telecom

MDPI

Article

Incremental Online Machine Learning for Detecting Malicious Nodes in Vehicular Communications Using Real-Time Monitoring

Souad Ajjaj [1,*], Souad El Houssaini [2], Mustapha Hain [1] and Mohammed-Alamine El Houssaini [3]

[1] ENSAM, AICSE Laboratory, Hassan II University, Casablanca 20000, Morocco
[2] Faculty of Sciences, Chouaib Doukkali University, El Jadida 24000, Morocco
[3] ESEF, Chouaib Doukkali University, El Jadida 24000, Morocco
* Correspondence: souad.ajjaj-etu@etu.univh2c.ma

Abstract: Detecting malicious activities in Vehicular Ad hoc Networks (VANETs) is an important research field as it can prevent serious damage within the network and enhance security and privacy. In this regard, a number of approaches based on machine learning (ML) algorithms have been proposed. However, they encounter several challenges due to data being constantly generated over time; this can impact the performance of models trained on fixed datasets as well as cause the need for real-time data analysis to obtain timely responses to potential threats in the network. Therefore, it is crucial for machine learning models to learn and improve their predictions or decisions in real time as new data become available. In this paper, we propose a new approach for attack detection in VANETs based on incremental online machine learning. This approach uses data collected from the monitoring of the VANET nodes' behavior in real time and trains an online model using incremental online learning algorithms. More specifically, this research addresses the detection of black hole attacks that pose a significant threat to the Ad hoc On Demand Distance Vector (AODV) routing protocol. The data used for attack detection are gathered from simulating realistic VANET scenarios using the well-known simulators Simulation of Urban Mobility (SUMO) and Network Simulator (NS-3). Further, key features which are relevant in capturing the behavior of VANET nodes under black hole attack are monitored over time. The performance of two online incremental classifiers, Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN), are assessed in terms of Accuracy, Recall, Precision, and F1-score metrics, as well as training and testing time. The results show that ARF can be successfully applied to classify and detect black hole nodes in VANETs. ARF outperformed KNN in all performance measures but required more time to train and test compared to KNN. Our findings indicate that incremental online learning, which enables continuous and real-time learning, can be a potential method for identifying attacks in VANETs.

Keywords: VANETs; routing attacks; detection; incremental learning; online learning

Citation: Ajjaj, S.; El Houssaini, S.; Hain, M.; El Houssaini, M.-A. Incremental Online Machine Learning for Detecting Malicious Nodes in Vehicular Communications Using Real-Time Monitoring. *Telecom* **2023**, *4*, 629–648. https://doi.org/10.3390/telecom4030028

Academic Editor: Barbara M. Masini

Received: 22 June 2023
Revised: 14 July 2023
Accepted: 5 September 2023
Published: 11 September 2023

1. Introduction

Vehicular Ad hoc Networks (VANETs) is a type of Mobile Ad hoc Network (MANET) consisting of vehicles equipped with advanced communication devices and sophisticated features. VANETs provide a platform for various applications, such as safety, traffic efficiency, and infotainment services [1]. However, VANETs are subject to several kinds of attacks, including denial-of-service, jamming, black holes, and worm hole attacks, that might compromise the network's security and privacy [2]. To detect these attacks, various approaches based on machine learning algorithms are suggested [3]. These methods are proven to be effective in detecting both known and unknown attacks, but they face several challenges in VANET networks [4]. First, attack patterns may change over time, which may affect the performance of a model trained on a fixed dataset [5]. Second, the complexity of the data in VANET networks can increase rapidly over time, making it challenging to

scale traditional machine learning algorithms to handle new patterns in the data. In that regard, incremental online learning algorithms can be an effective solution for real-time applications compared to traditional machine learning techniques, which typically require large training datasets. Incremental online learning algorithms allows the model to learn continuously over time, adapting to new data as they become available [6].

The core interest of this study is to explore incremental online learning algorithms in the context of attack detection in VANETs. More specifically, our research focuses on the detection of the black hole attack, estimated as the most damaging attacks that might compromise Ad hoc On Demand Distance Vector (AODV) routing protocol in VANETs in terms of data availability [7,8]. To this end, we have used data gathered from extensive simulations of realistic VANET scenarios based on two well-known simulators, namely SUMO [9] and NS-3 [10]. The collected data incorporate essential features which are relevant in capturing the behavior of VANET nodes under a black hole attack. We selected more appropriate features, namely CTRLpackets, CountRREQ, CountRREP, CountRERR, Throughput, SentPckts, ReceivedPckts and Dropping ratio. Further, the performance of two incremental online classifiers, namely Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN), is investigated [11,12]. Additionally, metrics such as Accuracy, Precision, Recall, and F1-score are assessed in order to track the classifiers' performance over time.

The findings demonstrate that ARF outperforms the KNN algorithm in terms of Accuracy, Precision, Recall, and F1-score. However, the latter records a higher running time. Furthermore, this study outlines the effectiveness of the proposed approach as it enables the creation of constantly updated models by processing small amounts of data at a time, rather than the entire dataset at once, unlike traditional machine learning methods that require substantial computing resources when dealing with large datasets. Further, this approach is especially beneficial in dynamic applications such as VANET networks, where data are continuously generated in a streaming manner. Moreover, it is adaptive, allowing it to adapt and improve predictions or decisions as new data arrive, thereby accommodating changing data patterns over time.

The main contributions of this paper are as follows:

- We introduce a more dynamic approach for detecting attacks based on incremental online machine learning algorithms trained on data generated in real-time;
- We collect data form VANET scenarios using a robust methodology for VANET simulations based two well-known simulators, namely SUMO and NS-3;
- We select essential features that are relevant in capturing the behavior of black hole nodes in the AODV routing protocol;
- We assess the overall performance of classifiers in terms of multiple performance metrics, namely Accuracy, Precision, Recall and F1-score. Further, each performance metric is tracked over time to continuously evaluate the classifiers;
- The complexities of both classifiers in terms of training and testing time are computed and compared.

