution or using kernel estimators. In cases where we cannot make assumptions about the normality of this probability the kernel function is used. As we cannot make assumptions about the prior probability the next best thing that can be applied in our case is kernel estimators and thus the value of this attribute is set to true.

7. **useSupervisedDiscretization:** Discretisation can be really helpful in the case of continuous nominal variables. As our attributes are only nominal we use discretisation and the value is set to true.

A.2.1. Multilayered Perceptron (MLP)

Execution parameters for MLP are presented in Table 13. Below we explain the values selected for each parameter:

1. **GUI:** This does not affect classifier performance and thus we use the default value of false.

2. **autoBuild:** This does not affect classifier performance and thus we use the default value of false.

3. **batchSize:** As we are not using batch prediction, this value will be set to the default value of one hundred (100).

4. **debug:** This does not affect classifier performance and thus we use the default value of false.

5. **decay:** Learning rate decay, improves classifier performance in the case of a small datasets for training, as if the learning rate is kept constant and with a high value, the classifier will just learn the instances and not how to generalise. Thus we set the value to true.

Table 13: Multilayered Perceptron (MLP) Parameters & Selected Values

#	Parameter	Use	Value
1	GUI	Brings up a GUI interface	false
2	autoBuild	Adds and connects up hidden layers in the network	false
3	batchSize	The preferred number of instances to process if batch prediction is being performed	100
4	debug	If set to true, classifier may output additional info to the console	false
5	decay	This will cause the learning rate to decrease	true
6	doNotCheckCapabilities	If set, classifier capabilities are not checked before classifier is built (use with caution to reduce runtime)	false
7	hiddenLayers	This defines the hiddenLayers of the network	2
8	learningRate	The amount the weights are updated	0.1
9	momentum	Momentum applied to the weights during updating	0.05
10	nominalToBinaryFilter	This will preprocess the instances with the filter	true
11	normalizeAttributes	This will normalize attributes	true
12	normalizeNumericClass	This will normalize the predicted class if it is numeric	true
13	numDecimalPlaces	The number of decimal places to be used for the output of number in the model	2
14	reset	This will allow the network to reset with a lower learning rate	true
15	seed	The seed used to initialise the random number generator	0
16	trainingTime	The number of epochs to train through	50
17	validationSetSize	The percentage size of the validation set	0
18	validationThreshold	Used to terminate validation testing	0

6. **doNotCheckCapabilities:** This does not affect classifier performance and thus we use the default value of false.

7. **hiddenLayers:** The number of hidden layers in the network, is related with the number of the predictive classes and whether those can be linearly separable. Also, more hidden layers made the classifier model more complicated. The appropriate value is computed by experimental evaluation adding hidden layers to the model until we find that an extra layer add complexity and doesn't offer to accuracy improvement. This was done by setting the value to be two (2).

8. **learningRate:** The learning rate is how quickly a network abandons old beliefs for new ones. This value must be small enough so the model to be trained without memorising only the instances (happens with a small dataset and high learning rate). For that reason, we set the learning rate to be (0.1).

9. **momentum:** Momentum use is similar with that of learningRate. A smaller value than learning rate is required in that case and thus we use (0.05).

10. **nominalToBinaryFilter:** Binary filtering improves the performance of the classifier when the training is done using the sigmoid function. As this is used by default from the classifier in WEKA we are using the default value of true.

11. **normalizeAttributes:** Exactly like nominalToBinaryFilter normalisation of attributes, improves the performance of the classifier and thus the recommended value of true is used.

12. **normalizeNumericClass:** Exactly like nominalToBinaryFilter normalisation of attributes, improves the performance of the classifier and thus the recommended value of true is used.

13. **numDecimalPlaces:** Changing this will not affect classifier performance, thus we are using the default value of two (2)

14. **reset:** Resetting learning rate to a lower value, helps to avoid over fitting in the case of not much training instances. By that, we set this value to be true.

15. **seed:** Changing this will not affect classifier performance, thus we are using the default value of zero (0).

16. **trainingTime:** A large number of epochs can lead the classifier to over fitting when the training instances are not much enough. For that reason we are only using fifty (50) epochs.

17. **validationSetSize:** The validation set size is set to zero (0), as we only experiment using the testing set in the evaluation.

18. **validationThreshold:** Similar as validationSetSize, this value is set to zero (0).

