Journal of
Sensor and Actuator Networks

MDPI

Article

Hybrid Deep Learning-Based Intrusion Detection System for RPL IoT Networks

Yahya Al Sawafi, Abderezak Touzene * and Rachid Hedjam

Department of Computer Science, Sultan Qaboos University, Muscat P.O. Box 36, Oman; yalsawafi@squ.edu.om (Y.A.S.); rachid.hedjam@squ.edu.om (R.H.)
* Correspondence: touzene@squ.edu.om

Abstract: Internet of things (IoT) has become an emerging technology transforming everyday physical objects to be smarter by using underlying technologies such as sensor networks. The routing protocol for low-power and lossy networks (RPL) is considered one of the promising protocols designed for the IoT networks. However, due to the constrained nature of the IoT devices in terms of memory, processing power, and network capabilities, they are exposed to many security attacks. Unfortunately, the existing Intrusion Detection System (IDS) approaches using machine learning that have been proposed to detect and mitigate security attacks in internet networks are not suitable for analyzing IoT traffics. This paper proposed an IDS system using the hybridization of supervised and semi-supervised deep learning for network traffic classification for known and unknown abnormal behaviors in the IoT environment. In addition, we have developed a new IoT specialized dataset named IoTR-DS, using the RPL protocol. IoTR-DS is used as a use case to classify three known security attacks (DIS, Rank, and Wormhole). The proposed Hybrid DL-Based IDS is evaluated and compared to some existing ones, and the results are promising. The evaluation results show an accuracy detection rate of 98% and 92% in f1-score for multi-class attacks when using pre-trained attacks (known traffic) and an average accuracy of 95% and 87% in f1-score when predicting untrained attacks for two attack behaviors (unknown traffic).

Keywords: intrusion detection systems; deep learning; machine learning; security; RPL; routing protocols

Citation: Al Sawafi, Y.; Touzene, A.; Hedjam, R. Hybrid Deep Learning-Based Intrusion Detection System for RPL IoT Networks. *J. Sens. Actuator Netw.* **2023**, *12*, 21. https://doi.org/10.3390/jsan12020021

Academic Editor: Mingjun Xiao

Received: 21 February 2023
Revised: 6 March 2023
Accepted: 7 March 2023
Published: 8 March 2023

1. Introduction

The IoT technology has recently been the fuel engine for many smart applications designed to improve life quality, such as smart cities, transportation, healthcare, energy management, agriculture, environmental monitoring, and others [1]. IoT transforms the physical objects from being traditional to being smart by taking advantage of underlying technologies such as sensor networks, embedded devices, pervasive computing, ubiquitous communications protocols, and applications. The majority of IoT systems use resource-constrained (low processing power and storage) devices and a low-power and lossy network (LLN) to communicate between them in an ad hoc way. Therefore, these systems are vulnerable and prone to many cyber-attacks, where traditional security measurements cannot be directly applied [1].

Each protocol layer in the IoT stack is susceptible to various security threats; however, most of the attacks target the network layer, as for traditional networks. These attacks usually affect routing protocols in terms of data flow disruption or network resources exhaustion. Routing protocols play a vital role in IoT network architecture. A routing protocol's primary role is to discover and establish a route from a source node to a destination node and maintain the availability of such routes for subsequent transmissions. Several routing protocols have been used for the Wireless Sensor Networks (WSN) and IoT. However, the Routing Protocol for Low-Power and Lossy networks (RPL) is a very

promising routing protocol because of its ability to provide efficient routing among resource-constrained IoT IP-based devices. RPL was standardized by IETF ROLL (Routing Over Low-Power Lossy Links) in 2012 [2]. However, the RPL protocol is vulnerable to a wide range of attacks that are difficult to detect and mitigate [3]. The attacks can target different network components in which some attacks are used to exhaust the network resources (energy, memory, and storage) to shorten the device's lifetime, which then shortens the network's lifetime. Other attacks target the RPL topology, aiming to disturb the normal data workflow. Moreover, some attacks might target data traffic confidentiality considered eavesdropping attacks [4].

Among the different security controls that are used to mitigate the security risks in IoT environments, Intrusion Detection Systems (IDS) are widely used techniques for detecting suspicious events. The main goal of the IDS is to monitor, analyze, and detect abnormal behaviors (attacks) in the network traffic [5]. There are three general types of IDS, which are signature-based, anomaly-based, and specification-based. For the signature-based approach, which is also known as rule-based, the system stores the signatures of the known attacks in a database and compares them to the network traffic pattern [6]. Whenever the pattern matches the existing attack signature, the system triggers an alert. Although this approach is considered very fast in detecting known attacks, it is not useful in identifying new attacks. In the anomaly-based detection, the system defines the network's legitimate activities and then compares them with anomalous activities (unknown attacks). The statistical methods and machine learning techniques are the most used in this type of IDS. However, many of the systems that apply in anomaly-based systems suffer from a higher rate of false-positives (regular traffic classified as malicious) if they are not designed and trained carefully. For the specification-based approaches, the expected legitimate behaviors of the network components such as the routing protocols and the nodes are defined such that any variation in this behavior is considered as an attack. Although this ensures lower false-positive rates, some security experts need to determine the elements' specifications, which is more time-consuming [6].

IDS systems have been studied and developed using machine learning (ML) techniques to detect anomaly network behaviors. However, with the rapid increase in connected network devices and traffics that produce a large scale of data, identifying various types of attacks becomes more sophisticated and challenging by using the simple or shallow machine learning approaches [7]. Deep learning (DL) is the advanced branch of machine learning that has shown its success in classification and dimensionality reduction tasks [8,9]. In deep learning, features can be learned from a large number of training samples and automatically reduce the network traffic complexity to find the correlations among them [10]. This makes it more powerful in detecting complex attack patterns and zero-day attacks. However, DL-based IDS have not been studied enough in the IoT network context and, more specifically, in RPL-based networks.

Moreover, to accurately train and evaluate the ML-DL-based IDS, a relevant dataset is required. The majority of the existing methods use existing benchmark datasets such as KDD99 [11], NSL-KDD [12], and UNSW-NB15 [13]. However, these datasets are obsolete, or they have been created using computer system traffics that are not suitable for the IoT IDS system. On the other hand, the existing datasets that are designed for IoT IDS systems, such as the WSN-DS [14] dataset, do not mimic the reality in collecting the network traffics. They are based on sniffing or monitoring mechanisms that need to be distributed across the IoT network, which is difficult to apply in practice. Moreover, the Bot-IoT [8] dataset, which is meant for IoT networks, does not consider common types of ad hoc networks where the devices form a network using routing protocols to communicate between each other to route the data to a central location.

In this paper, we propose a new dataset named IoTR-DS based on RPL (which is considered the de facto routing protocol for IoT networks). IoTR-DS is created by simulating three common attack (DIS, Rank, and Wormhole) traffics beside the normal traffics. Unlike the existing datasets, the data collection does not involve traffic sniffing or monitoring but

utilizes the same data packets sent by the nodes to the root by embedding the necessary information on it. The idea is to shift the IDS control to the powerful node and, at the same time, not to use the network sniffing technique, which is not practical and costly.

In addition, the paper proposes a hybrid DL-based IDS based on the hybridization of supervised Deep Artificial Neural Network (DANN) and semi-supervised Deep Autoencoder (DAE) models to classify attacks using the IoTR-DS dataset. The supervised DANN model is trained using labeled attacks, and it is used to identify known attacks. The semi-supervised DAE model is trained using normal traffic samples only and is used to predict the traffic that DANN was not trained for. The idea is to compare the average reconstruction error during the normal traffic-training phase with the predicted traffics. Another goal of the proposed approach is to validate the IoTR-DS dataset in terms of whether it contains enough features to efficiently classify different attacks. In summary, the contribution of this paper is twofold:

- Develop a new specialized dataset named IoTR-DS (https://github.com/alsawafi/IoT-DL-IDS (accessed on 20 October 2020)) by simulating three types of RPL attacks along with normal traffics and characterize attack features that will be used as the primary inputs to the IDS.
- Propose a new DL-based IDS using the hybridization of supervised DANN and semi-supervised DAE and evaluate it on the IoTR-DS dataset.

The remainder of this paper is organized as follows: Section 2 presents some background about RPL and attacks and reviews related works on machine–deep learning-based IDS and datasets. Section 3 offers the IoTR-DS dataset creation and RPL attacks modeling. The design details of the proposed hybrid DL-IDS using the IoTR-DS dataset is discussed in Section 4. The evaluation experiments' methodology and the results analysis conducted for the proposed protocol are shown and discussed in Section 5. Finally, Section 6 concludes the work and outlines future research directions.

A list of all of the abbreviations cited in this paper is summarized in the following Table 1:

Table 1. List of abbreviations.

Abbreviation	Description
6LoWPAN	IPv6 over Low-power WPAN
D2D	Device to Device
DAE	Deep Autoencoder
DAG	Directed Acyclic Graph
DANN	Deep Artificial Neural Network
DAO	Destination Advertisement Object
DIO	DODAG Information Object
DIS	DODAG Information Solicitation
DL	Deep learning
DODAG	Destination-Oriented Directed Acyclic Graph
DoS	Denial of Service
ETX	Expected Transmission Count
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoT	Internet of Things

2. Literature Review

This section first introduces the RPL protocol components and operation. Then, it describes the common attacks that target the RPL. Finally, it reviews some related works on ML-DL IDS and datasets used to detect attacks related to the IoT network.

2.1. Routing Protocol for Low-Power and Lossy Networks (RPL)

RPL is an IPv6 routing protocol that has been used in IoT and standardized by IETF in 2012. It is mainly designed to operate on energy-constrained devices that use low power, low-cost communication technologies, and less memory. Figure 1 illustrates the main components and terminology used in RPL. The RPL protocol arranges the nodes (sensors) into a tree topology called a destination-oriented directed acyclic graph (DODAG). The tree root (sometimes named a border) is a non-constrained node that initiates and orchestrates the tree construction. All nodes on each DODAG are rooted at a single root. A specific objective function is used to determine how a DODAG is structured based on specific application needs. It uses some metrics and constraints to compute some parameters such as the rank of the node (based on the distance to the root) and the preferred parent of the node. For example, the hop count metric can be used to compute the rank.

