electronics

MDPI

Article

An Anomaly-Based Intrusion Detection System for Internet of Medical Things Networks

Georgios Zachos [1,2,*], Ismael Essop [2], Georgios Mantas [1,2], Kyriakos Porfyrakis [2], José C. Ribeiro [1] and Jonathan Rodriguez [1,3]

[1] Instituto de Telecomunicações, 3810-193 Aveiro, Portugal; gimantas@av.it.pt (G.M.); jcarlosvgr@av.it.pt (J.C.R.); jonathan@av.it.pt (J.R.)
[2] Faculty of Engineering and Science, University of Greenwich, Chatham Maritime ME4 4TB, UK; i.a.essop@greenwich.ac.uk (I.E.); k.porfyrakis@greenwich.ac.uk (K.P.)
[3] Faculty of Computing, Engineering and Science, University of South Wales, Pontypridd CF37 1DL, UK
* Correspondence: g.zachos@av.it.pt

Abstract: Over the past few years, the healthcare sector is being transformed due to the rise of the Internet of Things (IoT) and the introduction of the Internet of Medical Things (IoMT) technology, whose purpose is the improvement of the patient's quality of life. Nevertheless, the heterogenous and resource-constrained characteristics of IoMT networks make them vulnerable to a wide range of threats. Thus, novel security mechanisms, such as accurate and efficient anomaly-based intrusion detection systems (AIDSs), considering the inherent limitations of the IoMT networks, need to be developed before IoMT networks reach their full potential in the market. Towards this direction, in this paper, we propose an efficient and effective anomaly-based intrusion detection system (AIDS) for IoMT networks. The proposed AIDS aims to leverage host-based and network-based techniques to reliably collect log files from the IoMT devices and the gateway, as well as traffic from the IoMT edge network, while taking into consideration the computational cost. The proposed AIDS is to rely on machine learning (ML) techniques, considering the computation overhead, in order to detect abnormalities in the collected data and thus identify malicious incidents in the IoMT network. A set of six popular ML algorithms was tested and evaluated for anomaly detection in the proposed AIDS, and the evaluation results showed which of them are the most suitable.

Keywords: Internet of Medical Things (IoMT); intrusion detection system (IDS); machine learning algorithms; anomaly-based intrusion detection; IoT datasets

Citation: Zachos, G.; Essop, I.; Mantas, G.; Porfyrakis, K.; Ribeiro, J.C.; Rodriguez, J. An Anomaly-Based Intrusion Detection System for Internet of Medical Things Networks. *Electronics* **2021**, *10*, 2562. https://doi.org/10.3390/electronics10212562

Academic Editor: Constantinos Kolias

Received: 18 September 2021
Accepted: 18 October 2021
Published: 20 October 2021

1. Introduction

The rise of the Internet of Things (IoT) is transforming the healthcare sector, introducing the Internet of Medical Things (IoMT) technology, whose aim is to improve the patient's quality of life by enabling personalized e-health services without limitations on time and location [1–3]. However, the wide range of different communication technologies (e.g., WLANs, Bluetooth, Zigbee) and types of IoMT devices (e.g., medical sensors, actuators) incorporated in IoMT edge networks are vulnerable to various types of security threats, raising many security and privacy challenges for such networks, as well as for the healthcare systems relying on these networks [4–6]. For instance, an adversary could intrude into the IoMT network in order to intercept transmitted medical data and/or gain unauthorized access to sensitive information [2]. In addition, attackers may compromise IoT-based healthcare systems through their IoMT networks in order to manipulate the sensing data (e.g., by injecting fake data) and cause malfunctions (e.g., by flooding the resource-constrained IoMT network with a large amount of requests) to the compromised IoT-based healthcare systems that, in turn, will jeopardize the integrity or the availability of the healthcare services provided by these systems. Consequently, security solutions pro-

tecting IoMT networks from adversaries are critical for the acceptance and wide adoption of such networks in the coming years.

Nevertheless, the high resource requirements of complex and heavyweight conventional security mechanisms cannot be afforded by (a) the resource-constrained IoMT edge devices with limited processing power, storage capacity, and battery life, and/or (b) the constrained environment in which the IoMT edge devices are deployed and interconnected using lightweight communication protocols [7]. Therefore, it is clear that there is an urgent need for novel security mechanisms to address the pressing security challenges of IoMT networks in an effective and efficient manner, taking into consideration their inherent limitations due to their resource-constrained characteristics, before IoMT networks gain the trust of all involved stakeholders and reach their full potential in the healthcare market [5,8–12]. Taking a step toward this direction, anomaly-based intrusion detection is currently foreseen by the industry and research community as a promising security solution that can play a significant role in protecting IoT networks, as long as novel lightweight anomaly-based intrusion detection systems (AIDSs) are developed [13]. However, so far, and to the best of our knowledge, there is only one related work on AIDSs for IoMT presented in [14], demonstrating the lack of proper works of AIDSs for IoMT networks. Therefore, our aim is to fill this significant research gap by developing a novel hybrid AIDS tailored to the resource-constrained characteristics of IoMT edge networks [7]. Thus, in this paper, we present the system architecture for a novel hybrid AIDS for IoMT networks, leveraging host-based and network-based techniques to reliably monitor and collect log files from the IoMT devices and the gateway, as well as traffic from the IoMT edge network, while simultaneously considering the computational cost. The detection process of the proposed AIDS is to be implemented by the detection engine running on the gateway of the IoMT edge network and relying on machine learning (ML) techniques, considering the computation overhead, in order to detect abnormalities in the collected data and thus identify malicious incidents in the IoMT network. Besides that, in order to evaluate potential detection ML algorithms and identify the most appropriate ones, among the most popular ML algorithms (e.g., naïve Bayes, the decision tree (DT), random forest (RF) and k-nearest neighbor (KNN)), for the proposed AIDS, we used (i) the network part of the "TON_IoT Telemetry dataset" [15], as it is the most recent and representative data-driven IoT/IIoT-based dataset, and (ii) the dataset that was produced according to the approach of the authors in [7] and that includes information related to the behavior of the IoT devices that is not included in the "TON_IoT Telemetry dataset" [15], but is very critical for building effective ML-based detection models for AIDSs. We used these two datasets to evaluate potential detection ML algorithms because, to the best of our knowledge, there is no publicly available IoMT-specific dataset containing all of this useful information other than the above mentioned two datasets. The evaluation results demonstrate that the decision tree (DT), random forest (RF), and k-nearest neighbor (KNN) algorithms are more suitable to be used as the core of the detection component. The proposed system design and the evaluation results constitute the basis for the next step of our work, which is the development of the proposed AIDS in an IoMT testbed consisting of a Raspberry Pi 4 device, playing the role of the gateway, and a set of MTM-CM5000-MSP sensors.

Following the introduction, this paper is organized as follows. Section 2 reviews related work on existing AIDSs for IoMT and ML algorithms employed in AIDSs for IoT networks. Section 3 gives the metrics for evaluating the performance of ML algorithms for intrusion detection. Existing datasets used to perform training and evaluation of AIDSs for IoMT networks are described in Section 4. Section 5 presents the scenario architecture (i.e., perception domain) where the proposed AIDS will be deployed. In Section 6, the proposed AIDS is introduced, and its different components are presented. In Section 7, the results of the performance evaluation of different ML algorithms (i.e., decision tree, naïve Bayes, linear regression, random forest, k-nearest neighbor, and support vector machines) are presented and discussed. In Section 8, challenges and future work are discussed. Finally, Section 9 concludes the paper.

2. Related Work

This section initially discusses existing IDSs for IoMT. Then, a set of the most popular ML algorithms for IoT AIDS, tested and evaluated for anomaly detection in our case, are presented, followed by the metrics based on which their performance is evaluated. Finally, we discuss available IoT datasets and the datasets that we considered in our experiments.

2.1. Anomaly-Based Intrusion Detection Systems (AIDSs) for IoMT

To the best of our knowledge, the AIDS presented in [14] is the only existing AIDS specifically designed for IoMT networks. The authors designed and developed a ML-based intrusion detection solution utilizing mobile agent technology in order to protect the network, which comprises connected medical IoT devices. The proposed attack detection mechanism in connected medical devices is hierarchical and distributed using autonomous mobile agents. Every node in the network acts as a computing node, while mobile agents migrate, learn, and collaboratively perform attack detection. Three types of agents are employed. The first type, named as a sensor agent (SA), is spawned by the cluster-head (CH) of a WBAN cluster and traverses a set of sensor nodes specified by the CH (i.e., itinerary parameters) in order to locally detect a specific category of attack on sensor nodes, based on the aggregated logs accumulated over a period of time, and generate an alert based on the detection results. The second type, called the cluster-head agent (CA) functions in a more distributed manner than the SAs and is responsible for detecting anomalies among the CHs of multiple interconnected WBAN clusters. A CA may be static, residing on only one CH and performing intrusion detection at regular intervals, or it can be mobile, traversing several CHs, based on its specified itinerary parameters, and performing intrusion detection on each CH. The third type, named as a detective agent (DA), is spawned by a CH only in the case where the SA cannot classify the network behavior as normal or malicious. Then, the DA traverses the entire cluster, collecting network activity data, which are sent back to its CH. The CH utilizes the collected data by the DA and employs a conflict resolution detection algorithm for the specific cluster. The result of the conflict resolution detection algorithm triggers an appropriate alarm. In order to detect network level intrusions, as well as anomalies in the sensor data, ML algorithms and regression algorithms are employed, respectively. The authors simulated a hospital network topology and experimented with several subsets of wireless body area networks and connected medical devices. The performance of various ML algorithms (i.e., support vector machines, decision trees, naïve Bayes, k-nearest neighbor, and random forests) was evaluated in order to distinguish the best algorithm for network level intrusion detection. Similarly, polynomials of various orders (i.e., regression algorithms) were evaluated in order to find the optimal order value for the case of device intrusion detection. Their simulation results demonstrate that the proposed IDS is able to achieve a high detection accuracy with minimal energy consumption overhead. However, the datasets produced and used by the authors in [14] do not include a wide variety of attacks and cannot be easily considered as the most representative dataset reflecting the attacks targeting IoMT networks.

2.2. Machine Learning Algorithms for IoT Intrusion Detection

In this section, we review the following most popular ML algorithms for IoT AIDS: decision tree (DT), naïve Bayes (NB), linear regression (LR), random forest (RF), support vector machines (SVM), and k-nearest neighbor (KNN). As mentioned in [13], each of these ML algorithms has been frequently used in the design of various AIDS for IoT. The authors in [15] also state that support vector machines (SVM), k-nearest neighbor (KNN), naïve Bayes (NB), decision-tree-based methods (i.e., random forest (RF)), and logistics regression (LR) are suitable ML algorithms for the design of an AIDS. Additionally, at the end of the section, Table 1 presents a summary of the six ML algorithms, along with their advantages and drawbacks when applied for anomaly detection and associated studies mentioned in each of the subsections.