A.3. Multinomial Logistic Regression (MLG)

Execution parameters for MLG are presented in Table 14. Below we explain the values selected for each parameter:

1. **batchSize:** As we are not using batch prediction, this value will be set to the default value of one hundred (100).

Table 14: Multinomial Logistic Regression (Logistic) Parameters & Selected Values

#	Parameter	Use	Value
1	batchSize	The preferred number of instances to process if batch prediction is being performed	100
2	debug	If set to true, classifier may output additional info to the console	false
3	doNotCheckCapabilities	If set, classifier capabilities are not checked before classifier is built (use with caution to reduce run time)	false
4	maxIts	Maximum number of iterations to be performed	-1
5	numDecimalPlaces	The number of decimal places to be used for the output of number in the model	2
6	ridge	Set the ridge value in the log-likelihood	1.0E-8
7	ruseConjugateGradientDescent	Use conjugate gradient descent rather than BFGS updates faster for problems with many parameters	false

2. **debug:** This does not affect classifier performance and thus we use the default value of false.

3. **doNotCheckCapabilities:** This does not affect classifier performance and thus we use the default value of false.

4. **maxIts:** Maximum iteration refers to how much cross-validation iteration are performed until the maximum likelihood estimator ridge is reached. As we need to fully optimised the values we have, we set that value being to minus one (-1), where in the tool that means that it will continue till feature class weights are fully optimised.

5. **numDecimalPlaces:** This does not affect classifier performance and thus we use the value of two (2).

6. **ridge:** The higher the value the closer to zero are the penalised maximum likelihood estimates. For that reason we use a small value of 1.0E-8 (default one) as we don't need to penalise the maximum likelihood estimates.

7. **useConjugateGradientDescent:** Our dataset does not have many parameters and thus the use of conjugate gradient descent will not improve the performance of the classifier. For that reason we are using the default BFGS updates and the value is then set to false.

B. Simulation Setup

Our simulated setup was constructed in OPNET simulator. Zigbee protocol is used for the communication between peripheral nodes and the central coordinator. The simulated network architecture adapted is the one described in Section 2.

B.1. Simulated Classes

There are simulated scenarios for each class we study, as those are described in subsection 4.1. Peripheral devices are configured to report measurements to the central coordinator every sixty seconds (one minute). This central coordinator is denoted by Energy Services Interface (ESI), which integrates a Smart Meter with intelligent control units.