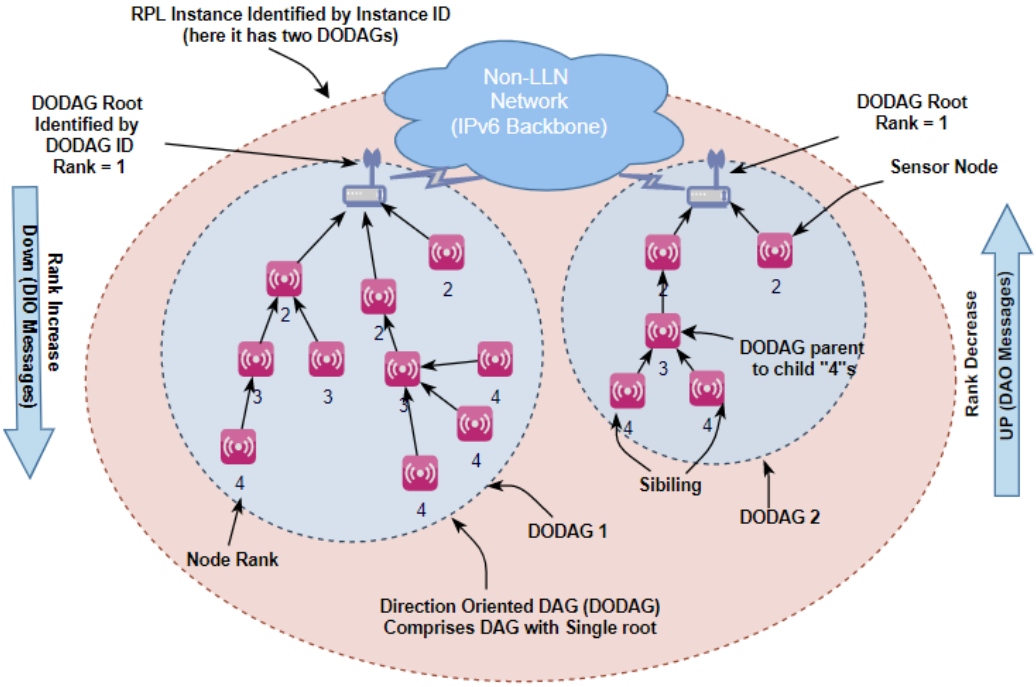

Figure 1. RPL Components and Terminology.

In order to exchange messages between the nodes themselves and with the root (in an ad hoc manner), RPL uses some ICMPv6-based control messages. The main control message used is the DODAG Information Object (DIO). It is used by the root to initiate and maintain a DODAG tree and by other nodes to join this tree and to keep track of its RANK (position in relation to the DODAG root). For DODAG consistency purposes, each node will inspect the next received DIOs for a different DODAG version or different RANK number than the previous ones. To search for new DODAG or maintain an existing one, if the node does not receive any DIO message within a specific time, it will multicast a DODAG Information Solicitation (DIS) message. This will solicit a DODAG DIO message from an RPL node. RPL also supports downward traffic routes by using a Destination Advertisement Object (DAO) control message, which is a unicast message that is used to propagate destination information upward along the DODAG.

2.2. RPL Attacks

In this section, we describe three different attacks against the RPL protocol that are used in this study. We also describe the relevant features that might help in identifying such an attack and could be used for the dataset.

Flooding (DIS) Attack: In flooding attacks, the attacker nodes usually target the availability of the network by sending a large amount of traffic, which exhausts the resources of the neighboring nodes and makes them unavailable. It is a Denial of Service attack (DoS). In RPL, the malicious node may send a large number of DIS control messages to flood the neighboring nodes to solicit DIO messages. As a response, the nodes receiving DIS messages generate more traffic to the network by broadcasting DIO messages [15]. The important features that can be observed to identify such an attack are: the number of DIS messages received; the number of DIO messages sent by each node; the end-to-end delay; and the average packet delivery ratio.

Rank attacks: The node rank plays an important role in RPL. It is used to construct the optimal network topology and prevent the formation of loops in the network. Attacking the rank by deliberately changing its value can lead to two different types of known attacks: increased rank attacks and decreased rank attacks. The malicious node advertises a higher rank value in the increased rank attack than it is supposed to have. In this case, its new preferred parent might be its previous child in the prior sub-DODAG. This will form a routing loop between neighbor nodes. Although a loop avoidance mechanism is designed to fix such loops, it requires many DIOs to be exchanged between the nodes, resulting in exhausting node resources. In the decrease rank attack, the malicious node advertises a lower rank value, aiming to attract other nodes to connect to it. The rank attacks lead to the selection of non-optimized routes and may lead to poor network performance [3]. The important features that can be observed to identify such attacks are: the number of DIOs sent; the frequency of parent and RANK change; the end-to-end delay; and the average packet delivery ratio.

Wormhole attack: The wormhole attack requires at least two nodes to create a dedicated communication tunnel between them. They usually have an alternative network interface and use either wired or wireless links to make the out-of-band connection. Once the tunnel is made, one attacker can replay all the messages received from the normal path to the second attacker node using the dedicated link [16]. The attacker nodes might be far away from each other since they might use better communication coverage ranges or wire connections. In RPL, each attacker node will first replay the DIOs messages received from other nodes to the second attacker using their second interface. The receiving attacker node will process this DIO (on its second interface) and add the sender (attacker node) as its preferred parent. Now, the route is set up between the two nodes. All normal data traffic received by the first attacker on its first interface will be replayed on the second interface.

Many consequences can result from the wormhole attack depending on the intention of carrying out such an attack. It can be used to disturb the normal operation of the routing by selecting non-optimal paths. It can also be used to eavesdrop on all traffic passing through the tunnel (confidentiality attack). To make it more effective, sometimes, this attack is combined with other attacks such as a decreased rank attack, which attracts more neighbors to connect to this malicious node. In this case, the aim is to route more traffic using a wormhole tunnel. The important features that can be extracted during this wormhole attack are: the number of DIOs sent by each node; the end-to-end delay; and the average packet delivery ratio. Although the same features might be present in different attack types, their values will differ from one attack to another, in which the IDS system is supposed to capture the difference ranges between them.

2.3. Related Works

Several IDS systems based on deep learning have been proposed, aiming to secure IoT systems. In [17], the authors introduce a DL-based IDS that uses the spider monkey optimization (SMO) algorithm to extract the most relevant features from the dataset and a stacked-deep polynomial network (SDPN) to classify the data as normal or abnormal. The model is evaluated using the NSL-KDD dataset and shows a high detection rate for different attack categories (DoS, U2R, R2L, and probe). The authors in [18] implemented a DL-based IDS for the IoT at the fog level rather than at a centralized cloud. They demonstrate

that distributed attack detection at the fog level is scalable, and DL models outperform shallow ML models when used to detect attacks in the NSL-KDD dataset. Similarly, at the fog level, the authors in [19] proposed an IDS using a deep multi-layered recurrent neural network. The system is composed of a cascaded filtering stage where each filter is tuned to different hyperparameters for enhancing the detection of specific attack types. The model is evaluated using the NSL-KDD dataset to detect particular types of attacks. A cloud-based distributed deep learning framework is proposed by [20] to identify and mitigate Botnet, DDoS, and phishing attacks. The framework consists of two components that work cooperatively. These two components are the Long–Short-Term Memory (LSTM) network model at the back-end for detecting Botnet attacks. A Distributed Convolutional Neural Network (DCNN) model hosted in the IoT devices is used to detect DDoS and phishing attacks. In [21], the authors propose using a Deep Auto-Encoder (DAE) and Deep Feedforward Neural Network (DFFNN) to detect anomaly behaviors in Internet Industrial Control Systems (IICSs). The (DAE) algorithm is used to learn normal network behaviors and tweak the optimal parameters (i.e., weights and biases), which then give the DFFNN a better tuning of the parameters and classify normal and abnormal network behaviors.

In more related works that use ML-based IDS in detecting attacks to the RPL, the author in [22] introduces an IDS to detect wormhole attacks in the RPL using three approaches based on ML, namely, the K-means-based approach, a decision tree (DT), and a hybrid approach that combines both methods. The K-mean algorithm is used to cluster routers into groups of safe zones from which a router can communicate with other nodes in the same zone. If the router tries to add new neighbors outside the safe zone, this is considered a wormhole attack. The DT is used to learn the safe distance between any two neighboring routers, where any attempt to communicate more than this distance will consider victims of wormhole attacks. The hybrid approach is used to enhance accuracy by filtering out some of the false-positives. This approach is only limited to one type of attack: the wormhole attack; it also uses additional control messages to send the mapping requests to other nodes. The router nodes' locations are required as input data to the IDS, which is considered impractical in more real implementation. In [23], the author proposed a hybrid IDS framework based on specification-based and anomaly intrusion detection models for detecting selective-forwarding and sinkhole attacks in an RPL-based network. The specification-based intrusion detection uses the agents located in the router nodes to monitor the behavior and send the results to the root. The anomaly-based intrusion detection works as a global detection approach. It is located in the root, and it uses an unsupervised optimum-path forest algorithm to analyze incoming data and detect anomalies. The hybrid method can achieve a reasonable true-positive and false-positive rate for detecting both attacks. This approach considers separating the network nodes into router and leaf nodes. The local IDS agents are only located on the router nodes, and they do not generate data. However, this is not the case in the usual IoT RPL-based network, where the node generates and routes the data simultaneously. In addition, the approach does not consider the energy consumption of those constrained nodes that have agents that might be high, and without this important overhead, it is difficult to assess this method.