Table 1. Summary of ML algorithms along with their advantages and drawbacks and associated studies.

ML Algorithm	Advantages	Drawbacks	Studies
Decision Tree	Simple to use. Performance is not different for linearly and non-linearly separated parameters.	Vulnerable to overfitting. Unstable (i.e., small data variation may result in the construction of extremely different DTs).	[16,17]
Random Forest	Resistant to overfitting. Feature selection is performed inherently.Fewer inputs required.	Fast only in the case of a small number of trees. May require large datasets.	[13,15,17–19]
Naïve Bayes	Can be used in both binary and multi- class classification. Simple to use. Few samples required to train.	The assumption about features independence can lead to low classification accuracy. "Zero frequency" problem. In the case where a class does not appear during training, it will be assigned a probability of zero.	[13,20]
Logistic Regression	Simple to use. Easy to implement.	Difficult to perform classification in case of non-linearly separable classes.	[15,21,22]
Support Vector Machine	Better performance in datasets with few classes and many instances per class. Scalable. Reduced storage requirements.	Finding the most appropriate kernel function is a challenge.	[13]
K-Nearest Neighbor	Simple to use. Easy to implement.	Difficult to find the optimal k. The computational speed decreases as the number of the k variable, the number of data points, or the number of classes increases.	[13,15]

2.2.1. Decision Tree (DT)

A decision tree (DT) is a ML algorithm that functions by extracting features of the instances of a training dataset and then constructing an ordered tree based on the values of the extracted features. In a DT, a node corresponds to a feature and the branches of that node correspond to the values of that feature. The construction of the DT starts from the origin node of the tree. The feature, which will be the origin node of the tree, is selected among those features that optimally split the tree in two. In order to identify the feature that optimally divides the tree, various metrics are employed, including the Gini index and information gain. DTs carry out the induction and inference processes [16].

The induction process involves the construction of a DT by combining unoccupied nodes and branches. Initially, based on the information gain or other measures, the most suitable feature is selected as the origin node of the DT. Then, the induction process continues and, in each subsequent step, features are selected as tree nodes. The selection of features is performed in such a way that the overlapping among the different classes of the training dataset can be kept to a minimum. In the end, the leaves of each sub-DT are identified and classified according to their corresponding classes.

The inference process occurs in a constructed DT. During this process, unknown instances are classified through an iterative comparison with the created DT. The classification process regarding the new sample is finished when a matching leaf node is found [16].

In our experiments, the Gini index was used as a measure to select both the origin node of the DT and the rest of the tree nodes. In addition, the minimum number of samples per leaf node was set to 10 in order to end up with a pruned tree and to avoid overfitting, as it is suggested in [17].

2.2.2. Random Forest (RF)

A random forest (RF) is a supervised ML algorithm consisting of multiple DTs that are used to perform more accurate and error resistant classifications [18]. During the training of the model, DTs are constructed randomly and are then are trained to classify instances according to majority voting [18]. RFs are trained in a different way compared to DTs. Whereas in a DT, a ruleset is created during training based on the training dataset, in a RF, the various DTs are generated with every DT using randomly picked instances from the training dataset as an input.

Due to the inherent randomness of the training process, the output of an RF model becomes more robust and accurate, and the RF model is more resistant to overfitting. Moreover, it requires significantly less inputs and does not require the process of feature selection. The authors in [19] showed that a RF classifier can perform a better detection of DDoS attacks in IoT networks than KNN, an artificial neural network (ANN), and SVM classifiers [13].

As in the case of DTs, in our experiments, the Gini index was used to construct the DT components, and the minimum number of samples per leaf node was set to 10 in order to avoid over fitting, as suggested in [17]. The RF consists of 10 DTs based on the work in [15].

2.2.3. Naïve Bayes (NB)

This algorithm utilizes Bayes' theorem to calculate the probability of occurrence of an event (either normal or abnormal) according to previous observations of similar events [20]. The NB classifier operates on a strong feature independence assumption. In other words, the NB model considers that the values of one feature do not affect the values of another feature at all. In ML scenarios, this assumption can be made in order to classify normal and abnormal behaviors, taking into account the previous observations in a supervised learning mode. The NB classifier is a commonly employed supervised classifier known for its simplicity and ease of implementation. It computes posterior probability and, based on that, unlabeled instances can be classified as normal or abnormal. Its training does not require many samples and it can be employed in both binary and multi-label classification problems. Nevertheless, due to its feature independence assumption, the NB classifier fails to perceive interdependencies among the features of a dataset, which can negatively impact its accuracy [13]. In our experiments, the Gaussian variant of the naïve Bayes algorithm was employed in particular, where the likelihood of the features is assumed to be Gaussian.

2.2.4. Logistic Regression (LR)

A logistic regression (LR) algorithm can estimate the probability of a particular instance belonging to a specific class, and, for that reason, is frequently employed in classification problems regarding intrusion detection and spam filtering [21]. Furthermore, the study in [22] designed and implemented a security solution based on a LR algorithm and showed that it is possible to secure an IoT-based production line against DDoS attacks by using ML algorithms and commonly available tools for network traffic analysis and evaluation.

The LR algorithm utilizes a predetermined probability threshold in order to classify the instances. For example, in the case of binary classification, a threshold of 50% would mean that an instance is normal if its estimated probability is less than 50%. If the estimated probability is greater than 50%, then the algorithm (i.e., LR classifier) will decide that this is an attack instance. LR estimates the probability using the following equation:

$$h_\theta(x) = \sigma(\theta^T \times x) \quad (1)$$

where h_θ is the hypothesis function, which outputs the estimated probability, x is the feature vector of the instance, θ is the model's parameter, θ^T is the transpose of θ, and σ(.) is a sigmoid (i.e., logistic) function that defines the threshold. The equation of σ(.) is the following:

$$\sigma(z) = 1/(1 + e^{(-z)}) \quad (2)$$

$$z = (\theta^T \times x) \quad (3)$$

The sigmoid function outputs a number between 0 and 1. A value that is closer to 0 signifies a normal observation and a value closer to 1 indicates an attack observation. The model's parameter θ is calculated during the training phase [15].

2.2.5. Support Vector Machine (SVM)

The SVM algorithm functions by creating a hyperplane in the feature set of two or more classes. This hyperplane splits the instances into groups and is determined by calculating a maximum distance of the nearest data point of each compared class. The best use case for SVMs is when the classification problem relates to classes with large feature sets and fewer data instances [13].

Due to its simplicity, a SVM classifier is highly scalable. Moreover, it can perform tasks such as anomaly-based intrusion detection in real-time, including real-time learning. In addition, a SVM classifier does not require much storage or memory to implement. As a result of their scalability and low requirements, SVMs are suitable for use in IDSs that are implemented in a resource-constrained IoT system. However, when the data are not linearly separable, it is crucial to carefully consider and select which kernel function the SVM algorithm will use to split the data. When finding the best kernel function to achieve a specific classification, its speed has always been a challenge [13]. In our experiments, a SVM classifier with a Gaussian radial basis function (RBF) kernel was utilized.

2.2.6. K-Nearest Neighbor (KNN)

The K-nearest neighbor (KNN) algorithm is simple to use and utilizes a distance function, which is typically the Euclidean distance function, in order to decide the class of an object based on its distance from its closest neighbors. The parameter K refers to the number of nearest neighbors that are used during the classification process. The value of K may change the classification result. Therefore, it is necessary for the accuracy of the KNN algorithm to find the optimal value of K. The value is found through testing, and this can be extremely time-consuming in some cases [13]. In our experiments, the value of K was set to 5, and the Euclidean distance was selected as the distance metric based on [15].

3. Evaluation Metrics

Various metrics are used to evaluate the performance of ML algorithms based on testing datasets. In order to calculate the evaluation metrics, the first step is the calculation of the values of the confusion matrix. The confusion matrix is generated when a trained ML model is used to classify the instances of a testing dataset. The confusion matrix compares values regarding the actual labels of the instances of the testing dataset and the corresponding labels predicted by the ML model. Table 2 shows the 2-by-2 confusion matrix regarding a classification problem with two classes (normal and attack).

Table 2. Confusion matrix for binary classification problems.

		Predicted Label	
		Positive (Attack)	**Negative (Normal)**
Actual Label	Positive (Attack)	True Positive (TP)	False Negative (FN)
	Negative (Normal)	False Positive (FP)	True Negative (TN)

The true positive (TP) and true negative (TN) relate to the correctly classified attack instances and normal instances, respectively. The false positive (FP) and false negative (FN) refer to the incorrectly classified normal instances and attacks instances, respectively. Based on these values, it is possible to compute several evaluation metrics, as shown in [13,23–25]. In our case, the metrics of accuracy, precision, recall, and F1-score were used, and each metric is shortly presented below, along with its equation.

- **Accuracy**: shows the overall success of the model by comparing the amount of the correctly classified attack and normal instances to the total amount of instances.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (4)$$

- **Precision**: estimates the overall effectiveness of the model by calculating the percentage that an observation recognized as an attack is actually an attack observation.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (5)$$

- **Recall**: shows the overall success of the model by computing the percentage that an actual attack observation is correctly classified.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (6)$$

- **F1-score**: is calculated by the precision and recall metrics as their harmonious mean. It is a statistical function for estimating the accuracy of the model. As the precision and recall of a model approach the value of 100%, the F1-score and accuracy are maximized, and every instance is classified correctly.

$$\text{F1-score} = 2 \times (\text{Recall} \times \text{Precision})/(\text{Recall} + \text{Precision}) \quad (7)$$

4. Datasets for AIDS in IoT

In this subsection, the following six existing datasets for the training and evaluation of IoT AIDSs are reviewed: (i) the LWSNDR dataset [26], (ii) the dataset presented in [27] for classifying IoT devices using network traffic characteristics, (iii) the "Bot-IoT" dataset [28], (iv) the dataset presented in [29] for detecting DoS attacks on IoT devices using network traffic traces, (v) the "TON_IoT Telemetry" dataset [15], which is the most recent and representative data-driven IoT/IIoT-based dataset [30], and (vi) the dataset generated as described in [7], which includes information related to the behavior of the IoT devices and the IoT network traffic based on a simulated benign scenario and a simulated malicious scenario. In this work, we utilized a part of the "TON_IoT Telemetry" dataset [15] and a part of the dataset generated as presented in [7] for the training and evaluation of the ML algorithms.

4.1. LWSNDR Dataset

The authors in [26] created two wireless sensor networks (WSNs) in order to serve as testbeds for the simulation of a single-hop sensor-data collection scenario and a multi-hop sensor-data collection scenario, respectively. In both scenarios, Crossbow TelosB motes were used as sensor nodes, and real humidity–temperature sensor data were collected.