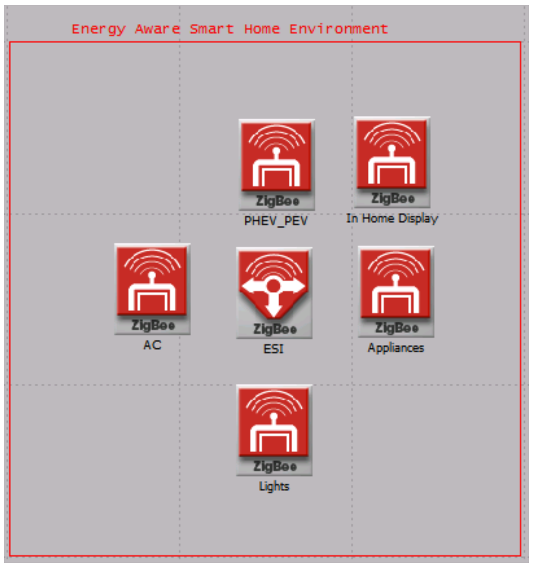


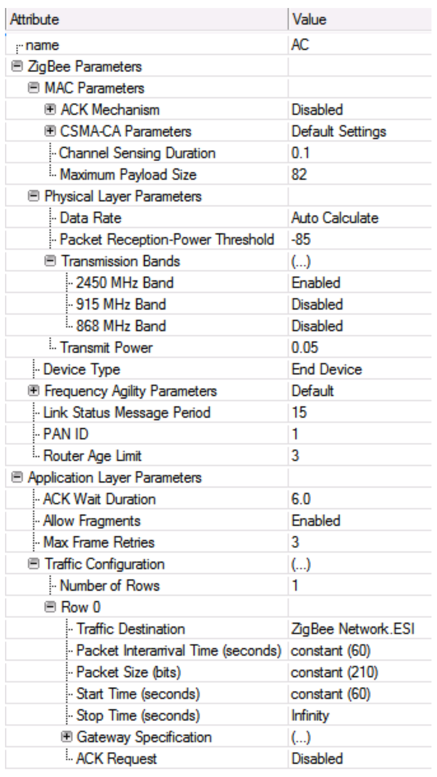
Figure 8: Simulation Architecture

B.2. Normal Class

In normal behaviour nodes capture energy consumption of their monitoring element and transmit packets to the central node with no attacks or system faults, affecting a node or the communication between a node and the central node. The normal class simulated scenario architecture is presented in Figure 8. Sensor node configuration is presented in Figure 9. During their execution the scenarios are properly configured in a way that the number of active nodes to be between four to six. Overall, the peripheral IoT enabled devices that are used for measurements transmission and monitoring are ESI, PHEV_PEV, AC, Appliances, Lights and Home Display. The same set of devices is also used in the case of faulty and attack classes scenarios.

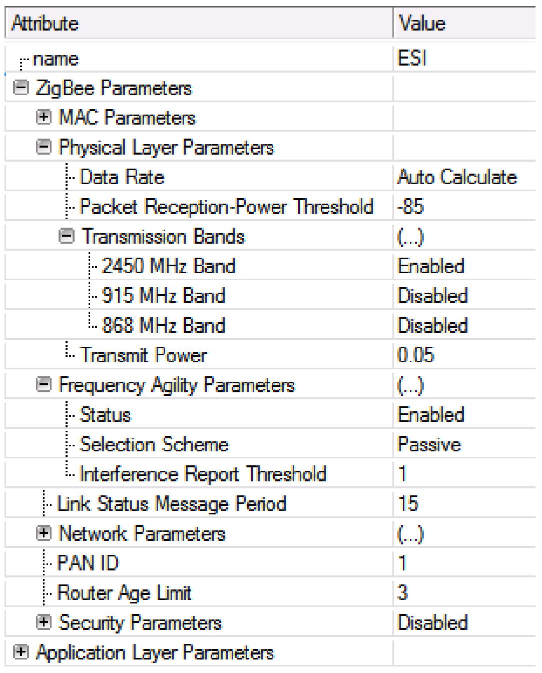
B.3. Faulty Classes

In faulty classes scenarios the network architecture is the same as the one presented in Figure 8. The same set of devices is used, with the only change being the introduction of the faulty AC node denoted by AC_F. As per the faulty class that is used to simulate a scenario the parameters of AC_F are changed in order to simulate the actual faulty class. The parameters configuration for AC_F, in each faulty class are given in each subsection.



Attribute	Value
name	AC
⊟ ZigBee Parameters	
⊟ MAC Parameters	
⊞ ACK Mechanism	Disabled
⊞ CSMA-CA Parameters	Default Settings
Channel Sensing Duration	0.1
Maximum Payload Size	82
⊟ Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
⊟ Transmission Bands	(...)
2450 MHz Band	Enabled
915 MHz Band	Disabled
868 MHz Band	Disabled
Transmit Power	0.05
Device Type	End Device
⊞ Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
⊟ Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
⊟ Traffic Configuration	(...)
Number of Rows	1
⊟ Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (60)
Stop Time (seconds)	Infinity
⊞ Gateway Specification	(...)
ACK Request	Disabled

Figure 9: Sensor Node Parameters (Normal Class Scenarios)

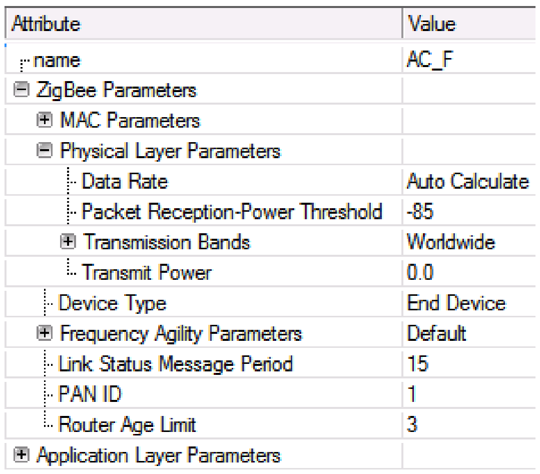


Attribute	Value
name	ESI
⊟ ZigBee Parameters	
⊞ MAC Parameters	
⊟ Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
⊟ Transmission Bands	(...)
2450 MHz Band	Enabled
915 MHz Band	Disabled
868 MHz Band	Disabled
Transmit Power	0.05
⊟ Frequency Agility Parameters	(...)
Status	Enabled
Selection Scheme	Passive
Interference Report Threshold	1
Link Status Message Period	15
⊞ Network Parameters	(...)
PAN ID	1
Router Age Limit	3
⊞ Security Parameters	Disabled
⊞ Application Layer Parameters	