In practice, most recent ML- and DL-based IDS approaches use existing benchmarking datasets to evaluate their works. The KDD99 [11] dataset is considered the most popular one released in 1999. It consists of 4.9 million labeled samples of regular traffic and 22 attack types that are categorized into four categories, namely, probing (probe) attacks, root to local (R2L) attacks, user to root (U2R) attacks, and denial of service (DoS) attacks. Another common dataset is NSL-KDD [12], an improved version of KDD99 that eliminates the redundant records in both the training and testing sets while keeping a reasonable number of records. The UNSW-NB15 [13] dataset was introduced to reflect real network traffic and modern low-footprint attacks. The dataset contains a total number of 2.5 million records, 49 features, and 9 types of modern attacks. A more recent WSN-DS dataset [14] was developed for a Wireless Sensor Network (WSN). It consists of 23 features and 4 labeled attacks (Blackhole, Grayhole, Flooding, and Scheduling). A more recent Bot-IoT dataset [8]

was developed for both normal IoT-related and other network traffic, along with different types of attack traffic generally used by botnets. It contains of a total number of 72 million records, 35 features, and 3 attack categories (probing, DDOS, and information theft).

However, the KDD99, NSL-KDD, and UNSW-NB15 datasets are not sufficient to be used as a benchmark for IoT-IDS. The network topologies used to create these datasets are based on client–server wired (Ethernet) communication, which is different from the one used in IoT, where wireless LLN networks are mostly used. Hence, the data traffic types, protocols, and attack types are different in those datasets compared to the IoT case. For example, in IoT networks that use RPL routing over 6LoWPAN, specific control messages (DIO, DIS, DAO, and DAO-ACK) are responsible for network creation and management. Therefore, they are sensitive to different types of attacks. For the WSN-DS dataset, although it was designed for WSN (IoT type network), it requires a set of monitoring nodes to watch the neighbors and report them to the base station. This process consumes more energy and is difficult to manage and apply in reality. With the Bot-IoT dataset, the IoT services are simulated in a direct communication fashion, where the devices are connected directly with the server. However, this does not replicate other scenarios where the IoT devices form a sort of ad hoc network that allows them to communicate with each other and route the data to a center using a routing protocol. In addition, the majority of normal and attack traffic in the dataset are related to traditional computer network systems, and very little traffic is associated with the IoT services.

Therefore, our proposed IoTR-DS dataset is more specialized for IoT-type networks where the RPL protocol is used over the 6LoWPAN link layer. It adds some features that are further related to IoT normal and attack traffic that are not considered in the other dataset. In addition, no monitoring or sniffing tools are required to collect the nodes' data. The important IDS parameters are carried on the data packet sent by the nodes to the root, without any extra overhead. A full description of the IoTR-DS dataset is given in the next section.

3. IoTR-DS Dataset Creation and Attacks Modeling

In this section, we will describe the proposed IoTR-DS dataset attributes (features) and the modeling of the envisaged attacks that produce traffic sample subsets.

3.1. IoTR-DS Attributes

As mentioned, instead of using a monitoring and sniffing tool to capture the nodes' traffics, the IoTR-DS is created by utilizing the sensing data packets sent by all nodes to the root. Some of the routing attributes are appended to the UDP (data) packet when they reach the network layer, such as numSentDIO, numSentDIS, numRankChange, and others (see Table 2). The root will also append some attributes of information related to the UDP (data) packet, such as recTime and numUdpRec. According to the literature, these attributes are useful and analyzed by the IDS system to detect abnormal behaviors. Their values might change due to normal network behaviors such as nodes joining the tree, changing their rank or parent, etc. On the other hand, the change can be due to anomaly behaviors in the network, such as attacks. Moreover, a specific attack can make changes to one or more parameters at a certain level. The IDS system's job is to differentiate between normal and abnormal traffic and classify the attack types.

The packets sent by the nodes are stored in the root as log files, which represent the raw datasets. Table 1 lists the dataset IoTR-DS attributes, where the last one represents the traffic label, whether it is normal traffic or an attack. When the malicious nodes attack, the assumption is that the whole network will be affected during the attack time. Therefore, all traffics are labeled with an attack number during the attack time. The number zero is reserved for labeling normal traffic (i.e., when there are no attacks).

Table 2. IoTR-DS Dataset Attributes.

Number	Feature/Abbreviation	Description
1	No	Packet sequence number
2	recTime	Packet receiving time
3	sendTime	Packet sending time
4	sendrIP	Source IP address
5	nodeID	Node ID
6	nextHopAddr	Node preferred parent
7	Delay	Time received–time sent
8	numSentDIO	Count of DIO sent messages
9	numSentDIS	Count of DIS sent messages
10	numPrefParentChange	Count of preferred parent changes at sending time
11	numRankChange	Count of node rank changes at sending time
12	residualEnergy	Node residual energy at sending time
13	numUdpSent	Count of UDP (data) packets sent at sending time
14	numUdpRec	Count of UDP (data) packets received at receiving time (calculated)
15	PDR	Packet delivery ratio (numUdpRec/numUdpSent) at receiving time (calculated)
16	label	Multi-class labels for different attacks and normal traffic

3.2. Attack Implementation

For the evaluation purpose, four subsets are created using four different scenarios in which the traffic is labeled according to an individual scenario. The first dataset represents the normal traffic behavior when there is no attack within the network. The other three datasets are the results of simulating three types of attacks, which are DIS, rank, and wormhole attacks. In all scenarios, the network's size consists of 100 nodes, whereas the number of malicious (attackers) nodes varies in a different attack scenario. The following gives more details about these attack implementation scenarios.

Algorithm 1 shows the pseudocode for implementing a flood attack (DIS attack in RPL). At a preset time interval, three selected nodes at different locations will periodically (every 0.2 s in our implementation) broadcast DIS messages. This process is repeated with another three malicious nodes at a different place and at different time intervals.

Algorithm 1 DIS Attack

```
Begin
If Malicious node and Start Attack < time < End Attack then
Create DIS, Send DIS to all reachable nodes every t time
If a node receiving DIS then
Trickle reset DIO timer, Send DIO to all reachable nodes on next interval
End
```

The pseudocode in Algorithm 2 shows the rank attack implementation. At a preset point of a time interval, three malicious nodes will initially decrease their rank by two and broadcast DIO messages, including the new ranks to their neighbors. The nodes receiving this DIO will see a better-preferred parent candidate than it currently has, and, thus, it will select it as its preferred parents, recalculate the rank, and broadcast the DIO with the new parameters. The nodes receiving this new DIO might also consider changing their rank and preferred parent. This process is repeated again with another three malicious nodes at a different location and at different time intervals.

Algorithm 2 Rank Attack

```
Begin
If Malicious node and Start Attack < time < End Attack then
Rank—(one time), Trickle reset DIO timer, Send DIO next interval
If a node receiving DIO and Sender Rank less than Preferred_Parent (default route) Rank then
Recalculate Preferred_Parent and this Node Rank
Trickle reset DIO timer, Send DIO to all reachable nodes on next interval
End
```

In a wormhole attack, the malicious node has two network interfaces. On the 802.15.4 interface, the node makes communication to the normal RPL tree, whereas it uses the 802.11 (WiFi) interface to create a tunnel with another malicious node. Although the two malicious nodes are far apart, they are considered neighbors to other nodes. Algorithm 3 shows the pseudocode for implementing this attack. After the tree creation, and at a particular time, one malicious node's multicast DIO message on its interface 802.11 (wider coverage) will be picked up by another malicious node, which has a similar interface. The second malicious node will connect to the sender (create a tunnel) and broadcast the DIO message on 802.15.4, advertising a better link. The node receiving the DIO from this malicious node will probably change its preferred parent to be the sender.

Algorithm 3 Wormhole Attack

```
Begin
If Malicious node and Start Attack < time < End Attack then
Replay and Multicast DIO on 802.11 interface every t time
If Malicious node receiving DIO on 802.11 interface then
Recalculate Preferred_Parent (to be the Malicious node) and this Node Rank
Trickle reset DIO timer, Send DIO (with new rank) on 802.11 to all reachable nodes on
next interval
If a node receiving DIO and Sender Rank less than Preferred_Parent (default route) Rank then
Recalculate Preferred_Parent and this Node Rank
Trickle reset DIO timer, Send DIO to all reachable nodes on next interval
End
```

The RPL protocol and attacks are implemented using the OMNeT++ simulation tool. OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators [24]. It is open-source and can run on top of different operating systems such as Linux, Windows, and MAC. It has the capability of implementing and simulating RPL and other routing protocols at a larger scale. We applied the attack algorithms mentioned earlier. Table 3 shows the configuration parameters for the implementation of attack scenarios.

Table 3. Simulation Parameters For Normal and Attack Traffic Scenarios.

Parameters	Value(s)
Area	500 m × 500 m
Simulation Time	5000 s
Number of Nodes	100
Attacker Second Physical Layer	IEEE 802.11 (WiFi)
WSN Physical Layer	IEEE 802.15.4
Wormhole Attacker Radio Range	250 m
WSN Radio Range	60 m
UDPApp (Packet Generation)	Starts at 10 s and ends at 5000 s
UDPApp Traffic Generation Rate	Every 2 s, 3 s, 5 s
Number of Malicious Nodes	6

Table 4 describes the five sub-datasets' results from the simulation scenarios. For the normal dataset, it contains only normal traffic samples without any attack behaviors. The DIS, Rank, and Wormhole datasets contain a mixture of both normal and abnormal (attack) traffic samples. The combined dataset represents the concatenation of the four datasets in one larger dataset. Notice that the number of samples varies from one dataset to another to give some samples diversity.

Table 4. Number of Samples in IoTR Subsets.

Dataset	Normal Traffic	Attack	Total
Normal	122,594	0	122,594
DIS	60,422	7838	68,260
Rank	55,851	16,581	72,432
Wormhole	98,641	18,805	117,446
Combined	337,508	43,224	380,732

4. Proposed Hybrid Deep Learning-Based IDS (DL-IDS)

The proposed IDS model applies semi-supervised and supervised deep-learning models to detect normal and abnormal (attacks) behaviors in the IoT network. The concept is to use the semi-supervised DAE to verify whether a given traffic behavior has been seen before or not by comparing its reconstructing error with known normal and attack reconstruction errors. If the reconstruction error is within predefined boundaries, it will pass it to the supervised DANN, which will classify the traffic more accurately. Otherwise, the traffic is considered a new attack type, and, therefore, a new class and reconstruction error is added to the system. The proposed model's overall architecture is presented in Figure 2, and the following subsection describes it in detail.