In the single-hop scenario, four motes are used as sensor nodes and one mote as the base station node. The four sensor nodes were split into two sets of two nodes, and the first set of nodes collected indoor data, whereas the other set of nodes collected outdoor data. Both sets of sensor nodes transmitted the gathered data to the base station node. In addition, anomalies were introduced to one sensor node in each set (i.e., indoor and outdoor) by utilizing a hot water kettle that alters both the temperature and the humidity simultaneously.

In the multi-hop scenario, four motes are used as sensor nodes, two motes as router nodes, and one mote as the base station node. The router nodes exist in the testbed because

the sensor nodes are placed at a distance from where they cannot directly transmit their data to the base station node. The sensor nodes and the router nodes are split in two sets. In each set, two sensor nodes are connected to one router node, whereas the router node connects to the base station node. The two sensor nodes collect humidity–temperature data and send these data to the router node, which then transmits the data to the base station node. The sensor nodes of the first set are responsible for gathering indoor sensor readings, whereas the sensor nodes of the other set collect outdoor sensor readings. Similar to the single-hop scenario, in the multi-hop scenario, anomalies were also introduced to one sensor node in each set (i.e., indoor and outdoor) using a hot water kettle, which leads to an increase in both the temperature and the humidity simultaneously.

In both the single-hop and multi-hop scenarios, real labeled data were generated and were organized in a labelled dataset in order to be used for the purpose of evaluating anomaly detection algorithms. However, the produced dataset (i.e., "LWSNDR" dataset) contains only pure sensor telemetry data, and no information related to either the sensor behavior (e.g., energy consumption) or the network traffic flowing through the WSN is included. In addition, the given dataset does not include any specific attack scenarios, as also mentioned in [15]. Finally, the "LWSNDR" dataset was created in 2010 and cannot be easily considered as recent and representative regarding the current IoT devices or the attacks targeting them.

4.2. A Dataset for Classifying IoT Devices Using Network Traffic Characteristics

The authors in [27] designed and developed a robust framework that performs the classification of IoT devices separately, in addition to one class of non-IoT devices, with high accuracy, utilizing statistical attributes derived from network traffic characteristics. One of the authors' contributions was the creation of a smart environment infrastructure that served as a testbed in order to gather and synthesize traffic traces from several IoT devices. The smart environment contains a wide range of IoT devices (i.e., 28 unique IoT devices), non-IoT devices (e.g., smart phones, laptops) and a WiFi access point (i.e., TP-Link access point). The WiFi access point enables the IoT devices and non-IoT devices to communicate with the Internet servers via a gateway [27]. The authors considered the following types of IoT devices: cameras, controllers/hubs, energy management devices (e.g., lights, plugs, motion sensors), appliances, and health-monitors.

Using the created smart environment, traffic traces were collected and synthesized for a period of six months. The traffic traces were collected using the "tcpdump" tool and were stored as "pcap" files on an external USB hard drive of 1 terabyte (TB) storage attached to the gateway. The captured IoT traffic traces comprise (a) traffic produced by the IoT devices without any human interaction (e.g., DNS, NTP), and (b) traffic produced because of the users' interaction with the IoT devices (e.g., motion sensors, lightbulb color change upon user request). Next, the traffic traces were analyzed to gain insight on how to utilize them in order to perform classification of the IoT devices. The analysis of the authors showed that network traffic characteristics, such as activity cycles, port numbers, signaling patterns, and cipher suites, can be exploited in order to properly classify each IoT device.

A subset of these traffic traces was made publicly available as a dataset in order to be used by the scientific community. However, these traffic traces cannot be used to train and evaluate anomaly-based intrusion detection mechanisms. They were not generated based on a specific type of attack scenario, and, as a result, they are not representative regarding the behavior of IoT devices or the traffic of IoT networks when under attack.

4.3. Bot-IoT Dataset

The authors in [28] generated a dataset, named as the "Bot-IoT" dataset, by incorporating simulated legitimate IoT network traffic, as well as IoT network traffic related to several different types of attacks. In order to generate the "Bot-IoT" dataset, a realistic testbed was developed, with the aim of being representative of an IoT network, and it comprises three components: (i) the network platforms, (ii) the simulated IoT services, and (iii) the

extracting features and forensics analytics. Initially, as far as the network platforms of the testbed are concerned, both normal and attacking virtual machines (VMs) with additional network devices (i.e., firewall, tap) were included. Furthermore, the Node-RED tool [31] was employed in order to simulate certain IoT services (e.g., weather station, smart fridge). Finally, regarding the extracting features and forensics analytics, after the authors gathered the normal and attack traffic of the testbed in "pcap" files, they employed the Argus tool in order to extract the flow data and used a MySQL database in order to further process the extracted flow data. Then, statistical models were used in order to identify the most important features for discriminating normal and abnormal instances, and ML techniques were trained and evaluated so as to assess the value of the dataset in comparison to other benchmark datasets [28]. The produced dataset contains both normal and attack network traffic based on benign scenarios and botnet scenarios, respectively. The botnet scenarios include probing, DoS, DDoS, data theft, and keylogging attacks.

The "Bot-IoT" dataset contains over 72 million records of network traffic, and a scaled-down version of the dataset with roughly 3.6 million records is also provided by the authors for evaluation purposes. However, the "Bot-IoT" dataset does not include a variety of attack types (e.g., ransomware and XSS cross-site scripting), as mentioned in [15]. Additionally, the "Bot-IoT" dataset was made available to the scientific community in 2018 and, thus, cannot be easily considered as the most recent and representative dataset containing information about normal or attack traffic of a current IoT network and information about the behavior of IoT devices when they function under normal operation conditions, as well as when they function under attack.

4.4. A Dataset for Detecting DoS Attacks on IoT Devices Using Network Traffic Traces

The authors in [29] created an IoT-based dataset by collecting both normal traffic and traffic generated when various types of DoS attacks (e.g., TCP SYN flooding, Ping of Death) are carried out. A testbed was designed and comprises (i) a TPLink gateway with OpenWrt firmware, (ii) several IoT devices (e.g., WeMo motion sensor, Samsung smart-camera, Philips Hue bulb), (iii) two attackers, and (iv) two victims. One attacker was placed locally (inside the LAN) and the other attacker existed remotely (on the Internet). Moreover, both attackers were capable of attacking both victims. In order to store the network packet traces of all of the network traffic, a 1 TB external hard disk was attached to the gateway. The packet traces were stored as "pcap" files using the "tcpdump" tool.

In addition, two types of attacks were implemented: (a) direct attacks (i.e., ARP spoofing, TCP SYN flooding, UDP flooding, and Ping of Death), and (b) reflection attacks (i.e., SNMP, SSDP, TCP SYN, and Smurf). All of the types of DoS attacks were performed using different traffic rates (i.e., how many packets were sent to the victim). Furthermore, the attacks originated from either one of the attackers or both of them and targeted either one of the victims or both of them.

The authors made their dataset available to the community. The released dataset refers to a one-month period of benign and attack traffic relating to ten IoT devices, and annotations of those attacks are included. The dataset consists of 30 "pcap" files, and each file corresponds to a trace collected over a day [29]. Nevertheless, this dataset does not have a variety of attack types (e.g., ransomware and XSS cross-site scripting), as mentioned in [15]. In addition, similarly to the "Bot-IoT" dataset mentioned in the previous subsection, this dataset was made available to the community in 2018 and, therefore, cannot be easily considered as the most recent and representative dataset containing information about normal or attack traffic of a current IoT network and information about the behavior of IoT devices when they function under normal operation conditions, as well as when they function under attack.

4.5. ToN_IoT Telemetry Dataset

The "TON_IoT Telemetry" dataset includes events of a variety of IoT-related attacks and legitimate scenarios, IoT telemetry data collected from heterogeneous IoT/IIoT data

sources, network traffic of the IoT/IIoT network, and audit traces of operating systems. Each of the classes of the "TON_IoT Telemetry" dataset describes either a normal record or the related type of attack in the case of an attack record. In [15], the authors presented the testbed that they developed in order to generate the "TON_IoT Telemetry" dataset [32]. The authors developed a testbed integrating IoT sensors (e.g., weather and modbus sensors), physical network components (e.g., switches, routers), several virtual machines (e.g., VMs of offensive Kali systems, VMs of Windows client systems), hacking platforms, cloud platforms, and fog platforms, and the testbed components were organized into the three layers of "Edge", "Fog", and "Cloud". In addition, the testbed employed a software-defined network (SDN) and network function virtualization (NFV) through the NSX-VMware platform [33]. The NSX-VMware platform enabled: (a) the establishment of a virtualized "Fog" layer and a virtualized "Cloud" layer that simultaneously operated to offer the IoT/IIoT and network services; (b) the emulation and control of multiple virtual machines (VMs) in the testbed for both hacking and normal operations, and (c) the management of the interaction between the three layers.

4.5.1. Testbed "Edge" Layer

The "Edge" layer is fundamental in IoT/IIoT applications because its devices measure real-world physical conditions and transmit the collected information to the "Fog" or "Cloud" for further analysis [34]. The "Edge" layer of the testbed contains various IoT/IIoT devices (e.g., weather and light bulb sensors) and physical gateways (i.e., routers and switches) to the Internet, as well as host systems. Besides, the "Edge" layer includes the physical host systems "NSX-VMware Server" and "vSphere System" used to deploy the "Fog" layer and the "Cloud" layer, respectively, by means of virtualization through the NSX-VMware platform [33] and the NSX-VMware hypervisor platform, respectively. The "Edge" layer of the testbed is linked to the "Fog" layer through the "vSwitch".

4.5.2. Testbed "Fog" Layer

The purpose of the "Fog" layer is to extend the Cloud computing and services to the "Edge" layer of the network in order to provide limited computing capacity and storage near to the data sources [34]. The "Fog" layer of the testbed consists of the VMs and the virtualization technology that manages the VMs and their services using the NSX-VMware platform [15].

4.5.3. Testbed "Cloud" Layer

The general purpose of the "Cloud" layer is to host large-size data centers with a significant capacity for both computation power and storage in order to support IoT/IIoT applications and satisfy the resource requirements for big data analysis.

4.5.4. ToN_IoT Datasets

The authors in [15] simulated several different types of attack scenarios (i.e., scanning, DoS, DDoS, ransomware, backdoor, data injection, cross-site scripting (XSS), password cracking, and man-in-the-middle (MITM)) on their testbed, and collected data from the different components of their testbed in dataset files. All of the datasets are provided in files that follow the "csv" (comma separated vector) format. The datasets files are split into two main folders: (i) the "Processed" datasets folder, and (ii) the "Train_Test" datasets folder.

The "Processed" datasets contain a processed and filtered version of the datasets with: (a) their standard features, (b) a label feature indicating whether an observation is normal or malicious, and (c) a type feature indicating the attacks' sub-classes for multi-class classification problems [15]. On the other hand, the "Train_Test" datasets contain selected records of the "Processed" datasets that were used by the authors in [15] as training and testing datasets for training and evaluating the accuracy and efficiency of various ML algorithms.