Figure 10: ESI Node Parameters (Low Energy Scenario)

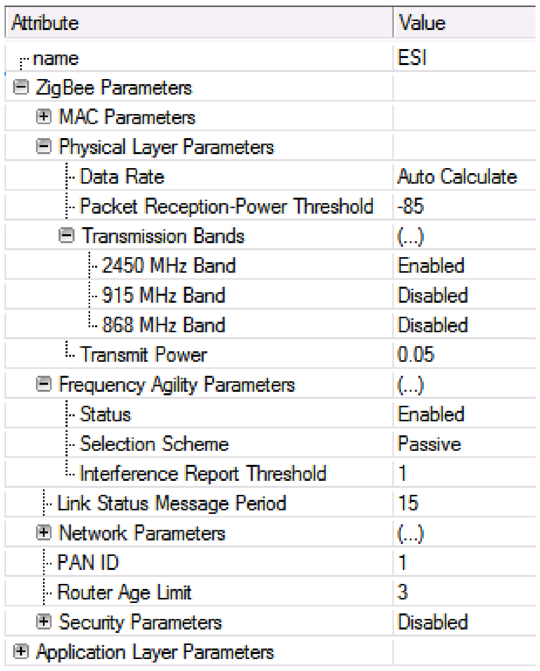
B.3.1. Low Energy Failure (F1)

In this faulty class, the faulty sensor's energy (power) falls below an acceptance threshold. This resulting into packets transmittion that might lead to drop by the central coordinator. Due to the small distances, defined in the simulated scenarios the transmitting power value need to be set to zero (0), as this will let the sensor to use it's inherent energy and not any additional one for packet generation and transmittion. Figures 10 and 11 denotes this execution parameters, for the central coordinator and the faulty AC_F node. For the rest of the sensing



Attribute	Value
name	AC_F
⊟ ZigBee Parameters	
⊞ MAC Parameters	
⊟ Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
⊞ Transmission Bands	Worldwide
Transmit Power	0.0
Device Type	End Device
⊞ Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
⊞ Application Layer Parameters	

Figure 11: AC Faulty Node Parameters (Low Energy Scenario)



Attribute	Value
name	ESI
⊟ ZigBee Parameters	
⊞ MAC Parameters	
⊟ Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
⊟ Transmission Bands	(...)
2450 MHz Band	Enabled
915 MHz Band	Disabled
868 MHz Band	Disabled
Transmit Power	0.05
⊟ Frequency Agility Parameters	(...)
Status	Enabled
Selection Scheme	Passive
Interference Report Threshold	1
Link Status Message Period	15
⊞ Network Parameters	(...)
PAN ID	1
Router Age Limit	3
⊞ Security Parameters	Disabled
⊞ Application Layer Parameters	

Figure 12: ESI Node Parameters (Routing Failure Scenario)

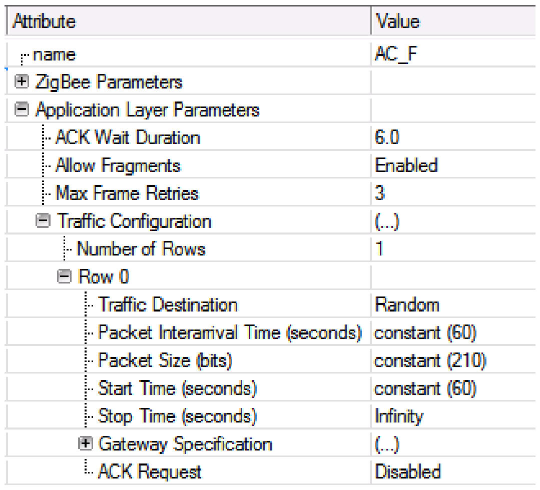
devices the configuration does not change.

B.3.2. Routing Failure (F2)

We simulate the routing failure class, by interfering the routing tables of the faulty node. This tables are frequently updated during a network operation and in our simulation we simulate the fault by altering the routing table parameters of AC_F faulty node during the simulation execution. This also affects in coordination the transmittion of packets to the receiving end. Both execution parameters for faulty node and the central coordinator are presented in Figures 12 and 13, respectively. Execution parameters for the rest of the nodes remained unchanged.