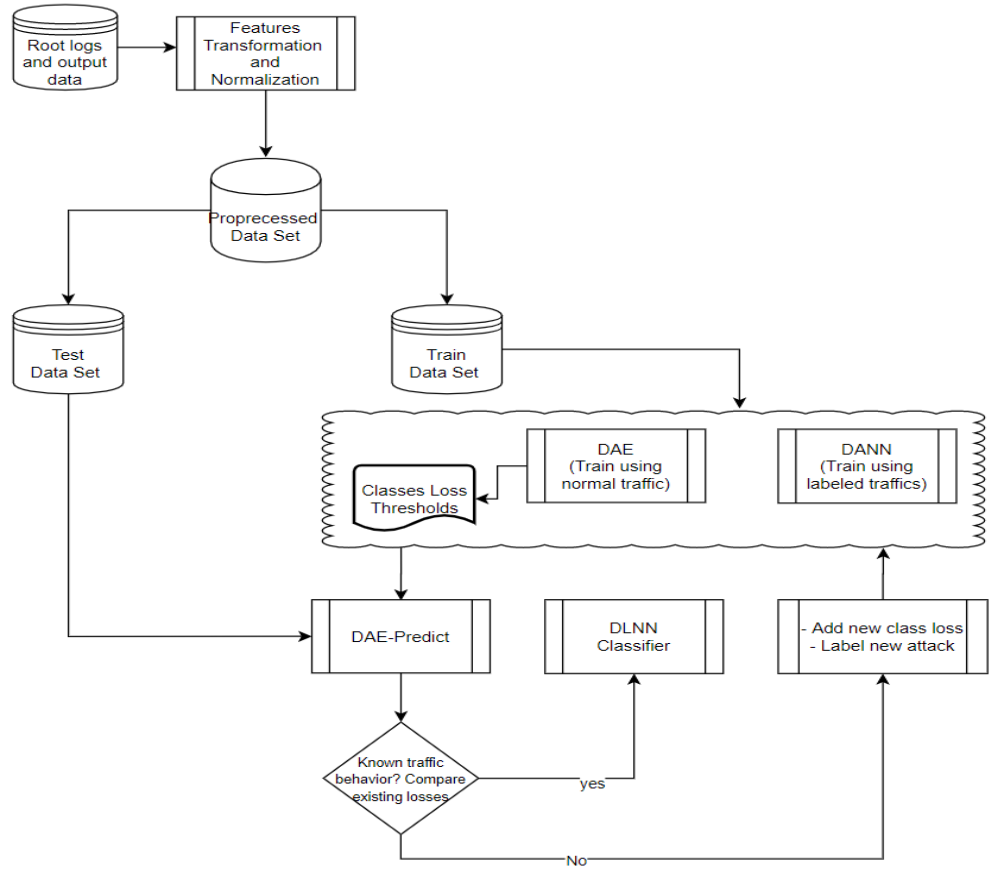

Figure 2. Hybrid DL-IDS Architecture.

4.1. Data Preprocessing

The log files created by the root, which contain data collected from all network nodes, are considered raw data and cannot be used directly as inputs to the DL-IDS model. Therefore, the first step is to preprocess those files. Hence, Python with supported Pandas [25] and Numpy [26] libraries are used to process the raw datasets. Some of the features, such as the DIO packet count, DIS packet count, rank changes count, and others, cannot be used because their values are accumulated each time. Therefore, the first stage is to calculate their values at the sending time by finding the difference between the previous value and the current value at the receiving time. The average packet delivery ratio (PDR) and delay features are also calculated at this stage, as shown in Equations (1) and (2), respectively.

$$PDR = \frac{number\ of\ packets\ received}{number\ of\ packets\ sent} \tag{1}$$

$$Delay = (time\ packet\ received_i - time\ packet\ sent_i) \tag{2}$$

where i is node's packet sequence number.

Subsequently, since the individual sample traffic might not give enough information to the DL model to process, it is good to group some samples to compute the feature averages. Therefore, the traffic instances are grouped according to the receiving time window (in seconds), and the average feature value is taken at this window time. This will also reduce the total number of instances within the dataset when fed to the DL model, which will reduce the learning time. In our implementation, we found that taking the average of the samples every 2 s gives the best result.

After that, we apply normalization techniques to the data features to be on a similar scale to fit the DL-IDS model; Equation (3) is applied to each column feature X to produce a normalized value between 0 and 1. Now, the dataset is ready to feed the DL model.

$$Xnorm = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{3}$$

4.2. Hybridization of DAE and DANN Models (DAE-DANN)

This section describes the IDS training and classification phase using the DAE-DANN approach. Algorithm 4 shows the main steps of the approach. Let us first define some terms: let $\mathbb{R}$ be a set of real numbers and $\mathbf{X} \in \mathbb{R}^{N \times d}$ be a matrix of N training samples (network traffics) of d features each; therefore, $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ stands for the ith training traffic samples, and $\mathbf{y} \in \{y_0, y_1, \ldots, y_i\}^{N \times 1}$ is a vector of N multi-labels of the training samples. $\mathbf{Z} \in \mathbb{R}^{M \times d}$: a matrix of M testing samples (network traffics) of d features each; therefore, $\mathbf{z}_i = (z_{i1}, z_{i2}, \ldots, z_{id})$ stands for the ith traffic test samples. $\boldsymbol{l} \in \{l_0, l_1, \ldots, l_i\}^{M \times 1}$ is a vector of M multi-labels of the testing samples.

In the first phase, the DAE model is trained using normal traffic samples. During the training process, the model will reconstruct the initial input by minimizing the mean squared error between the input and output. The average loss value (reconstructed mean square error) result of the training process will be stored as a threshold named $normalLoss_{threshold}$. During the test phase, the model calculates the average reconstruction error for a given test sample z_i using the function shown in Equation (4).

This function computes the difference between the average sum of the original input features ($z_{i_{test}}$) of a given sample z_i and the reconstructed inputs ($z_{i_{predict}}$) when applying the DAE to the same sample z_i and using the tuned training parameters' weight and the bias that was used with the normal training samples $\mathbf{X}$. The resulting reconstruction loss value ($z_i loss_{diff}$) will then be compared with the stored normal samples' loss threshold ($normalloss_{threshold}$) boundary and attack ($attack_l thresholds$) boundaries. The ($normalloss_{threshold}$) boundary is defined between 0 and $normalloss_{threshold} \times 2$. The attack

$attack_l thresholds$ boundary is defined between $\frac{attack_l threshold}{2}$ and $attack_l thresholds \times 2$. If $z_i loss_{diff}$ is out of these boundaries, then this traffic is considered a new attack label.

$$z_i loss_{diff} = \frac{\left\| \sum_{j=1}^{d} \left(z_{i_{predict_j}} - z_{i_{test_j}} \right) \right\|^2}{d}, \tag{4}$$

where d is the number of features.

To simplify, let us consider the following example: Suppose that we are starting with training set $\mathbf{X}$ containing normal traffic samples and one known attack labeled (y_0, y_1), where y_0 is the normal traffic label and y_1 is the attack traffic label number 1. Therefore the $normalloss_{threshold}$ and $attack_1 threshold$ are known. Let us assume that we have another unknown attack added to the test sample $\mathbf{Z}$ in the range of z_1.. z_m that occurs at a specific time sequence. These test samples are passed to the DAE, compute the $z_i loss_{diff}$ of each sample i, and are compared to the exciting $normalloss_{threshold}$ and $attack_1 threshold$. If the $z_i loss_{diff}$ value is within the $normalloss_{threshold}$ boundary or range, then this sample can be considered as normal traffic, whereas if $z_i loss_{diff}$ is within the $attack_1 threshold$, then this traffic can be considered as $attack_1$. Therefore, these samples will be passed to the supervised DANN model. However, since this is a new attack type, the $z_i loss_{diff}$ of attack samples should be different and out of the range of the $normalLoss_{threshold}$ and $attack_1 threshold$ boundaries. The average out-of-boundaries $z_i loss_{diff}$ of given samples within a time window, w, is calculated, the resulting value is stored as the new attack threshold ($attack_2 threshold$), and y is updated by adding a new label for the recently detected attack. The DANN will be retrained again based on the updated dataset. DANN was used to classify known behaviors, based on which the model was trained much faster and produced a higher classification accuracy.

Algorithm 4 Training and Classification phases of DAE-DANN

Input: ($\mathbf{X}$, $\boldsymbol{y}$): Training set; $\mathbf{Z}$: test set
Output: l: Test sample labels
Function DAE_Train (X, y):
Train DAE with on (X, y),
where $y = 0$, $0:$ normal traffic label
Compute the average $normalLoss_{threshold}$
Return $normalLoss_{threshold}$
Function DANN_Train (X, y):
Train DANN with on (X, y) to obtain optimal values of weight and bias and reduce loss
Return optimal weights & biases

Function DAE_Classifier (Z):
$z_{i_{predict}} = DAE.predict(z_i)$//Feed the DAE with new traffic instances
$loss_{z_i} = Loss_{Diff}(z_{i_{predict}}, z_i)$//Compute $loss_{diff}$ by Equation (4).
If $loss_{z_i} \in \{Normalloss_{Threshold}, attack_1 threshold, ..., attack_l threshold\}$ boundaris
Use Supervised DANN classifier
Else
Add $loss_{z_i}$ to a list V
Within time window w, compute the $Average\_loss_z$ in V
Update the number of classes (labels): $y = y + 1$
$attack_y threshold = Average\_loss_z$//new class threshold
Update (X, y) with new samples and label
Retrain DANN with updated (X, y)
Return $nattack_y threshold$

Function DANN_Classifier (Z):
Predict the labels l using the DANN classifier
Return l

The following is a description of the architecture and configuration of the DAE and DANN used in the IoT-IDS system.