Both the "Processed" datasets and the "Train_Test" datasets consist of four types of dataset files (i.e., "Network", "IoT", "Linux", "Windows"), with each referring to either the network traffic or a specific type of device (e.g., sensor, server, desktop) of the testbed, as also demonstrated in Figure 1. In particular, the "Network" datasets contain the traffic data that passed through the entire testbed and were captured during the simulations, whereas the "IoT" datasets contain the data related to each of the seven IoT/IIoT sensors that were simulated in the testbed. Finally, the "Linux" datasets and the "Windows" datasets contain the data relating to the two Ubuntu systems and the two Windows systems in the testbed, respectively.

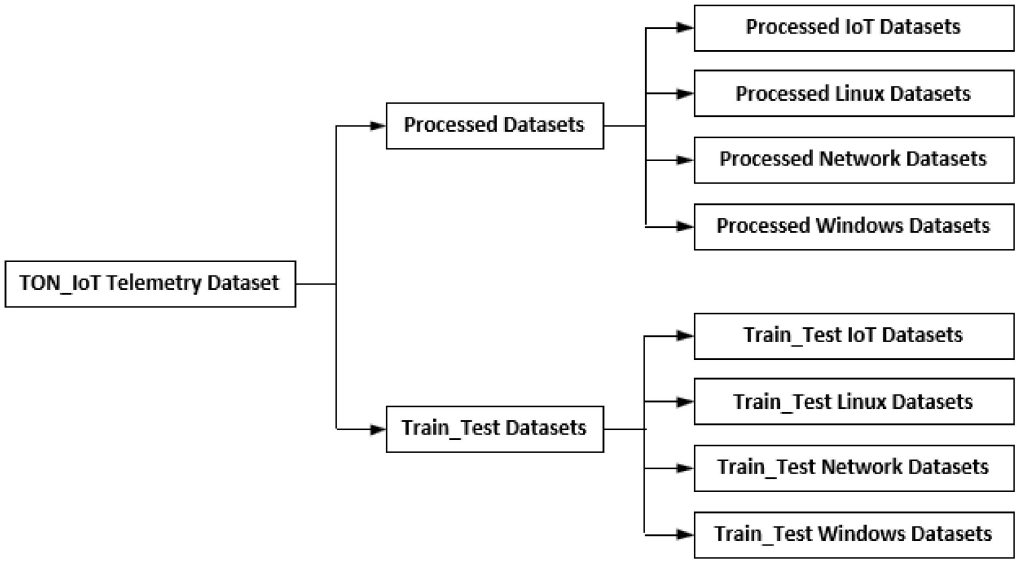

Figure 1. ToN_IoT Telemetry datasets hierarchy.

In our experiments, in order to train and evaluate the selected ML algorithms, we focused on the "Train_Test Network" datasets containing files with network-related data. In particular, the "Train_Test Network" datasets contain files with the traffic data that passed through the entire testbed and were captured during the simulations. Table 3 shows the 45 features of the "Train_Test Network" datasets along with their descriptions.

Table 3. Features and respective descriptions of the "Train_Test Network" datasets.

ID	Feature	Description
1	ts	Timestamp of connection between flow identifiers
2	src_ip	Source IP addresses that originate endpoints' IP addresses
3	src_port	Source ports that originate endpoint's TCP/UDP ports
4	dst_ip	Destination IP addresses that respond to endpoint's IP addresses
5	dst_port	Destination ports that respond to endpoint's TCP/UDP ports
6	proto	Transport layer protocols of flow connections
7	service	Dynamically detected protocols, such as DNS, HTTP, and SSL
8	duration	The time of the packet connections, which is estimated by subtracting "time of last packet seen" and "time of first packet seen"
9	src_bytes	Source bytes that are originated from payload bytes of TCP sequence numbers
10	dst_bytes	Destination bytes that are responded payload bytes from TCP sequence numbers

Table 3. *Cont.*

ID	Feature	Description
11	conn_state	Various connection states, such as S0 (connection without replay), S1 (connection established), and REJ (connection rejected)
12	missed_bytes	Number of missing bytes in content gaps
13	src_pkts	Number of original packets that is estimated from source systems
14	src_ip_bytes	Number of original IP bytes that is the total length of IP header field of source systems
15	dst_pkts	Number of destination packets that is estimated from destination systems
16	dst_ip_bytes	Number of destination IP bytes that is the total length of IP header field of destination systems
17	dns_query	Domain name subjects of the DNS queries
18	dns_qclass	Values that specify the DNS query classes
19	dns_qtype	Value that specifies the DNS query types
20	dns_rcode	Response code values in the DNS responses
21	dns_AA	Authoritative answers of DNS, where T denotes server is authoritative for query
22	dns_RD	Recursion desired of DNS, where T denotes request recursive lookup of query
23	dns_RA	Recursion available of DNS, where T denotes server supports recursive queries
24	dns_rejected	DNS rejection, where the DNS queries are rejected by the server
25	ssl_version	SSL version that is offered by the server
26	ssl_cipher	SSL cipher suite that the server chose
27	ssl_resumed	SSL flag indicates the session that can be used to initiate new connections, where T refers to the SSL connection being initiated
28	ssl_established	SSL flag indicates establishing connections between two parties, where T refers to establishing the connection
29	ssl_subject	Subject of the X.509 cert offered by the server
30	ssl_issuer	Trusted owner/originator of SLL and digital certificate (certificate authority)
31	http_trans_depth	Pipelined depth into the HTTP connection
32	http_method	HTTP request methods, such as GET, POST, and HEAD
33	http_uri	URIs used in the HTTP request
34	http_version	The HTTP versions utilized, such as V1.1
35	http_request_body_len	Actual uncompressed content sizes of the data transferred from the HTTP client
36	http_response_body_len	Actual uncompressed content sizes of the data transferred from the HTTP server
37	http_status_code	Status codes returned by the HTTP server
38	http_user_agent	Values of the User-Agent header in the HTTP protocol
39	http_orig_mime_types	Ordered vectors of mime types from source system in the HTTP protocol
40	http_resp_mime_types	Ordered vectors of mime types from destination system in the HTTP protocol
41	weird_name	Names of anomalies/violations related to protocols that happened
42	weird_addl	Additional information is associated to protocol anomalies/violations
43	weird_notice	Indicates if the violation/anomaly was turned into a notice
44	label	Tags normal and attack records, where 0 indicates normal and 1 indicates attacks
45	type	Tags attack categories, such as normal, DoS, DDoS, and backdoor attacks, and normal records

4.6. IoT Device Behavior Datasets

Behavior datasets of IoT devices play a significant role in the deployment of a more accurate and efficient AIDS for IoT networks. However, despite the recent efforts focused on the generation of IoT-specific datasets, and also mentioned in the previous subsections, the generated datasets are limited in terms of information related to the behavior of IoT devices. Therefore, more efforts are required toward datasets including information about the behavior of IoT devices when they function under normal operation conditions, as well as when they function under attack. To this direction, and to the best of our knowledge, a first step is the IoT device behavior datasets generated by the work in [7]. The IoT device behavior datasets include information related to the behavior of the IoT devices based on a simulated benign scenario and a simulated malicious scenario. The classes of the IoT device behavior datasets are two (i.e., normal behavior, abnormal behavior) corresponding to the case of a binary classification problem. The authors in [7] utilized the Cooja simulator of the open source Contiki operating system (OS) [35] in order to simulate a benign IoT network scenario and a malicious IoT network scenario, and generate corresponding benign and malicious datasets. Each simulated scenario utilized five UDP-client motes and one UDP-server mote. The type of each mote was the "Tmote Sky", which is an ultralow power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping [36].

The "benign" scenario utilized only "benign" motes and produced the "benign" datasets, which include only normal events. The "malicious" scenario utilized four "benign" UDP-client motes, one "malicious" UDP-client mote, and one "benign" UDP-server mote, and produced the "malicious" datasets, which contain both attack and normal events. Both the "benign" datasets and the "malicious" datasets are further divided into their respective "powertrace" datasets and "network traffic" datasets. The "powertrace" datasets includes information collected every 2 s on the energy consumption (i.e., behavior-related information) of the motes (i.e., IoT devices) of the simulated IoT network, whereas the "network traffic" datasets contain records related to the IoT network traffic features, such as the source/destination IPv6 address, packet size, and communication protocol [7]. In our experiments, we needed IoT device behavior datasets so as to train and test ML algorithms for the proposed AIDS, and, thus, we utilized both the "benign powertrace" dataset and the "malicious powertrace" dataset. Table 4 shows the features of the "benign powertrace" dataset and the "malicious powertrace" dataset, along with their descriptions.

Table 4. Features and respective descriptions of the "benign powertrace" dataset and the "malicious powertrace" dataset.

Feature	Description
sim time	simulation time
clock_time()	clock time (i.e., by default, 128 ticks/second)
ID	Mote ID
P	label
rimeaddr	rime address
seqno	sequence number
all_cpu	accumulated CPU energy consumption during the simulation
all_lpm	accumulated low power mode energy consumption during the simulation
all_transmit	accumulated transmission energy consumption during the simulation
all_listen	accumulated listen energy consumption during the simulation
all_idle_transmit	accumulated idle transmission energy consumption during the simulation
all_idle_listen	accumulated idle listen energy consumption during the simulation
cpu	CPU energy consumption for this cycle of 2 s
lpm	LPM energy consumption for this cycle of 2 s
transmit	transmission energy consumption for this cycle of 2 s
listen	listen energy consumption for this cycle of 2 s
idle_transmit	idle transmission energy consumption for this cycle of 2 s
idle_listen	idle listen energy consumption for this cycle of 2 s

5. Scenario Architecture

The proposed AIDS is designed for the IoMT edge network of IoT-based healthcare systems, as shown in Figure 2. The IoMT edge network interacts with objects, such as physical things (i.e., patient's body and patient's environment), through the IoMT devices of the IoMT edge network. Specifically, the IoMT edge network integrates:

- "bio-sensors", a type of IoMT sensor, whose purpose is to collect vital signs (e.g., blood pressure, body temperature) of the patient;
- "context-aware sensors", another type of IoMT sensor, for gathering context information (e.g., air pressure, humidity, or room temperature) from the patient environment;
- "IoMT actuators", for supporting the real-time provisioning of medical treatment (e.g., an insulin pump, which may be controlled remotely to inject the patient with insulin).