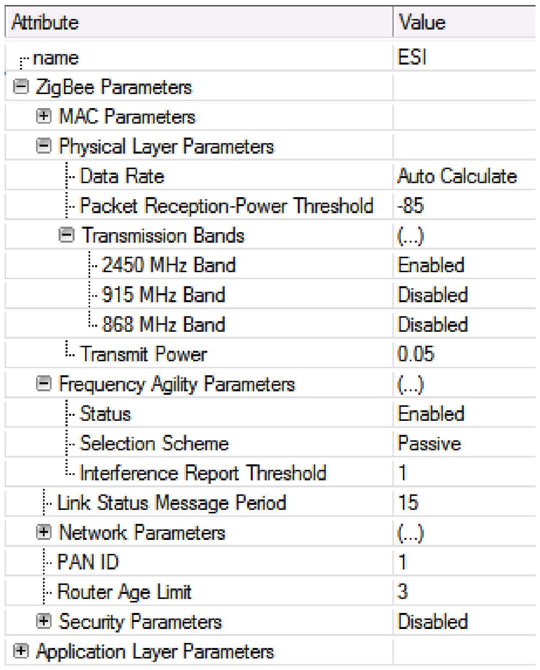
B.3.3. Packet Drooped Failure (F3)

Packet Drooped Failure describes the faulty class, where generated packets at sender experience additional noise that lead to not readable packets by receiver. The additional noise affects the SNR quality at the sender and lead to bit mistakes



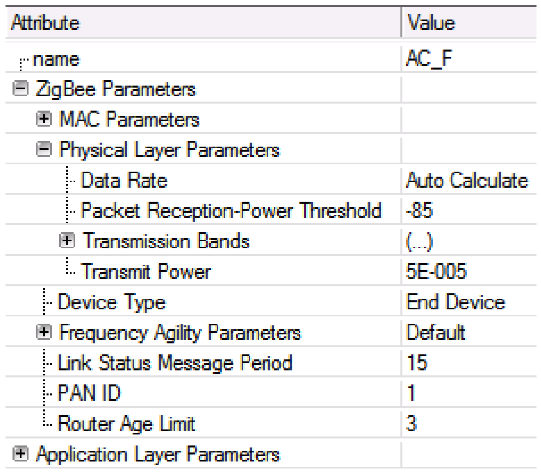
Attribute	Value
name	AC_F
ZigBee Parameters	
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	(...)
Number of Rows	1
Row 0	
Traffic Destination	Random
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (60)
Stop Time (seconds)	Infinity
Gateway Specification	(...)
ACK Request	Disabled

Figure 13: AC Faulty Node Parameters (Routing Failure Scenario)



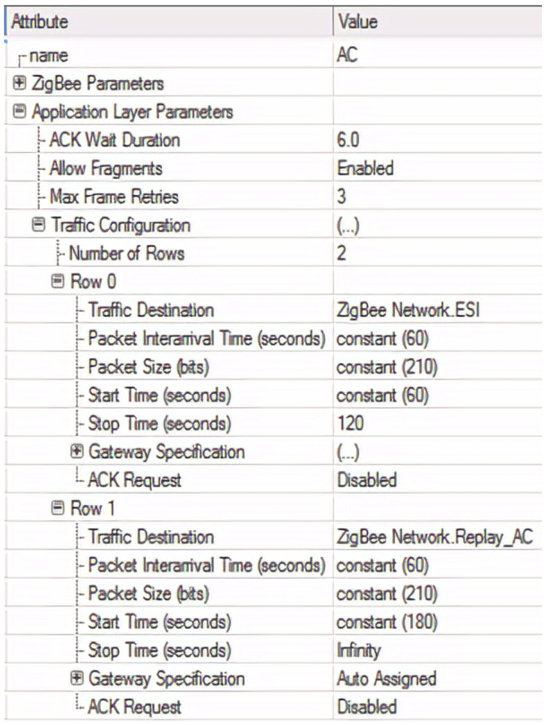
Attribute	Value
name	ESI
ZigBee Parameters	
MAC Parameters	
Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
Transmission Bands	(...)
2450 MHz Band	Enabled
915 MHz Band	Disabled
868 MHz Band	Disabled
Transmit Power	0.05
Frequency Agility Parameters	(...)
Status	Enabled
Selection Scheme	Passive
Interference Report Threshold	1
Link Status Message Period	15
Network Parameters	(...)
PAN ID	1
Router Age Limit	3
Security Parameters	Disabled
Application Layer Parameters	