4.2.1. Deep Autoencoder (DAE)

The AE is a type of ANN that is usually used in unsupervised machine learning. Figure 3 shows the overall architecture of the AE that is implemented in the proposed model. It consists of an input layer of d neurons, which represent the number of features. The input layer is then encoded (compressed) by passing it to a hidden layer l_1 with a size of $d/2$ neurons. Another layer with a size of $l_1/2$ neurons is added, which also represents the compressed data representation of the original input. The AE is then trained to reconstruct the inputs from the compressed layer (bottleneck) by having the hidden decoding layers. All layers are followed by dropout hidden layers (not shown in the figure) to prevent the data's overfitting. The encoder maps the input vector $\boldsymbol{x}_i$ to the hidden representation unit, which represents the latent space of the bottleneck layers, as shown in Equation (5).

$$\boldsymbol{l}_i = f(x_i) = \mathcal{g}(Wx_i + b_i) \tag{5}$$

where $\boldsymbol{W}$ is the weight matrix, $\boldsymbol{b}$ is the bias vector, and $\mathcal{g}$ is the activation function. The well-known activation functions, namely, the Rectified Linear Unit (ReLU) and hyperbolic tangent (tanh), are used interchangeably in the model. Experiments have found that combining these functions produces better results. *ReLU* (Equation (6)) is a non-linear activation function, and it is very suitable in MPL with many hidden layers because it is fast and helps to reduce the error gradient issue and state vanishing [27]. The tanh (Equation (7)) function is more like a sigmoid function but differs in the output range of $(-1, 1)$, where the range sigmoid is $(0, 1)$.

$$\mathcal{g}(\boldsymbol{x}) = ReLU(\boldsymbol{x}) = \max(0, \boldsymbol{x}) \tag{6}$$

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{7}$$

where x is the input to the function.

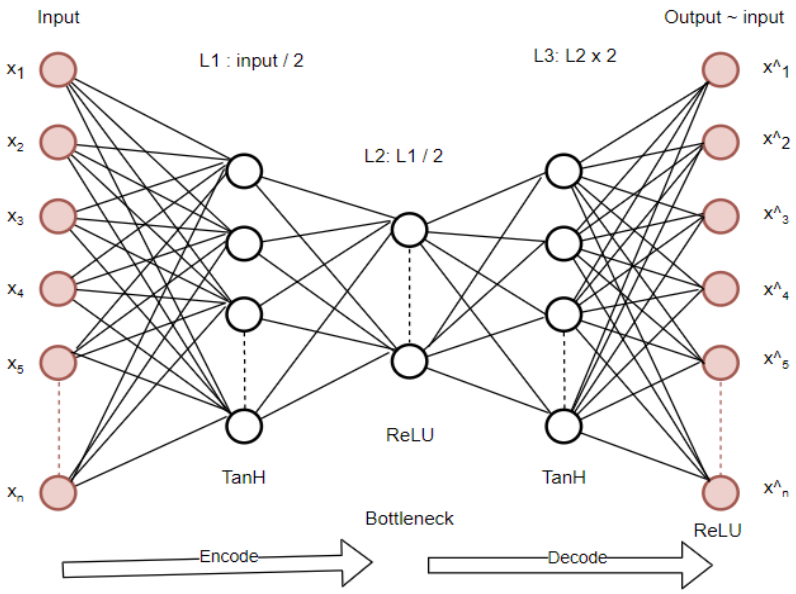

Figure 3. DAE Architecture.

The decoder map l_n reconstructs the x' of the same input size x:

$$lx' = \mathcal{g}'(\boldsymbol{W}'\boldsymbol{l}_i + \boldsymbol{b}_i') \tag{8}$$

By using backpropagation of the error in the training process, the autoencoder tries to minimize the average reconstruction error (or loss) between the input and the reconstructed output (Equation (9))

$$RE(x,\ x') = \frac{1}{m}\left\|\sum_{1}^{m}(\boldsymbol{x}_i - \boldsymbol{x}_i')\right\|^2 \tag{9}$$

where m is the number of samples.

4.2.2. Deep Artificial Neural Network (DANN)

A multilayer perceptron (MLP) model is adopted in the proposed architecture, which represents a deep neural network. The architecture consists of a typical deep neural network, which has an input layer, multiple hidden layers, and an output layer, as presented in Figure 4.

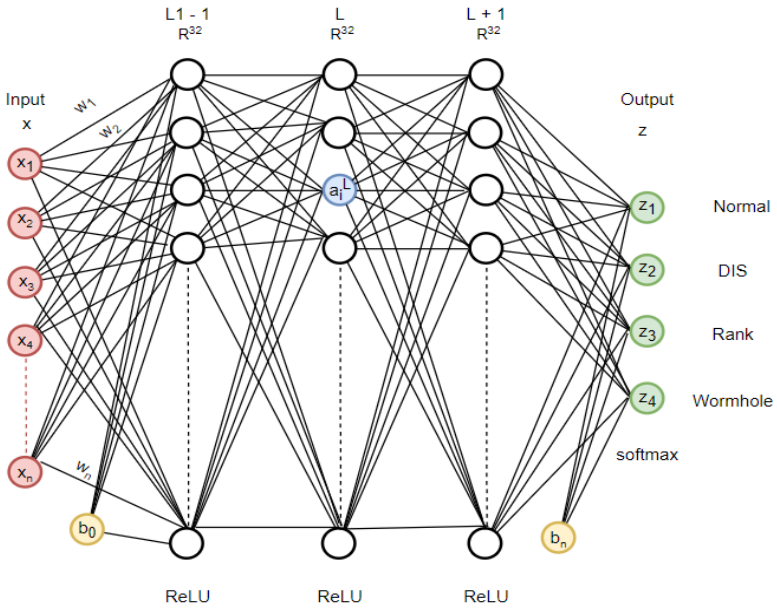

Figure 4. DANN Architecture.

The neurons in each layer are fully connected with other neighbor layer neurons, where the data are transformed from layer to layer in a forward direction. Each neuron in the hidden layer is activated by calculating an output value based on the input data from the previous layer, along with the weight (**w**) values and bias (**b**). Let the $l \in \{1, \ldots, L\}$ index be the hidden layers of the L number. From Figure 4, the output of activating a_i^l is shown in Equation (10).

$$a_i^l = f(\sum_{i=0}^{n} a_i^{l-1} w_i^l + b_i^l) \tag{10}$$

where i is any hidden neuron, n is the number of neurons at the hidden layer, a_i^{l-1} is the i neuron at the previous layer, w_i^l is the weight connection between the a_i^l and a_i^{l-1} neurons, and b_i^l is the bias of layer l. f is the activation function, and $ReLU$ is used for the hidden layers.

Since the network outputs are in multiple classes (labeled attacks), the softmax function is used, which calculates the probability distribution across the classes. The input parameter is converted to be in the form of a one-hot encoded matrix. Equation (9) calculates the probability of class z_i for traffic input $\boldsymbol{x}_i$, where k is the class number of **C** classes, and the

input value of z_i and z_k are calculated using Equation (11). Each class's output is in the range of [0, 1] and adds up to 1 for all the classes.

$$P\left(Z = z_i | \boldsymbol{x}_i\right) = \frac{e^{z_i}}{\sum_{k=1}^{C} e^{z_k}} \tag{11}$$

To calculate the amount of the difference between the predicted class value and the original true class value, the categorical cross-entropy loss function is used. This function is designed for the multi-class classification tasks, where an instance input can belong to one possible class. Equation (12) shows the categorical cross-entropy loss function (CL).

$$CL = PCL\left(z_i, \widehat{z_l}\right) = -\sum_{i=1}^{C} z_i \log \widehat{z_l} \tag{12}$$

where C denotes the number of classes, z_i is the true class value, and $\widehat{z_l}$ is the predicted class value that is calculated using the softmax function (Equation (9)). To minimize the loss value during the training process, the Adam optimization algorithm backpropagates to calculate the gradient (change).

5. Experiment Setup and Evaluation

The experiments were conducted using the TensorFlow platform [28] and Keras [29] as a higher-level framework. The hardware used for the implantation is: Intel(R) Xeon(R) CPU E3-1535M v6@3.10 GHz, RAM 64 GB, 1 TB SSD HD. The same hardware could be used at the root node of the WSN real architecture.

5.1. Experiment Scenarios

Different experiments are designed for various possible scenarios to evaluate the performance of the IoT IDS-DL model using the IoTR-DS dataset. The first experiment was designed to evaluate the model's performance in predicting a single unlabeled attack based on training with the normal traffic. The model was first trained using the normal traffic dataset, and then it was tested with a single attack type using the dataset containing the normal and attack traffic. The second experiment used combined datasets to evaluate detecting two unseen and untrained attacks, among other trained attacks. The third experiment was set to evaluate the supervised binary classification based on having only one known (labeled) attack. Each time, an individual attack dataset was used to train the model and to identify normal and malicious (attack) traffic. The fourth experiment was conducted to evaluate the system in detecting multi-trained (labeled) attacks using multi-class classification. The combined dataset was used to train the model to distinguish between four classes: normal, DIS, rank, and wormhole attacks. Finally, the system was evaluated against four different ML-DL learning models, which are J48 [30], KNN [31], SVM [32], and LSTM [33]. In all of the supervised experiment scenarios, the datasets were split into 70% train, 30% evaluate, and 30% test from the 70% train. For predicting unseen attacks using mainly the semi-supervised model, the normal dataset is the only one used for the training and evaluation (70% train, 30% evaluate), whereas the full attack datasets are used for testing purposes. This was to ensure that the attack behavior was not seen (trained) by the model.

Since the IoT-DL models' performance depends on the optimal hyperparameters configuration, such as the number of hidden layers, the number of neurons per layer, the learning rate, the activation function, and others. Tables 5 and 6 show the near-optimal hyperparameters configuration for DANN and DAE, respectively, based on the best results of running different experiment trials.

Table 5. DANN Hyperparameter Configuration.

Parameter	Binary Classification	Multi-Class Classification
Number of hidden layers	3	3
Number of neurons in the hidden layer	32	32
Dropout	0.5	0.5
Activation function in the hidden layers	ReLU	ReLU
Activation function in the output layer	sigmoid	softmax
Optimizer	adam	adam
Loss function	BinaryCrossentropy	Categorical_crossentropy
Learning rate	0.001	0.001
Number of epochs	1000	1000
Batch Size	120	120
Early_stopping	yes	yes

Table 6. DAE Hyperparameter Configuration.