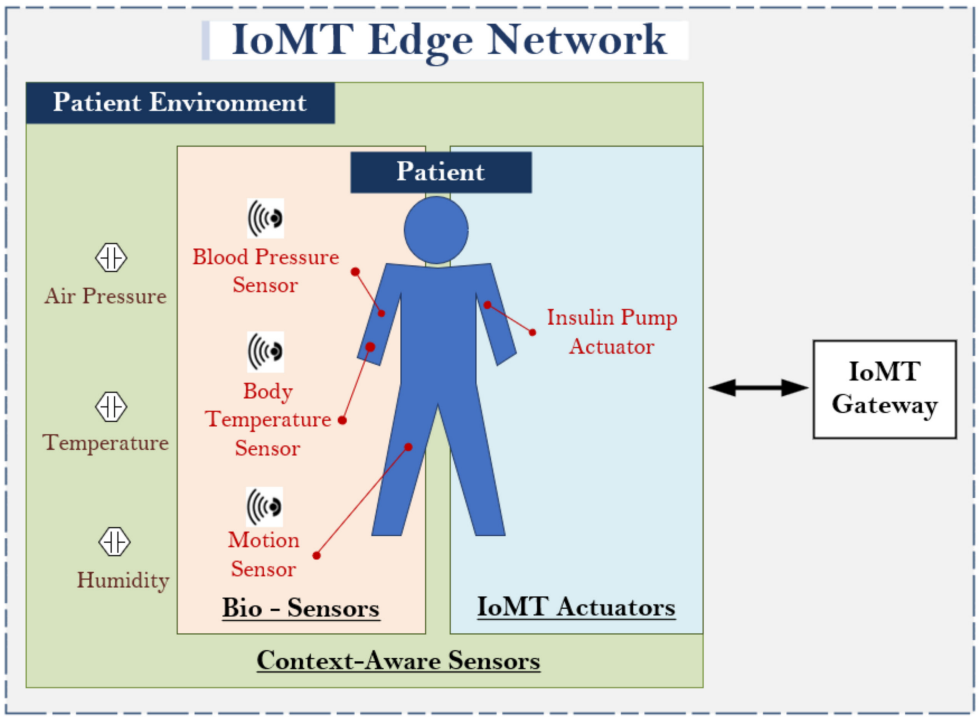

Figure 2. IoT-based health monitoring system perception domain.

In other words, the main purpose of the IoMT edge network is to measure, collect, and handle the information provided by the monitored physical things (i.e., patient's body and patient's environment), as well as to transmit the collected information through the IoMT gateway to the application layer, where the cloud-based healthcare platform of the IoT-based healthcare system is located, for processing, analysis, storage, and decision making. Furthermore, the IoMT edge network facilitates the reception of the appropriate control commands from the application layer to the actuators again through the IoMT gateway.

6. Proposed Anomaly-Based IDS

6.1. System Description

The purpose of the proposed AIDS is to protect the IoMT edge network and its IoMT devices and gateway from internal and external threats that may exploit the inherent security vulnerabilities of IoT technology, by taking into consideration not only all of the current known IoT attack vectors but also unknown ones that may both appear in the future at all four layers of the ITU-T IoT reference model [37] and target the IoMT edge network, its IoMT devices, or the gateway. The proposed AIDS consists of (a) a set of distributed monitoring and data acquisition (MDA) components (i.e., a monitoring and

data acquisition component runs on each IoMT device deployed and interconnected in the IoMT edge network), and (b) a central detection (CD) component (i.e., detection engine) running on the gateway, as illustrated in Figure 3.

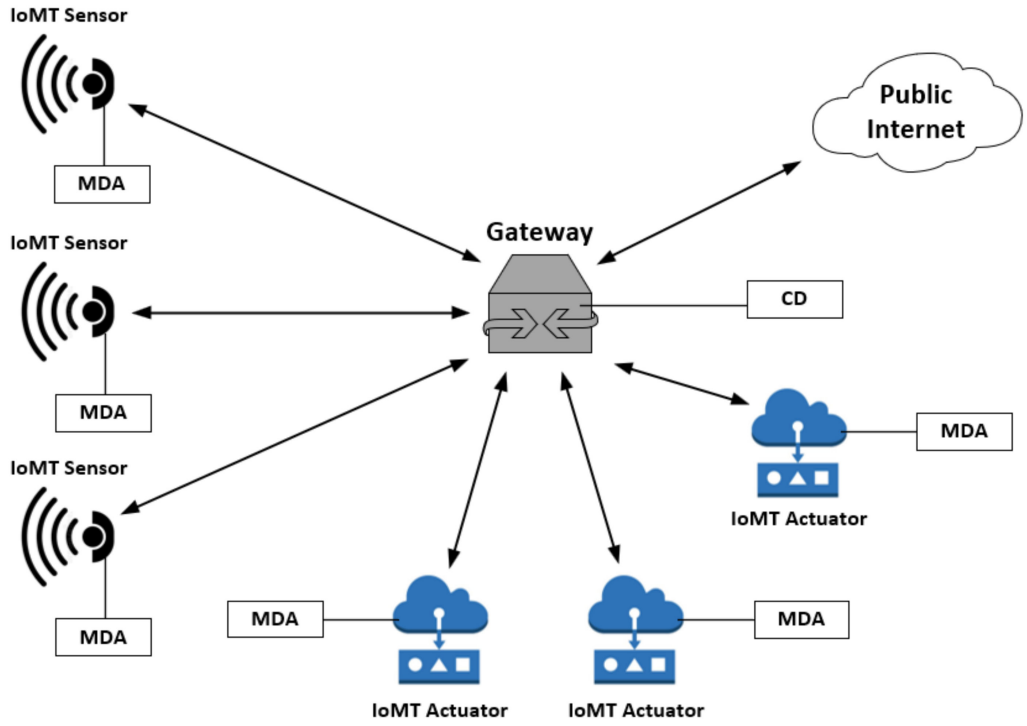

Figure 3. The MDA components and the CD component of the proposed AIDS in the IoMT edge network.

At this point, it is important to mention the requirements that the gateway and the IoMT devices have to meet in order to support the proposed AIDS. First of all, the gateway is required to: (a) be capable of accessing behavior data about itself (e.g., energy consumption); (b) be capable of capturing packets and network-related information regarding the IoMT devices of the IoMT edge network; and (c) have enough resources (e.g., Raspberry Pi 4 Model B) to support both the standard operations as a relay node and the functions of the CD component running on it. On the other hand, each IoMT device is essential to: (a) be capable of accessing behavior data about themselves (e.g., energy consumption); and (b) have enough resources to handle both its normal operation as a sensor or actuator and the functions of the MDA component running on it.

6.2. Monitoring and Data Acquisition (MDA) Component

The monitoring and data acquisition (MDA) component runs on each IoMT device (i.e., IoMT sensor and IoMT actuator) connected to the gateway. Its aim is to monitor the behavior of the IoMT device hosting it and to collect relevant device behavior data, such as the CPU energy consumption, during a specific MDA period (i.e., sampling period). Furthermore, the MDA component is responsible for sending the collected data to the gateway as an MDA report. The MDA component consists of the following modules, whose relations are shown in Figure 4.

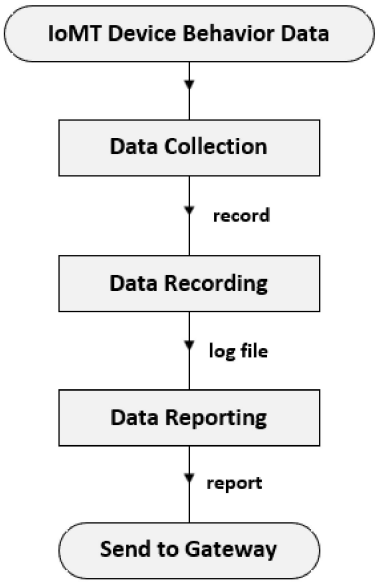

Figure 4. The monitoring and data acquisition (MDA) component.

"Data Collection" module: Collects data on run-time for the set of features summarized in Table 5, during the sampling period (i.e., a specific MDA period), and creates a record in CSV format;

Table 5. Summary of features collected by the data collection module of the MDA component and their descriptions.

Feature	Description
CPU usage	Amount/percentage of used CPU resources
CPU processes	Amount of active CPU processes
MEM usage	Amount/percentage of used internal memory resources
Disk usage	Amount/percentage of used external storage resources
Wi-Fi usage	Amount of bandwidth used by the Wi-Fi interface
Set of energy consumption features	Set of features (e.g., "powertrace" features in [7]) regarding energy consumption during the different modes of the IoMT device

"Data Recording" module: Writes the records created by the "data collection" module in log files in CSV format. Each log file contains a specific number of records defined by the maximum size of the log file;

"Data Reporting" module: Constructs reports including the log files created by the "data recording" module. Each report can contain a specific number of log files defined by the maximum number of log files for the given report. In principle, the "data reporting" module accumulates the continuously produced log files by the "data recording" module, and, when the number of the accumulated log files reach a certain value (i.e., maximum number of log files for the given report), they are grouped into a report together with the IoMT device ID. Finally, the report is transmitted to the gateway. To reduce the required bandwidth for reports transmission, the "data reporting" module may perform compression on the report if the host IoMT device has enough resources.

6.3. Central Detection (CD) Component

The central detection (CD) component runs on the gateway of the IoMT edge network. The aim of the CD component is to:

- Monitor the behavior of the gateway hosting it and collect relevant behavior data, such as the accumulated CPU energy consumption, during a specific monitoring period (i.e., sampling period);

- Monitor the network traffic passing through the gateway and gather relevant network traffic data, such as source IP address, destination IP address, connection status information, and packet content information, during a specific monitoring period (i.e., sampling period);
- Receive the reports transmitted by the MDA components running on the IoMT devices that are connected to the gateway;
- Leverage the aforementioned data in order to identify whether an attack incident has occurred in the IoMT edge network, and trigger a corresponding security alert.

The CD component consists of the following modules, whose relations are shown in Figure 5.