Figure 14: ESI Node Parameters (Packet Drooped Failure Scenario)



Attribute	Value
name	AC_F
ZigBee Parameters	
MAC Parameters	
Physical Layer Parameters	
Data Rate	Auto Calculate
Packet Reception-Power Threshold	-85
Transmission Bands	(...)
Transmit Power	5E-005
Device Type	End Device
Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
Application Layer Parameters	

Figure 15: AC Faulty Node Parameters (Packet Drooped Failure Scenario)



Attribute	Value
name	AC
ZigBee Parameters	
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	(...)
Number of Rows	2
Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (60)
Stop Time (seconds)	120
Gateway Specification	(...)
ACK Request	Disabled
Row 1	
Traffic Destination	ZigBee Network.Replay_AC
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (180)
Stop Time (seconds)	Infinity
Gateway Specification	Auto Assigned
ACK Request	Disabled

Figure 16: AC Node Parameters (Replay Attack Scenario)

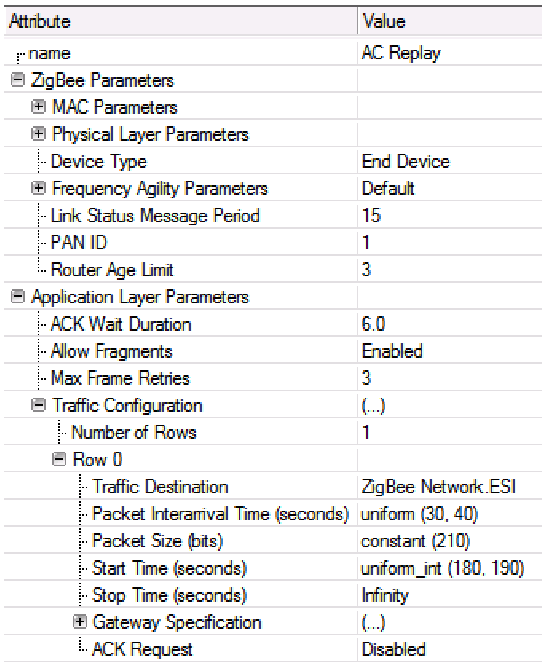
in the packets. For the simulation the additional noise is modelled in both sender and receiver. For the sender the fault is modelled by altering the Transmit Power, to low value and in the receiver via the Packet-Reception-Power Threshold that defines receiver sensitivity in dBm. The packets that does not satisfy this lower threshold are not extracted by receiver and are dropped. Parameters configuration's for both the faulty node and the central coordinator are presented in Figures 14 and 15, respectively.

B.4. Attack Classes

The simulated architecture displayed in Figure 8 in changed in the attack classes, simulation scenarios as we had the introduction of an additional device, this of the attacker. The attacker in this simulations will be a successful MITM device, that stands in the middle of the communication between peripheral node (AC) and central coordinator (ESI). Based on the simulated attack class this attacker is differently configured and has a different name. In all the attack scenarios that are simulated, the successful MITM interruption of the communication between the sender (AC) and the receiver (ESI), lead to change on the simulated traffic configurations of AC. The actual configuration parameters for both AC and attacker node are described in the description of each attack class in this subsection.

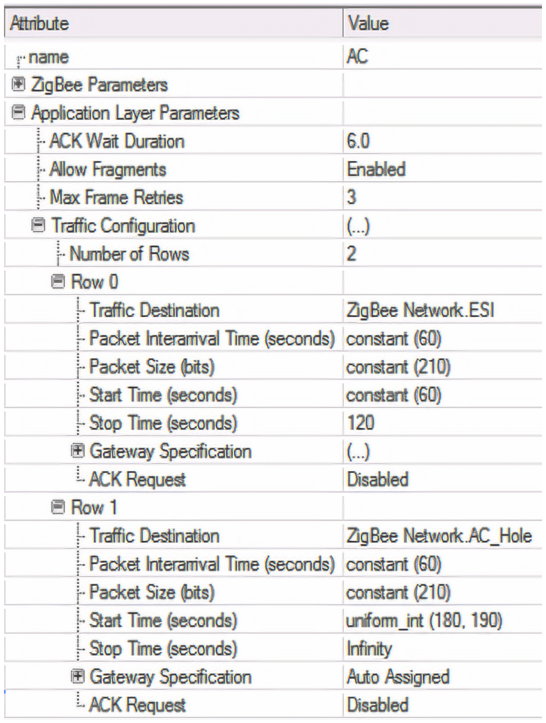
B.4.1. Replay Attack (A1)