Parameter	Value
Encoder	[6, 3]
Decoder	[3, 6]
Dropout	0.5
Activation function in hidden layers	Tanh, ReLU
Optimizer	Adamax
Loss function	Mean squared error
Learning rate	0.001
Number of epochs	300
Batch Size	100

5.2. Evaluation Metrics

In order to determine the detection accuracy rate of the proposed model using IoTR-DS, it is evaluated using well-known performance metrics used in machine learning, which are: accuracy, recall, precision, and F1 score. These metrics are calculated using four parameters: True-Positive (T_P), True-Negative (T_N), False-Positive (F_P) and False-Negative (F_N). T_P determines the amount of malicious traffic that is correctly classified as malicious traffic by the model, where T_N indicates the amount of normal traffic that is correctly classified as normal by the model. On the other hand, F_P determines the amount of normal traffic that is incorrectly classified as malicious traffic by the model, where F_N defines the amount of malicious traffic that is incorrectly classified as normal traffic by the model.

The accuracy metric measures the correct prediction ratio by dividing the total number of correct predictions (T_P and T_N) by the sum of all predictions, as illustrated in Equation (13).

$$Accuracy = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} \tag{13}$$

Precision (Equation (14)) estimates the ratio of correctly classified attack traffic to all of the predicted attack traffic, whereas recall (Equation (15)) indicates the ratio of correctly classified attack traffic to the total attack traffic. The f1 score (Equation (16)) can be defined as the weighted average of precision and recall.

$$Precision = \frac{T_P}{T_P + F_P} \tag{14}$$

$$Recall = \frac{T_P}{T_P + F_N} \tag{15}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precsion + Recall} \tag{16}$$

5.3. Evaluation Results

5.3.1. Semi-Supervised Classification Results

In this evaluation experiment, the model was first evaluated in predicting a single unlabeled attack at a time when the model did not see attack behaviors during the training phase. Then, the model was evaluated in predicting two attacks that were not seen by the model during the training phase using a mixture of testing traffic of DIS and rank attacks, besides the normal traffic. The model was trained using the normal traffic samples only where the loss threshold was stored and used to detect the abnormal behaviors.

Figure 5 shows the loss values relevant to the number of epochs over the normal training dataset. The loss value reaches its lower value near 300 epochs, which means that the epochs are sufficient for the model. The average reconstruction error (loss) recorded by the model during the training phase is 0.085, which is stored as a threshold value. As mentioned earlier, when predicting new traffic, the model will reconstruct the error using trained normal samples parameters (weight and bias). If the given traffic is not within the normal traffic range, the reconstruction error should be relatively higher than the threshold. Figures 6–8 show the average reconstruction error when the DIS, rank, and wormhole attack are predicted, respectively. These figures clearly show the difference between the normal loss threshold and the average attack loss values. In addition, they demonstrate that each attack loss value or cluster does not interchange with other attack loss values; therefore, they can separate the attack domain between each other. The higher the distance between the attack loss value and the normal threshold, the higher the detection rate of the attack.

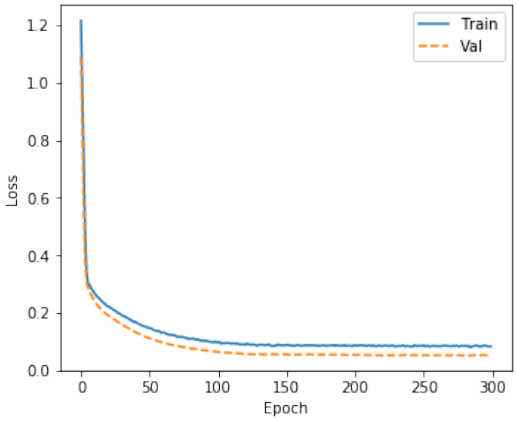

Figure 5. Loss Performance Over a Number Of Epochs Using Normal Traffic (No Attacks).

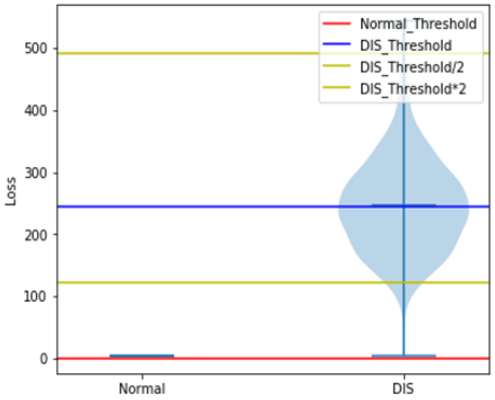

Figure 6. Average Predicted DIS Attack Loss Compared to the Normal Loss Threshold.

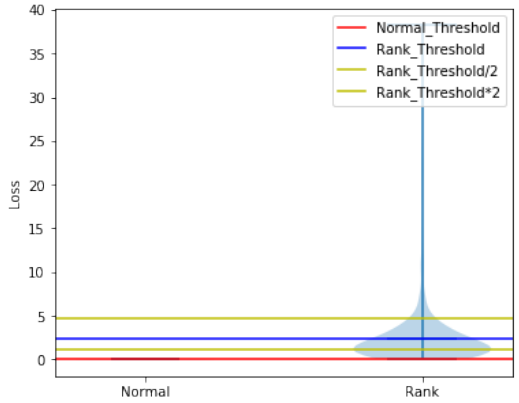

Figure 7. Average Predicted Rank Attack Loss Compared to the Normal Loss Threshold.

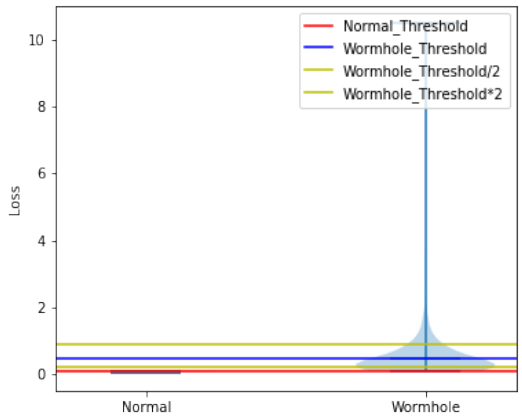

Figure 8. Average Predicted Wormhole Attack Loss Compared to the Normal Loss Threshold.

Table 7 presents the performance metrics of the binary classification using DIS testing instances. It shows that the overall DIS attack detection accuracy is 99%, with a precision, recall, and f1-score of 99%, 98%, and 98%, respectively, using the DIS dataset. With the rank attack classification, the accuracy is 96%, and the precision, recall, and f1-score are 85%, 98%, and 91%, respectively, as shown in Table 8. For the wormhole DAE classification, the overall accuracy detection rate is 92%, with a precision, recall, and f1-score of 70%, 93%, and 80%, respectively, as shown in Table 9. These values are much lower than those of the other two attack cases because the reconstruction error value for the wormhole attack traffic is lower than the other attack errors. In other words, the difference in the feature values changes between the wormhole attack, and there is less normal traffic compared with those change values in other attacks. Finally, by predicting a combination of two untrained attacks (DIS, rank), the overall accuracy of classifying the traffic is 95%. The precision, recall, and f1-score reported in DIS are 99%, 95%, and 97%, respectively, which are much higher than those in the rank attack, which are 93%, 53%, and 68%, respectively, as listed in Table 10. Again, this is due to the reconstruction error value for the rank attack traffic being much lower than that of the DIS attacks, making it more challenging to detect.

Table 7. Performance Metrics in DAE DIS Attack Classification.

	Precision	**Recall**	**F1-Score**	**Accuracy**
Normal	0.99	0.98	0.98	0.99
Attack	0.99	0.98	0.98	

Table 8. Performance Metrics in DAE Rank Attack Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.96	0.98	0.96
Attack	0.85	0.98	0.91	

Table 9. Performance Metrics in DAE Wormhole Attack Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.98	0.92	0.95	0.92
Attack	0.70	0.93	0.80	

Table 10. Performance Metrics in the DAE of Both DIS and Rank Attacks Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.94	0.99	0.97	0.95
DIS	0.99	0.95	0.97	
Rank	0.93	0.53	0.68	
Average	0.95	0.82	0.87	

5.3.2. Supervised Binary Classification

For the binary classification, loss, and Area Under Curve (AUC) performance, there are plots based on the training process. Although the training iterations (number of epochs) was initially set to 1000, the training stopped, and the loss AUC achieved was very good, even much earlier (converged). The dropout and callback techniques were used to speed up the training process and avoid overfitting. Figure 9 shows that the loss and AUC converge to the best performance with 70 epochs in the DIS dataset. For the rank dataset, 140 training epochs were needed to reach a higher performance rate (Figure 10), and 200 training epochs were needed in the wormhole case (Figure 11).

The performance detail results for DIS, rank, and wormhole attacks binary classification are reported in Tables 11–13, respectively. The model classification accuracy rates are 99, 98, and 97% for the DIS, rank, and wormhole attacks, respectively. The other performance metrics (precision, recall, and f1 score) also achieved a higher rate between 91 and 99%.

Table 11. Performance of DNN IDS Binary Classification (DIS Attack).

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.99
DIS Attack	0.98	0.99	0.98	

Table 12. Performance of DNN IDS Binary Classification (Rank Attack).

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.98
Rank Attack	0.96	0.97	0.97	

Table 13. Performance of DNN IDS Binary Classification (Wormhole Attack).