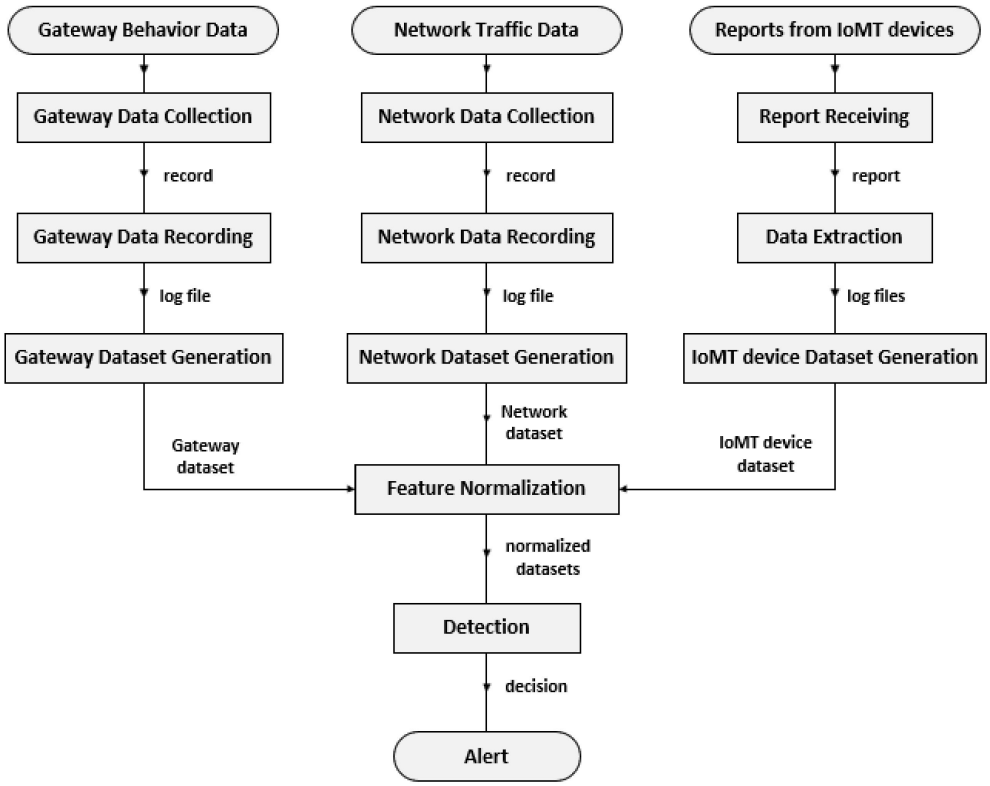

Figure 5. The central detection (CD) component.

"Gateway Data Collection" module: Collects behavior data, regarding the gateway, on the run-time for the set of features summarized in Table 6, during the sampling period, and creates a record in CSV format;

Table 6. Summary of features collected by the gateway data collection module of the CD component and their descriptions.

Feature	Description
CPU usage	Amount/percentage of used CPU resources
CPU processes	Amount of active CPU processes
MEM usage	Amount/percentage of used internal memory resources
Disk usage	Amount/percentage of used external storage resources
Wi-Fi usage	Amount of bandwidth used by the Wi-Fi interface
Set of energy consumption features	Set of features regarding energy consumption during the different modes of the gateway

"Gateway Data Recording" module: Writes the records created by the "gateway data collection" module in log files in CSV format. Each log file contains a specific number of records defined by the maximum size of the log file;

"Gateway Dataset Generation" module: Utilizes the log files created by the "gateway data recording" module and generates a gateway dataset in CSV format. Each gateway dataset is generated based on a specific number of log files defined by the maximum number of log files for the given gateway dataset. The value of the maximum number of log files for the given gateway dataset highly depends on the requirements of the detection engine (i.e., detection module);

"Network Data Collection" module: Collects data, regarding the traffic passing through the gateway, on the run-time for the set of features summarized in Table 7, during the sampling period, and creates a record in CSV format;

Table 7. Summary of features collected by the network data collection module of the CD component, and their descriptions.

Feature	Description
Source IP address	Source IP address of the sender endpoint
Destination IP address	Destination IP address of the sender endpoint
Packet size	Length of packet in bytes
Communication protocol information features	Features related to the protocol used for the transmission of the packet

"Network Data Recording" module: Writes the records created by the "network data collection" module in log files in CSV format. Each log file contains a specific number of records defined by the maximum size of the log file;

"Network Dataset Generation" module: Uses the log files created by the "network data recording" module and generates a network dataset in CSV format. Each network dataset is generated based on a specific number of log files defined by the maximum number of log files for the given network dataset. The value of the maximum number of log files for the given network dataset highly depends on the requirements of the detection engine (i.e., detection module);

"Report Receiving" module: Receives the reports sent by the MDA components running on every IoMT device that is connected to the gateway. Each received report contains the device ID in order to know the device from which it originates;

"Data Extraction" module: Extracts the log files that are included in each report that was received by the "report receiving" module. All of the log files extracted from a given report originate from the same IoMT device and are associated with the ID of the given IoMT device;

"IoMT Device Dataset Generation" module: Utilizes the log files that originate from a specific IoMT device and that were produced by the "data extraction" module in order to generate an IoMT device dataset in CSV format. The IoMT device dataset is generated based on a specific number of log files defined by the maximum number of log files for the given IoMT device dataset. The value of the maximum number of log files for the given IoMT device dataset highly depends on the requirements of the detection engine (i.e., detection module). In the case where there are multiple IoMT devices connected to the gateway, their corresponding IoMT device datasets are generated separately;

"Feature Normalization" module: Receives a gateway dataset, a network dataset, and one or multiple IoMT device datasets, and performs normalization on the features of the datasets. Each dataset is normalized independently from the other datasets and, thus, the module creates the following three different types of normalized datasets in CSV format: normalized gateway dataset, normalized network dataset, and normalized IoMT device dataset;

"Detection" module: Receives the three types of normalized datasets and detects whether or not an intrusion has occurred on the network consisting of the gateway and

its connected IoMT devices. In the case of an intrusion incident, an alarm is triggered. In principle, this is the core module of the proposed AIDS and leverages ML algorithms in order to identify known, as well as still unknown, attacks that may target the IoMT edge network, its IoMT devices, or the gateway. Additionally, since different normalized datasets are provided to the "detection" module, it can perform detection operations for each one of the different devices (i.e., gateway and connected IoMT devices) and, thus, can identify from where an attack originates or which device (i.e., IoMT device or gateway) is compromised.

7. Performance Evaluation

In this section, we focus on the performance evaluation of the following most popular ML algorithms for IoT AIDS [13,15], when they are applied for anomaly detection in the proposed AIDS: decision tree (DT), naïve Bayes (NB), linear regression (LR), random forest (RF), k-nearest neighbor (KNN), and support vector machines (SVM). Using four-fold cross validation, we trained and tested these algorithms over the same datasets consisting of (a) a specific part of the "Train_Test Network" dataset of the "TON_IoT Telemetry" dataset [32], and (b) the "benign powertrace" dataset and the "malicious powertrace" dataset produced following the approach in [7].

It is worthwhile to mention that, from the "Train_Test Network" dataset of the "TON_IoT Telemetry" dataset, we kept only the network records related to the edge layer of the testbed where the IoT devices are deployed, as described in [15]. For this reason, the desired network records were isolated from the "Train_Test_Network" dataset based on the IP address of the sender node and the destination node, and a record was kept only if the IP address of the sender node or the destination node belonged to the IP address range of the edge layer. Then, the isolated network records were merged into a new dataset, which will be referred to as the "Train_Test edge network" dataset.

Moreover, regarding the "benign powertrace" dataset and the "malicious powertrace" dataset from [7], we merged the two datasets into a new dataset by adding the records of the second dataset to the end of the records of the first dataset, as both datasets have the same features. The generated new dataset will be referred to as the "all_powertrace" dataset.

In our experiments, the Python language version 3.8.2 was used, along with the Scikit-Learn [38] library. We utilized specific functions of the Scikit-Learn library and a Python script was created utilizing these functions in order to perform the training and testing of the ML algorithms. Table 8 presents the functions that we utilized from the Scikit-Learn package, along with an explanation of how these functions were used.

Table 8. Utilized functions of the Scikit-Learn package, along with a brief explanation of how they were used.

Function	Explanation of Usage
OrdinalEncoder()	Pre-processing of the data of the datasets
ColumnTransformer()	Pre-processing of the data of the datasets
MinMaxScaler()	Normalization of the data of the datasets
train_test_split()	Split of a dataset into training and testing parts
DecisionTreeClassifier()	Implementation of a decision tree algorithm to train and evaluate
RandomForestClassifier()	Implementation of a random forest algorithm to train and evaluate
LogisticRegression()	Implementation of a logistic regression algorithm to train and evaluate
GaussianNB()	Implementation of a naïve Bayes algorithm to train and evaluate
SVC()	Implementation of a support vector machine algorithm to train and evaluate
KNeighborsClassifier()	Implementation of a k-nearest neighbor algorithm to train and evaluate
StratifiedKFold()	Split of a training part of a dataset into k-folds to perform k-fold cross validation
cross_validate()	Performing k-fold cross validation

7.1. Dataset Pre-Processing and Normalisation

It is necessary to prepare the datasets before they are utilized to train and test the ML algorithms. The preparation of the data includes data pre-processing and data normaliza-

tion. In our case, the pre-processing step involved the removal of unnecessary features and the conversion of the nominal values of the categorical features to numeric values.

7.1.1. Dataset Pre-Processing

Initially, the feature "ts" was omitted from all records of the "Train_Test edge network" dataset because this feature may cause some ML algorithms to overfit the training data, as also highlighted by the authors in [15].

As far as the "all_powertrace" dataset is concerned and similar to the feature "ts" of the "Train_Test edge network" dataset, the feature "clock_time" was filtered out. In addition, the features related to the simulation time (i.e., "sim time" feature) or the simulation duration (i.e., "all_cpu", "all_lpm", "all_transmit", "all_listen", "all_idle_transmit", "all_idle_listen", and "seqno" features) were filtered out from the "all_powertrace" dataset. Additionally, the "P" feature was omitted, because it only has a fixed value throughout all of the collected records of the "all_powertrace" dataset.

Finally, the nominal values of the categorical features of the "Train_Test edge network" dataset and the "all_powertrace" dataset were converted to numeric values to facilitate their use by the ML algorithms. For example, if a feature possessed the values of "on" and "off", these values were converted to "1" and "0", respectively. This was achieved by employing a label-encoding method [17].

7.1.2. Normalization

After the conversion of the values of all nominal features was completed in the pre-processing step, the data normalization step was then performed to the numeric values of each feature. If the values of a feature are significantly larger compared to the values of other features, this may lead to inaccurate results. Thus, data normalization helps to ensure that features with significantly large values do not outweigh features with smaller values. To achieve this, all of the features' values are scaled within the range of [0.0, 1.0] by performing a min–max normalization process on each feature. This normalization process is described by the following equation:

$$z = (x - x_{min})/(x_{max} - x_{min}) \quad (8)$$

where z is the normalized value (i.e., after scaling), x is the value before scaling, and x_{max} and x_{min} are the maximum and minimum values of the feature, respectively.

7.2. Training Process of ML Algorithms

The selected ML algorithms were trained and tested separately over (a) the "Train_test edge network" dataset, and (b) the "all_powertrace" dataset. Initially, each of the two datasets was split into two parts: the train part and the test part. The train part consisted of 80% of the dataset and the ML algorithms were trained and evaluated with this part. On the other hand, the test part consisted of 20% of the dataset and was held back for further evaluation of the models with unseen data. The percentage split of 80% train–20% test was determined according to [17] as the best ratio to avoid the overfitting problem. After that, the training process of each ML algorithm over each dataset was performed using the four-fold cross validation method. According to this method, the training dataset is divided into four subsets of equal size and the records of each subset are randomly selected. The training process is repeated four times. Each time, three of the four subsets are utilized for the training of the ML algorithm and the remaining subset is used for validation. The final performance results are produced by averaging the results of the four folds [17]. Table 9 presents a summary of the hyperparameters of each of the six ML algorithms when the ML algorithm requires a hyperparameter to be set.