Replay attack class is simulated by setting the attacker to resents legitimate packets multiple times in order to disrupt communication by introducing further delay and out of order packets transmissions. Simulated scenario behaviour has as follow: AC node attacker's victim behaves normally for the first two minutes of the simulation. At the third (3) minute of the simulation the attack takes place and since that until the



Attribute	Value
name	AC Replay
ZigBee Parameters	
MAC Parameters	
Physical Layer Parameters	
Device Type	End Device
Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	(...)
Number of Rows	1
Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	uniform (30, 40)
Packet Size (bits)	constant (210)
Start Time (seconds)	uniform_int (180, 190)
Stop Time (seconds)	Infinity
Gateway Specification	(...)
ACK Request	Disabled

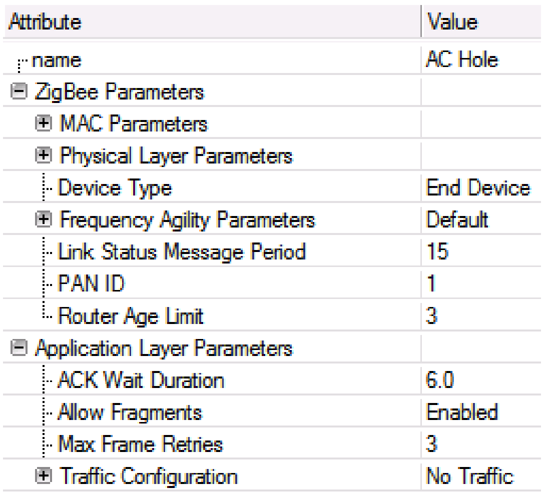
Figure 17: AC Replay Node Parameters (Replay Attack Scenario)



Attribute	Value
name	AC
ZigBee Parameters	
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	(...)
Number of Rows	2
Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (60)
Stop Time (seconds)	120
Gateway Specification	(...)
ACK Request	Disabled
Row 1	
Traffic Destination	ZigBee Network.AC_Hole
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	uniform_int (180, 190)
Stop Time (seconds)	Infinity
Gateway Specification	Auto Assigned
ACK Request	Disabled

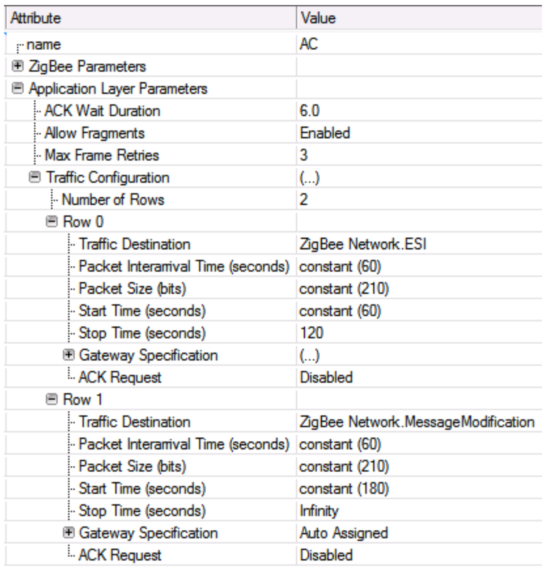
Figure 18: AC Node Parameters (Sink Hole Attack Scenario)

end of the simulation the packets that originally sent to the central coordinator are now forwarded to the attacker. The attacker, then records those packets and retransmit them to the central node in a random manner. The inter arrival time (x) between messages transmittion to the ESI from attacker is uniformly selected with a value between thirty (30) to forty (40) seconds. The parameters that are changed in order to simulate this attack are related to the traffic configuration of the nodes, basically in terms of destination. The execution parameters used in the simulator for the target node AC and attacker node are presented in Figures 16 and 17, respectively.