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.98	0.98	0.97
Wormhole Attack	0.91	0.94	0.92	

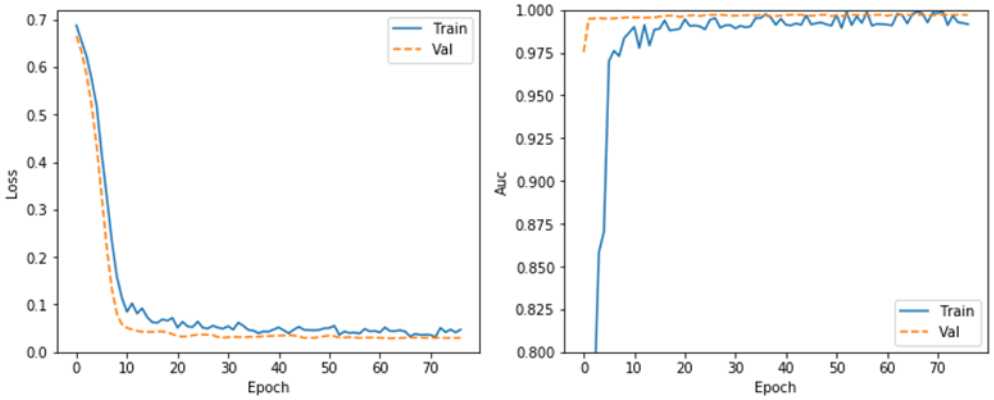

Figure 9. DANN Model Loss and AUC for DIS Attack Binary Classification.

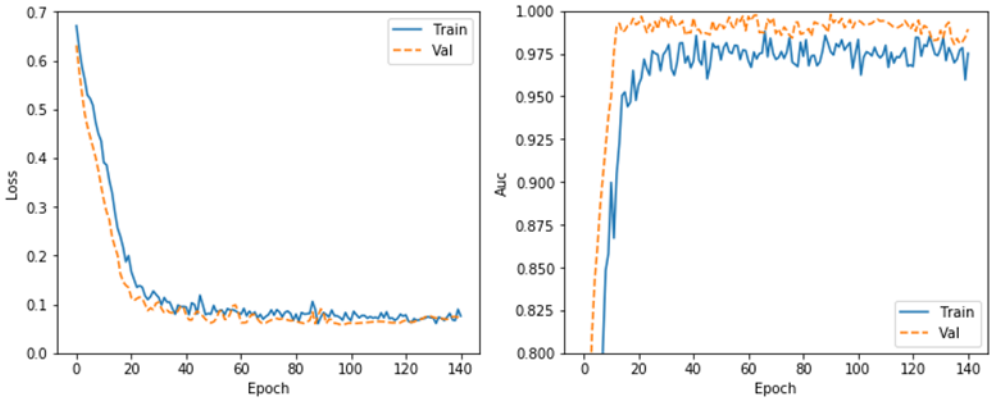

Figure 10. DANN Model Loss and AUC for Rank Attack Binary Classification.

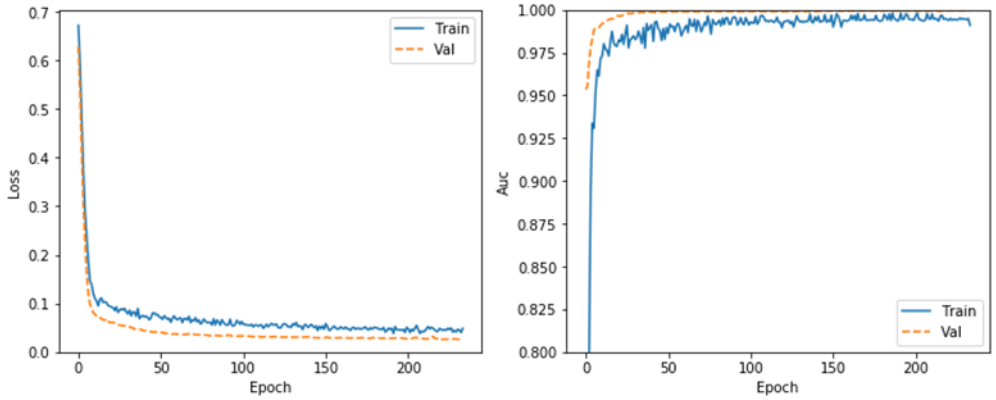

Figure 11. DANN Model Loss and AUC for Wormhole Attack Binary Classification.

5.3.3. Supervised Multi-Class Classification

Since the DANN model's main purpose is to conduct multi-classification, the model is trained using the combined attack datasets to distinguish among four classes: normal, DIS, rank, and wormhole. Different depths of hidden layers (1, 2, 3, and 4 layers) were implemented to find the model's optimal one. Figure 12 shows the loss and AUC using a single DANN hidden layer model. It took almost the full training trials (1000 epochs) to achieve a higher performance. Table 14 presents the performance metrics with a single-layer multi-class classification. The model achieves an overall accuracy of 96%, with a higher precision, recall, and F1-score in detecting the normal class, and the lowest one with the wormhole attack class.

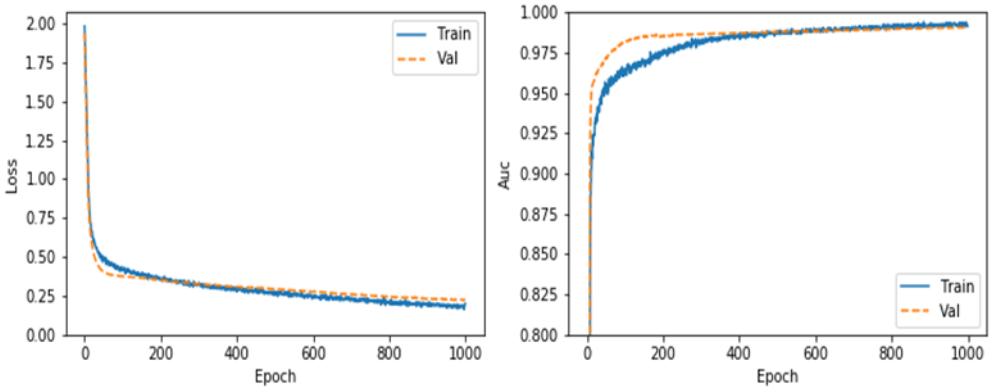

Figure 12. Loss and AUC With One DANN Hidden Layer Used in Multi-Class Classification.

Table 14. Performance Metrics with Single-Layer Multi-Class Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.96
DIS Attack	0.98	0.97	0.98	
Rank Attack	0.86	0.77	0.82	
Wormhole Attack	0.65	0.67	0.66	
Average	0.87	0.85	0.86	

Using two hidden layers, the model achieves the best loss and AUC performance at 500 training epochs, as shown in Figure 13. Table 15 shows that the overall class detection accuracy is 96%. Similarly, as with using the one-layer model, the highest precision, recall, and f1-score are in normal class detection, and the lower performance metrics are in detecting the wormhole attack.

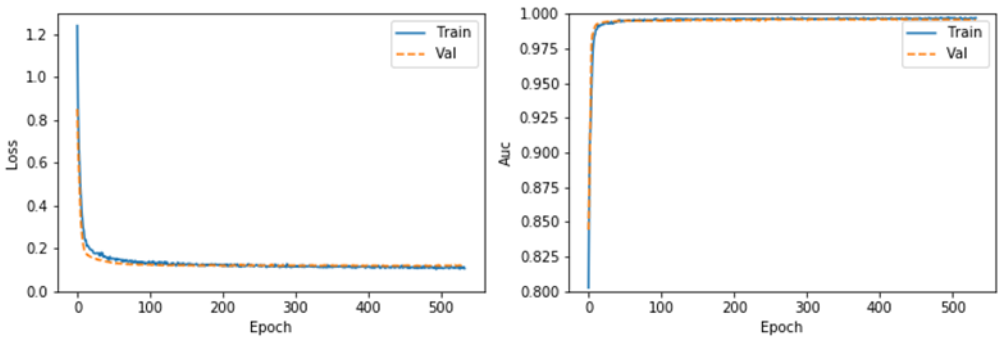

Figure 13. Loss and AUC With Two DANN Hidden Layers Used in Multi-Class Classification.

Table 15. Performance Metrics with Two-Layer Multi-Class Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.96
DIS Attack	0.99	0.97	0.98	
Rank Attack	0.83	0.74	0.78	
Wormhole Attack	0.71	0.74	0.73	
Average	0.88	0.86	0.87	

Figure 14 shows that the loss and AUC reach the optimal performance at 300 epochs only by using three hidden layers. The model also gives an overall 98% classification accuracy, as presented in Table 16. The higher precision, recall, and f1-score performance

percentage are achieved in classifying the normal class (99%). The overall average performance metrics are very good (92%) in classifying the three attacks in addition to the normal traffic.

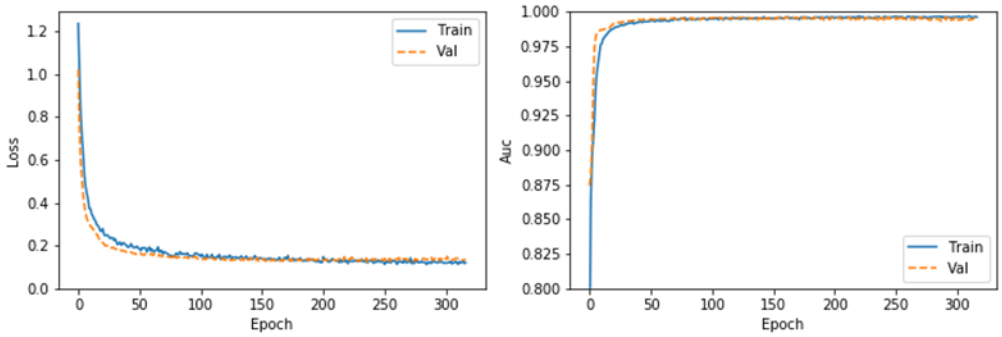

Figure 14. Loss and AUC with Three DANN Hidden Layers Used in Multi-Class Classification.

Table 16. Performance Metrics with Three-Layer Multi-Class Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.98
DIS Attack	0.98	0.99	0.98	
Rank Attack	0.89	0.88	0.89	
Wormhole Attack	0.80	0.81	0.80	
Average	0.92	0.92	0.92	

Adding more hidden layers does not always mean improving model performance. Figure 15 shows that, with four hidden layers, the model requires more than 600 training epochs (which is more than in the three-layers model) to reach the best loss and AUC performance. Table 17 also presents that the overall accuracy detection rate is 97% less than when using three layers. Moreover, it is noticed that the average precision, recall, and f1-score of the four classes are less than those in the three-layers model. By considering these results, it was decided to use three hidden layers in the DNN-IDS multi-class model.