Table 9. Summary of the hyperparameters of each ML algorithm when the ML algorithm requires a hyperparameter to be set.

ML Algorithm	Hyperparameters
Decision Tree	(1) The Gini index was used to select tree nodes. (2) Minimum samples per leaf node set to 10
Random Forest	(1) The Gini index was used to select tree nodes. (2) The minimum samples per leaf node was set to 10. (3) The random forest consisted of 10 decision trees.
Naïve Bayes	The Gaussian variant of the NB algorithm was used.
Logistic Regression	-
Support Vector Machine	The Gaussian radial basis function (RBF) was set as the kernel function.
K-Nearest Neighbor	(1) The value of K was set to 5. (2) The Euclidean distance was set as the distance metric.

7.3. Performance Evaluation Results

The selected ML algorithms were trained and tested on the "Train_Test edge network" dataset and the "all_powertrace" dataset for binary classification, using the four-fold cross validation method. The performance of the selected ML algorithms was evaluated by the evaluation metrics of accuracy, precision, recall, and F1-score. The numerical results of the evaluation metrics for the selected ML algorithms, when applied to the "Train_Test edge network" dataset, are shown in Table 10 and Figure 6.

Table 10. Evaluation metrics for binary classification for the "Train_Test edge network" dataset.

ML Algorithm	Accuracy	Precision	Recall	F1-Score
DT	0.9997	0.9997	0.9991	0.9994
NB	0.3444	0.2791	0.9997	0.4364
LR	0.9870	0.9552	0.9955	0.9750
RF	0.9996	0.9989	0.9995	0.9992
KNN	0.9998	0.9995	0.9997	0.9996
SVM	0.9873	0.9530	0.9993	0.9756

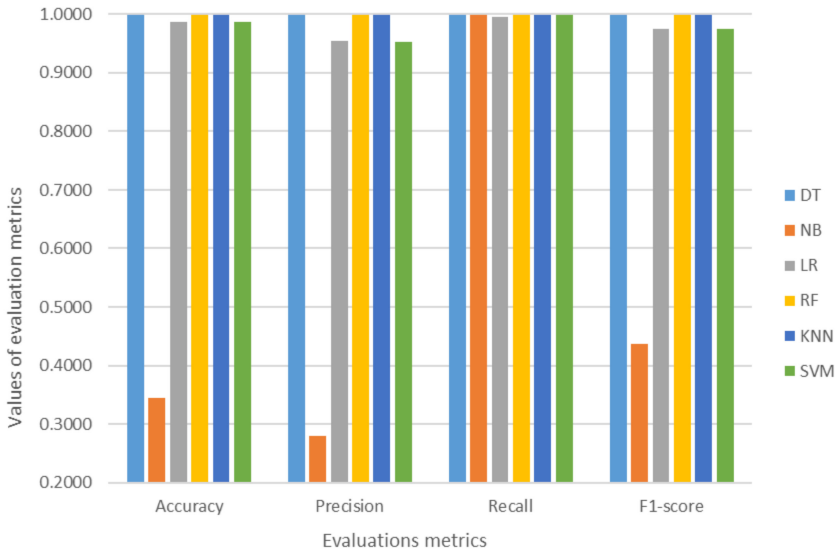

Figure 6. Evaluation metrics for binary classification for the "Train_Test edge network" dataset.

It can be easily observed that almost all of the ML algorithms demonstrate a high performance for the "Train_Test edge network" dataset. The DT, RF, and KNN algorithms show an almost perfect accuracy score (i.e., 0,99), followed by the LR and SVM methods (i.e., 0,98). The same trend can be seen in the precision, recall, and F1-score, which are extremely high. However, the NB classifier performs significantly worse than the rest of the algorithms in almost all evaluation metrics. In particular, although the NB method achieves a very high recall (0,99), it demonstrates a low accuracy (i.e., 0.34), precision (i.e., 0,28), and F1-score (i.e., 0,44) due to dependencies among the features of the "Train_Test edge network" dataset.

Furthermore, the "all_powertrace" dataset was also used to train and test the selected ML algorithms for binary classification, using the four-fold cross validation method. The performance of the selected ML algorithms was also evaluated by the evaluation metrics of accuracy, precision, recall, and F1-score. The numerical results of the evaluation metrics for the selected ML algorithms are shown in Table 11 and Figure 7.

Table 11. Evaluation metrics for binary classification for the "all_powertrace" dataset.

ML Algorithm	Accuracy	Precision	Recall	F1-Score
DT	0.9889	0.9742	0.9587	0.9664
NB	0.9613	0.8218	0.9805	0.8942
LR	0.9785	0.9378	0.9326	0.9352
RF	0.9900	0.9718	0.9684	0.9701
KNN	0.9887	0.9752	0.9566	0.9658
SVM	0.9785	0.9375	0.9333	0.9354

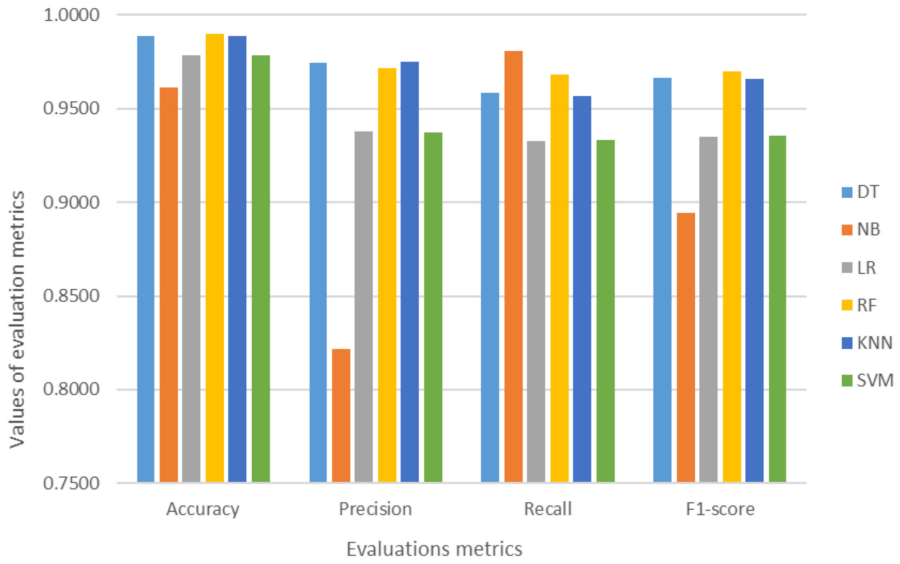

Figure 7. Evaluation metrics for binary classification for the "all_powertrace" dataset.

Similarly to the results related to the "Train_Test edge network" dataset, all of the ML algorithms demonstrated an extremely high accuracy, with the lowest accuracy value being 0.96. The DT, RF, and KNN classifiers showed an almost perfect precision that was close to 0.99, whereas the NB, LR, and SVM classifiers demonstrated a high precision that was between 0.82 and 0.94. Moreover, all of the ML algorithms showed a high recall and high F1-score, with the lowest values being 0.93, and 0.89 respectively. It is noteworthy to mention that the performance of NB has improved due to reduced dependencies among the features of the "all_powertrace" dataset.

The evaluation results demonstrate that the DT, RF, and KNN algorithms are more suitable to be used as the core of the detection component (i.e., CD component). Based on the

above tables and figures, the DT, RF, and KNN algorithms presented high values regarding all of the evaluation metrics (i.e., accuracy, precision, recall, F1-score) while being trained and testing using either the "Train_Test edge network" dataset or the "all_powertrace" dataset. In both cases, the lowest values for all of the three algorithms (i.e., DT, RF, KNN) regarding the accuracy, precision, recall, and F1-score were 0.99, 0.97, 0.96, and 0.97, respectively. Therefore, it is evident that, among the six selected popular ML algorithms, the DT, RF, and KNN algorithms are the three best algorithms based on their performance.

8. Challenges and Future Work

As future work, we intend to develop a prototype of the proposed AIDS in order to evaluate its performance in terms of computational overhead on the gateway and the sensors. In particular, the next first step is the implementation of the central detection (CD) component of the proposed AIDS, relying on DT, RF, and KNN algorithms for anomaly detection, on a Raspberry Pi 4 device that plays the role of the gateway in an IoMT network. The DT, RF, and KNN will be implemented to run on the Raspberry Pi 4 device, and their computational overhead will be evaluated. In addition, the monitoring and data acquisition (MDA) component is planned to be implemented for an IoMT sensor (i.e., MTM-CM5000-MSP sensor) that will be connected to the gateway (i.e., Raspberry Pi 4 Model B device). The developed MDA component will be evaluated in terms of its computational overhead.

Another direction of our future work is the usage of our proposed AIDS for the case of multi-class classification. In our work, we have considered binary classification, meaning that the proposed AIDS is capable of identifying whether there is an attack incident or not. However, as the proposed AIDS is able to distinguish between normal and malicious incidents (i.e., binary classification), the next step is the improvement of the AIDS to support multi-class classification by integrating the ability of discerning which type of attack correlates to a specific malicious incident in the detection engine of the CD component, while, at the same time, considering the computational overhead.

In addition, more ML algorithms will be implemented and evaluated in order to identify those that achieve a high accuracy, precision, recall, and F-score, while, at the same time, not introducing a high computational cost. It is also worthwhile to mention that the selection of the appropriate values of the hyperparameters of all ML algorithms under study will be properly investigated, while simultaneously taking into consideration the computational cost. The values of the hyperparameters may affect both the performance of the ML algorithms and their computational overhead. Therefore, extensive simulations will be carried out as future work in order to identify the appropriate hyperparameter(s) of the ML algorithms and achieve a balance between the performance and computational cost.

Finally, it is noteworthy to mention that, since the proposed AIDS is designed to be deployed in resource-constrained devices, deep learning techniques, which are complex, heavyweight, and are characterized by a high computational overhead, have not been considered. However, deep learning techniques, such as those mentioned in [39,40], could be considered, as future work, in the extension of the autonomous and lightweight proposed AIDS to a cloud-based AIDS for IoMT edge networks.