Attribute	Value
name	AC Hole
ZigBee Parameters	
MAC Parameters	
Physical Layer Parameters	
Device Type	End Device
Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	No Traffic

Figure 19: AC Hole Node Parameters (Sink Hole Attack Scenario)



Attribute	Value
name	AC
ZigBee Parameters	
Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
Traffic Configuration	(...)
Number of Rows	2
Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (60)
Stop Time (seconds)	120
Gateway Specification	(...)
ACK Request	Disabled
Row 1	
Traffic Destination	ZigBee Network.MessageModification
Packet Interarrival Time (seconds)	constant (60)
Packet Size (bits)	constant (210)
Start Time (seconds)	constant (180)
Stop Time (seconds)	Infinity
Gateway Specification	Auto Assigned
ACK Request	Disabled

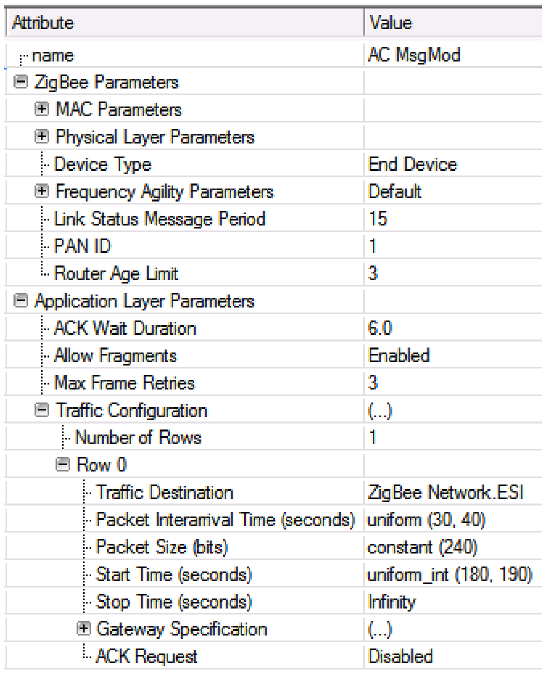
Figure 20: AC Node Parameters (Message Modification Attack Scenario)

B.4.2. Sink Hole Attack (A2)

We simulate Sink Hole attack class, by allowing attacker to record all packets transmitted between the actual targets and dropped them, without reporting them back to their destination. In simulation execution the attack is performed in the third (3) minute of the simulation and then all packets captured by attacker are never transmitted. This simulates the aggressive sink hole attack. The parameters that are changes in order to simulate this attack, are related to the traffic configuration of the nodes, basically in terms of destination. The execution parameters used in the simulator for the target node of AC and attacker node are presented in Figures 18 and 19, respectively.

B.4.3. Message Modification Attack (A3)

Message Modification Attack class is simulated by setting the victim node packets to be recorded and retransmitted by attacker with a modification altering their original message. Packet's payload changes with the introduction of fraudulent data. We simulate this attack by introducing it to the network



Attribute	Value
name	AC MsgMod
⊟ ZigBee Parameters	
⊞ MAC Parameters	
⊞ Physical Layer Parameters	
Device Type	End Device
⊞ Frequency Agility Parameters	Default
Link Status Message Period	15
PAN ID	1
Router Age Limit	3
⊟ Application Layer Parameters	
ACK Wait Duration	6.0
Allow Fragments	Enabled
Max Frame Retries	3
⊟ Traffic Configuration	(...)
Number of Rows	1
⊟ Row 0	
Traffic Destination	ZigBee Network.ESI
Packet Interarrival Time (seconds)	uniform (30, 40)
Packet Size (bits)	constant (240)
Start Time (seconds)	uniform_int (180, 190)
Stop Time (seconds)	Infinity
⊞ Gateway Specification	(...)
ACK Request	Disabled

Figure 21: AC MsgMod Node Parameters (Message Modification Attack Scenario)

at the third (3) minute of the simulation and the extra payload is added to the extracted packets, from the attacker. The transmittion of those packets by the adversary node (MsgMod_AC) to the central ESI node is performed after every x seconds, with the actual packet inter-arrival time (x) being selected uniformly with an integer value in the range of [30,40]. The parameters changed in order to simulate this attack, are related to the traffic configuration of the nodes in terms of packet size (payload) and destination. The execution parameters used in the simulator for AC and attacker node are presented in Figures 20 and 21, respectively.