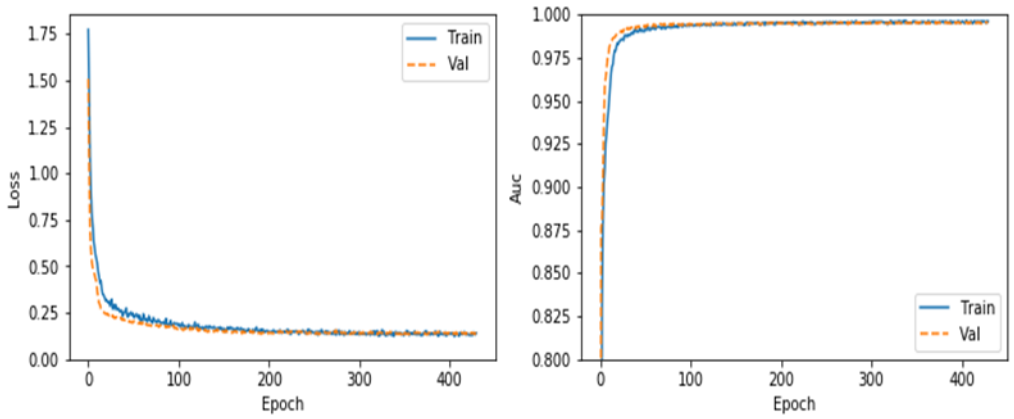

Figure 15. Loss and AUC With Four DNN Hidden Layers Used in Multi-Class Classification.

As mentioned earlier, the DL-IDS using the IoTR-DS dataset is also evaluated against other classical and deep learning models to see how the model performs compared to those models. Since it is difficult to evaluate the hybrid model, we have considered comparing the f1-score and accuracy performance of the supervised part (DANN) against the J48, KNN, SVM, and LSTM models, and the results are presented in Table 18. As shown, the DANN outperforms the other models in terms of both f1-score and accuracy metrics. The

second-ranked detection model is LSTM, one of the deep learning models, which indicates that deep learning models are better than classical machine learning in this type of attack detection. It is also worth mentioning that all the models have difficulty classifying both rank and wormhole attacks because they have common features that are affected during the attacks.

Table 17. Performance Metrics with Four-Layer Multi-Class Classification.

	Precision	Recall	F1-Score	Accuracy
Normal	0.99	0.99	0.99	0.97
DIS Attack	0.98	0.97	0.98	
Rank Attack	0.86	0.81	0.84	
Wormhole Attack	0.71	0.72	0.71	
Average	0.88	0.87	0.88	

Table 18. Performance of Different ML-DL Models.

	DANN		J48		KNN		SVM		LSTM	
	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy
Normal	0.99	0.98	0.98	0.95	0.98	0.94	0.98	0.94	0.98	0.96
DIS Attack	0.98		0.97		0.96		0.96		0.96	
Rank Attack	0.89		0.79		0.78		0.77		0.84	
Wormhole Attack	0.80		0.75		0.67		0.61		0.77	
Average	0.92		0.87		0.84		0.83		0.88	

6. Conclusions and Future Work

This paper proposes a hybrid IDS deep learning-based model for detecting and classifying cyber-attacks for IoT networks based on the RPL routing protocol. Due to the lack of datasets that are developed for IoT traffic, a new dataset named IoTR-DS was created by simulating three types of common RPL attacks, which are DIS, Rank, and Wormhole attacks, in addition to the normal traffic. Unlike the existing techniques, which use sniffing and monitoring tools to collect the data traffic, in our IoTR-DS, the data attributes used for analysis are contained in the data packet and stored in the root where the hybrid DL-IDS is implemented. The hybrid DL-IDS is composed of supervised DANN and semi-supervised DAE, which are designed to predict attacks that are trained and untrained. The goal was to design a scalable model for detecting new attacks where their traffic behavior was not seen before by comparing the traffic reconstruction loss with the normal traffic and the existing attack loss values.

The evaluation results show a detection accuracy rate of 98% for multi-class attacks when using pre-trained attacks (known) and an average of 95% when predicting unseen two-attack behaviors (untrained). This high classification detection rate shows the good performance of the proposed model; it also indicates that the IoTR-DS dataset contains enough features and information to classify such attacks. Other attacks related to the routing and application layer will be included in the dataset and tested by the hybrid model in our future work.

Author Contributions: Writing—original draft preparation, Y.A.S.; review and editing, A.T. and R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Thamilarasu, G.; Chawla, S. Towards deep-learning-driven intrusion detection for the internet of things. *Sensors* **2019**, *19*, 1977. [CrossRef]
2. Winter, T. RPL: IPv6 routing protocol for low-power and lossy networks. *Net. Architect. Serv.* **2012**, *19*, 1–157.
3. Verma, A.; Ranga, V. Security of RPL based 6LoWPAN Networks in the Internet of Things: A Review. *IEEE Sens. J.* **2020**, *20*, 5666–5690. [CrossRef]
4. Mayzaud, A.; Badonnel, R.; Chrisment, I. A Taxonomy of Attacks in RPL-based Internet of Things. *Int. J. Netw. Secur.* **2016**, *18*, 459–473.
5. Choudhary, S.; Kesswani, N. A survey: Intrusion detection techniques for internet of things. *Int. J. Inf. Secur. Priv. (IJISP)* **2019**, *13*, 86–105. [CrossRef]
6. Tabassum, A.; Erbad, A.; Guizani, M. A survey on recent approaches in intrusion detection system in IoTs. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 1190–1197.
7. Aldweesh, A.; Derhab, A.; Emam, A.Z. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowl.-Based Syst.* **2020**, *189*, 105124. [CrossRef]
8. 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]
9. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef]
10. Hatcher, W.G.; Yu, W. A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access* **2018**, *6*, 24411–24432. [CrossRef]
11. University of California. UCI Machine Learning Repository. Network-Based Intrusion Detection (KDD99). Available online: http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data (accessed on 1 October 2020).
12. Network-Based Intrusion Detection (NSL-KDD). Available online: https://www.unb.ca/cic/datasets/nsl.html (accessed on 1 October 2020).
13. Moustafa, N.; Slay, J. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Inf. Secur. J. A Glob. Perspect.* **2016**, *25*, 18–31. [CrossRef]
14. Almomani, I.; Al-Kasasbeh, B.; Al-Akhras, M. WSN-DS: A dataset for intrusion detection systems in wireless sensor networks. *J. Sens. Vol.* **2016**, *2016*, 1–17. [CrossRef]
15. Verma, A.; Ranga, V. Addressing flooding attacks in IPv6-based low power and lossy networks. In Proceedings of the TENCON 2019 IEEE Region 10 Conference (TENCON), Kochi, India, 17–20 October 2019; pp. 552–557.
16. Deshmukh-Bhosale, S.; Sonavane, S.S. A real-time intrusion detection system for wormhole attack in the RPL based Internet of Things. *Procedia Manuf.* **2019**, *32*, 840–847. [CrossRef]
17. Otoum, Y.; Liu, D.; Nayak, A. DL-IDS: A deep learning–based intrusion detection framework for securing IoT. *Trans. Emerg. Telecommun. Technol.* **2019**, *33*, 1–14. [CrossRef]
18. Diro, A.A.; Chilamkurti, N. Distributed attack detection scheme using deep learning approach for Internet of Things. *Future Gener. Comput. Syst.* **2018**, *82*, 761–768. [CrossRef]
19. Almiani, M.; AbuGhazleh, A.; Al-Rahayfeh, A.; Atiewi, S.; Razaque, A. Deep recurrent neural network for IoT intrusion detection system. *Simul. Model. Pract. Theory* **2020**, *101*, 102031. [CrossRef]
20. Parra, G.D.L.T.; Rad, P.; Choo, K.-K.R.; Beebe, N. Detecting Internet of Things attacks using distributed deep learning. *J. Netw. Comput. Appl.* **2020**, *163*, 102662. [CrossRef]
21. Muna, A.-H.; Moustafa, N.; Sitnikova, E. Identification of malicious activities in industrial internet of things based on deep learning models. *J. Inf. Secur. Appl.* **2018**, *41*, 1–11.
22. Shukla, P. ML-IDS: A machine learning approach to detect wormhole attacks in Internet of Things. In Proceedings of the 2017 Intelligent Systems Conference (IntelliSys), London, UK, 7–8 September 2017; pp. 234–240.
23. Bostani, H.; Sheikhan, M. Hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on MapReduce approach. *Comput. Commun.* **2017**, *98*, 52–71. [CrossRef]
24. OMNeT++, Discrete Event Simulator. Available online: https://www.omnetpp.org/ (accessed on 1 November 2022).
25. McKinney, W. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; Volume 445, pp. 51–56.
26. Walt, S.v.d.; Colbert, S.C.; Varoquaux, G. The NumPy array: A structure for efficient numerical computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30. [CrossRef]
27. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the International Conference of Machine Learning—ICML, Atlanta, GA, USA, 16–21 June 2013; Volume 30, p. 3.
28. Tensorflow. Available online: https://www.tensorflow.org/ (accessed on 1 October 2020).
29. Keras. Available online: https://keras.io/ (accessed on 1 October 2020).

30. Bhargava, N.; Sharma, G.; Bhargava, R.; Mathuria, M. Decision tree analysis on j48 algorithm for data mining. *Proc. Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2013**, *6*, 74–78.
31. Zhang, M.-L.; Zhou, Z.-H. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognit.* **2007**, *40*, 2038–2048. [CrossRef]
32. Tsochantaridis, I.; Hofmann, T.; Joachims, T.; Altun, Y. Support vector machine learning for interdependent and structured output spaces. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; p. 104.
33. Sundermeyer, M.; Schlüter, R.; Ney, H. LSTM neural networks for language modeling. In Proceedings of the Thirteenth Annual Conference of the International Speech Communication Association, Portland, OR, USA, 9–13 September 2012.