9. Conclusions

This paper proposed a new AIDS adapted to the constraints of IoMT networks so as to constitute an efficient and effective security solution for this type of networks. The proposed AIDS is to leverage host-based and network-based techniques to reliably monitor and collect log files from the IoMT devices and the gateway, as well as traffic from the IoMT edge network, taking into account the computational cost. The detection process of the proposed AIDS is to be implemented by the detection engine running on the gateway of the IoMT edge network and relying on machine learning (ML) techniques, considering the computation overhead, in order to detect abnormalities in the collected data and thus identify malicious incidents in the IoMT network. To evaluate potential detection ML

algorithms and identify the most appropriate algorithms for the proposed AIDS, we used (i) the network part of the "TON_IoT Telemetry dataset" [15] and (ii) a dataset that was produced according to the approach of the authors in [7] to train and test the following most popular ML algorithms for IoT AIDS: DT, NB, LR, RF, SVM, and KNN. The evaluation results demonstrate that the DT, RF, and KNN algorithms are more suitable to be used for the central detection (CD) component of the proposed AIDS. In addition, it is worthwhile to highlight that the performance of NB demonstrated better results when it was trained and tested on the "all_powertrace" dataset due to the reduced dependencies among the features of this dataset.

Author Contributions: Conceptualization and methodology, G.Z., I.E., G.M., K.P., and J.C.R.; software, G.Z., I.E., and J.C.R.; validation, G.Z., I.E., J.C.R., and G.M.; investigation, G.Z., I.E., J.C.R., and G.M.; resources, G.Z., I.E., J.C.R., and G.M.; writing—original draft preparation, G.Z. and I.E.; writing—review and editing, I.E., G.M., K.P., and J.R.; visualization, G.Z., J.C.R., and I.E.; supervision, G.M., K.P., and J.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received external funding.

Acknowledgments: The research work leading to this publication has received funding through the Moore4Medical project under grant agreement H2020-ECSEL-2019-IA-876190 within ECSEL JU in collaboration with the European Union's H2020 Framework Programme (H2020/2014-2020) and Fundação para a Ciência e Tecnologia (ECSEL/0006/2019).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rodrigues, J.J.P.C.; Segundo, D.B.D.R.; Junqueira, H.A.; Sabino, M.H.; Prince, R.M.I.; Al-Muhtadi, J.; De Albuquerque, V.H.C. Enabling Technologies for the Internet of Health Things. *IEEE Access* **2018**, *6*, 13129–13141. [CrossRef]
2. Papaioannou, M.; Karageorgou, M.; Mantas, G.; Sucasas, V.; Essop, I.; Rodriguez, J.; Lymberopoulos, D. A Survey on Security Threats and Countermeasures in Internet of Medical Things (IoMT). *Trans. Emerg. Telecommun. Technol.* **2020**, 4049. [CrossRef]
3. Islam, S.M.R.; Kwak, D.; Kabir, M.H.; Hossain, M.; Kwak, K.-S. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access* **2015**, *3*, 678–708. [CrossRef]
4. Makhdoom, I.; Abolhasan, M.; Lipman, J.; Liu, R.P.; Ni, W. Anatomy of Threats to the Internet of Things. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1636–1675. [CrossRef]
5. Zhang, M.; Raghunathan, A.; Jha, N.K. Trustworthiness of Medical Devices and Body Area Networks. *Proc. IEEE* **2014**, *102*, 1174–1188. [CrossRef]
6. Karageorgou, M.; Mantas, G.; Essop, I.; Rodriguez, J.; Lymberopoulos, D. Cybersecurity attacks on medical IoT devices for smart city healthcare services. In *IoT Technologies in Smart Cities: From Sensors to Big Data, Security and Trust*; Institution of Engineering and Technology (IET): London, UK, 2020; pp. 171–187.
7. Essop, I.; Ribeiro, J.C.; Papaioannou, M.; Zachos, G.; Mantas, G.; Rodriguez, J. Generating Datasets for Anomaly-Based Intrusion Detection Systems in IoT and Industrial IoT Networks. *Sensors* **2021**, *21*, 1528. [CrossRef]
8. Gope, P.; Hwang, T. BSN-Care: A Secure IoT-Based Modern Healthcare System Using Body Sensor Network. *IEEE Sens. J.* **2016**, *16*, 1368–1376. [CrossRef]
9. Alsubaei, F.; Abuhussein, A.; Shiva, S. Security and Privacy in the Internet of Medical Things: Taxonomy and Risk Assessment. In Proceedings of the 2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops), Singapore, 9 October 2017; pp. 112–120.
10. Ribeiro, J.; Saghezchi, F.B.; Mantas, G.; Rodriguez, J.; Abd-Alhameed, R.A. HIDROID: Prototyping a Behavioral Host-Based Intrusion Detection and Prevention System for Android. *IEEE Access* **2020**, *8*, 23154–23168. [CrossRef]
11. Ribeiro, J.; Saghezchi, F.B.; Mantas, G.; Rodriguez, J.; Shepherd, S.J.; Abd-Alhameed, R.A. An Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. *Mob. Netw. Appl.* **2019**, *25*, 164–172. [CrossRef]
12. Ribeiro, J.; Mantas, G.; Saghezchi, F.B.; Rodriguez, J.; Shepherd, S.J.; Abd-Alhameed, R.A. Towards an Autonomous Host-Based Intrusion Detection System for Android Mobile Devices. In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*; Springer: Cham, Switzerland, 2018; Volume 263, pp. 139–148.
13. Asharf, J.; Moustafa, N.; Khurshid, H.; Debie, E.; Haider, W.; Wahab, A. A Review of Intrusion Detection Systems Using Machine and Deep Learning in Internet of Things: Challenges, Solutions and Future Directions. *Electronics* **2020**, *9*, 1177. [CrossRef]
14. Thamilarasu, G.; Odesile, A.; Hoang, A. An Intrusion Detection System for Internet of Medical Things. *IEEE Access* **2020**, *8*, 181560–181576. [CrossRef]
15. Alsaedi, A.; Moustafa, N.; Tari, Z.; Mahmood, A.; Anwar, A.N. TON-IoT Telemetry Dataset: A New Generation Dataset of IoT and IIoT for Data-Driven Intrusion Detection Systems. *IEEE Access* **2020**, *8*, 165130–165150. [CrossRef]
16. Kotsiantis, S.B. Decision trees: A recent overview. *Artif. Intell. Rev.* **2011**, *39*, 261–283. [CrossRef]

17. Géron, A. *Hands-On Machine Learning with Scikit-Learn and Tensor Flow: Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media: Sebastopol, CA, USA, 2017.
18. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
19. Doshi, R.; Apthorpe, N.; Feamster, N. Machine Learning DDoS Detection for Consumer Internet of Things Devices. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 29–35. [CrossRef]
20. D'Agostini, G. A multidimensional unfolding method based on Bayes' theorem. *Nucl. Inst. Methods Phys. Res. A* **1995**, *362*, 487–498. [CrossRef]
21. Subba, B.; Biswas, S.; Karmakar, S. Intrusion Detection Systems using Linear Discriminant Analysis and Logistic Regression. In Proceedings of the 2015 Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 2015.
22. Huraj, L.; Horak, T.; Strelec, P.; Tanuska, P. Mitigation against DDoS Attacks on an IoT-Based Production Line Using Machine Learning. *Appl. Sci.* **2021**, *11*, 1847. [CrossRef]
23. Moustafa, N.; Hu, J.; Slay, J. A holistic review of Network Anomaly Detection Systems: A comprehensive survey. *J. Netw. Comput. Appl.* **2019**, *128*, 33–55. [CrossRef]
24. Verma, A.; Ranga, V. Machine Learning Based Intrusion Detection Systems for IoT Applications. *Wirel. Pers. Commun.* **2020**, *111*, 2287–2310. [CrossRef]
25. Bhuyan, M.H.; Bhattacharyya, D.K.; Kalita, J.K. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Commun. Surv. Tutor.* **2013**, *16*, 303–336. [CrossRef]
26. Suthaharan, S.; Alzahrani, M.; Rajasegarar, S.; Leckie, C.; Palaniswami, M. Labelled data collection for anomaly detection in wireless sensor networks. In Proceedings of the 2010 Sixth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP 2010, Brisbane, QLD, Australia, 7–10 December 2010; pp. 269–274.
27. Sivanathan, A.; Gharakheili, H.H.; Loi, F.; Radford, A.; Wijenayake, C.; Vishwanath, A.; Sivaraman, V. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Trans. Mob. Comput.* **2019**, *18*, 1745–1759. [CrossRef]
28. Koroniotis, N.; Moustafa, N.; Sitnikova, E.; Turnbull, B. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Gener. Comput. Syst.* **2019**, *100*, 779–796. [CrossRef]
29. Hamza, A.; Gharakheili, H.H.; Benson, T.A.; Sivaraman, V. Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity. In Proceedings of the 2019 ACM Symposium on SDN Research, San Jose, CA, USA, 3–4 April 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 36–48.
30. Zachos, G.; Essop, I.; Mantas, G.; Porfyrakis, K.; Ribeiro, J.C.; Rodriguez, J. Generating IoT Edge Network Datasets based on the TON_IoT Telemetry Dataset. In Proceedings of the 2021 IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, Virtual Event, 25–27 October 2021.
31. Node-RED. Available online: https://nodered.org/ (accessed on 13 August 2021).
32. ToN_IoT Datasets | IEEE DataPort. Available online: https://ieee-dataport.org/documents/toniot-datasets (accessed on 19 October 2021).
33. What is VMware NSX? *Network Security Virtualization Platform AU*. Available online: https://www.vmware.com/au/products/nsx.html (accessed on 13 August 2021).
34. Stojmenovic, I.; Wen, S. The fog computing paradigm: Scenarios and security issues. In Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, 7–10 September 2014; pp. 1–8.
35. Österlind, F.; Dunkels, A.; Eriksson, J.; Finne, N.; Voigt, T. Cross-Level Sensor Network Simulation with COOJA. In Proceedings of the 2006 31st IEEE Conference on Local Computer Networks, Tampa, FL, USA, 14–16 November 2006; pp. 641–648.
36. Moteiv Corporation Tmote Sky—Ultra Low Power IEEE 802.15.4 Compliant Wireless Sensor Module. Available online: http://www.crew-project.eu/sites/default/files/tmote-sky-datasheet.pdf (accessed on 6 September 2021).
37. International Telecommunications Union—Telecommunication Standardization Sector (ITU-T). Recommendation ITU-T Y.2060: Overview of the Internet of Things. Available online: https://www.itu.int/ITUT/recommendations/rec.aspx?rec=y.2060 (accessed on 19 October 2021).
38. Scikit-Learn. Available online: https://scikit-learn.org/stable/ (accessed on 20 August 2021).
39. Latif, S.; Zou, Z.; Idrees, Z.; Ahmad, J. A Novel Attack Detection Scheme for the Industrial Internet of Things Using a Lightweight Random Neural Network. *IEEE Access* **2020**, *8*, 89337–89350. [CrossRef]
40. Huma, Z.E.; Latif, S.; Ahmad, J.; Idrees, Z.; Ibrar, A.; Zou, Z.; Alqahtani, F.; Baothman, F. A Hybrid Deep Random Neural Network for Cyberattack Detection in the Industrial Internet of Things. *IEEE Access* **2021**, *9*, 55595–55605. [CrossRef]