The rest of this paper is organized as follows: in Section 2, related works on attack detection in VANETs using ML-based approaches are reviewed. Section 3 presents the materials and methods including the proposed method, the experimental settings, the simulation environment, the data collection, as well as the incremental algorithms considered. Results and discussion are provided in Section 4. Finally, we summarize the achieved results and point out future research directions.

2. Related Works

Machine learning is an intensively researched area within the broader field of artificial intelligence (AI), as it has been used in a variety of real-world applications, such as image identification, speech recognition, natural language processing, autonomous cars, attack detection, fraud detection, etc. [3]. In the research area of attack detection in VANETs, several solutions based on machine learning algorithms have been suggested.

In what follows, we present the most recent studies that addressed attack detection in VANETs.

In the paper [13], the authors propose a novel technique for detecting jamming in wireless networks within vehicles. They utilize unsupervised machine learning and introduce a new metric called variations of the relative speed (RSV) between the jammer's vehicle and the receiver's vehicle. Additionally, parameters from the wireless communication systems on the receiving vehicle are taken into account. By applying unsupervised learning through clustering, the proposed method effectively distinguishes intentional and unintentional jamming and identifies the unique characteristics associated with each jamming attack. The authors highlight the importance of relative speed and its variations in successful jamming detection. They also demonstrate that relying solely on conventional wireless receiver metrics from the physical and network layers, such as PDR, SINR, and RSSI, is insufficient for discriminating interference from deliberate jamming events or recognizing the specific traits of an attack.

In another work [14], the objective is to analyze safety messages and detect false position information sent by misbehaving nodes. Machine learning (ML) techniques are applied to the VeReMi dataset to detect such misbehavior. The researchers consider the ML-based approach to be a practical and efficient method for identifying improper behavior in real-world VANET scenarios. The results indicate that SVM with normalization outperforms logistic regression with or without normalization. The goal of study [15] is to use machine learning to identify wormhole attacks in VANETs' multi-hop communication. The authors create a scenario of multi-hop communication using the AODV routing protocol on the NS3 simulator, employing mobility traces generated by the SUMO traffic simulator to replicate the attack in the VANET. The gathered traces are preprocessed to enable the model to learn about wormhole attacks, and then, K-NN and SVM algorithms are applied. The two machine learning techniques are evaluated in terms of detection accuracy for four different alert types.

In [16], various machine learning techniques are employed to identify five different attacks: constant attack, random attack, constant offset attack, eventual attack, and random offset attack. Initially, binary classification methods are used to identify each attack individually. Subsequently, a novel method is developed for multi-classification of attacks. The accuracy obtained for each attack type varies depending on the machine learning algorithm used. The Random Forest technique achieves the highest accuracy in the case of the new multi-classification procedure. The VeReMi dataset is used for malicious node detection in VANETs.

Another study [17] suggests a new machine learning approach that utilizes Random Forest and posterior detection based on coresets to enhance the performance of Intrusion Detection Systems (IDSs). The results show that, in comparison to conventional machine learning models utilized in related applications, the proposed model greatly increases detection accuracy.

For attack classification in VANETs, a hybrid optimization-based Deep Maxout Network (DMN) is developed in [18]. The hybrid optimization approach is used for Cluster Head (CH) selection and routing operations. Feature selection is crucial for efficient classification, and the DMN is employed for attack classification with a new optimization approach. The optimization model based on the DMN improves routing performance, energy consumption, and trust metrics. Precision and Recall measures are also reported. In [19], an Intelligent Intrusion Detection System (IDS) that combines deep learning and machine learning techniques is proposed. Convolutional Neural Networks (CNN) and the Adaptive Neuro Fuzzy Inference System (ANFIS) are both used in the system. This strategy overcomes this constraint by making use of soft computing techniques and an Intelligent IDS system, in contrast to previous systems that focus on identifying known threats in the VANET environment. A hybrid technique is used to identify several attack types, such as Denial-of-Service (DoS), Botnet, PortScan, and Brute Force assaults. Real-time data from

the CIC-IDS 2017 dataset are used for evaluation, and the proposed methodology shows effectiveness compared to other state-of-the-art techniques.

In study [20], a practical and effective machine learning (ML)-based method for detecting malicious behavior is suggested. The Vehicular Reference Misbehavior Dataset (VeReMi) is utilized by the proposed machine-learning-based misbehavior detection system. The VeReMi dataset contains labeled examples of five different types of position falsification attacks with varying vehicle and attacker densities. The authors propose a model that employs two sequential BSM approaches to detect these attacks. The categorization of the roadside unit's model enables the identification and removal of harmful nodes from the network, reducing computational overhead on moving vehicles. Researchers used SVM in the study [21] to identify false message attacks using the driving status, speed, acceleration, vehicle type, reputation, and distance as features. Additionally, they investigated message suppression strategies involving packet loss, packet delay, and packet forwarding. In order to combine various assessments, their proposed vehicle trust model calls for both a local vehicle trust module and a central Trust Authority (TA). In another paper by [22], authors provide a machine learning (ML) mechanism that makes use of three new features, namely those that relate to the sender position, to improve the performance of IDS against position falsification attacks. Additionally, it compares K-Nearest Neighbor (KNN) and Random Forest (RF), two distinct machine learning (ML) algorithms for classification that are used to identify malicious vehicles using these features.

All the aforementioned studies used approaches that have proven to be effective in detecting known and unknown attacks. However, they face several challenges in VANETs because of the evolving nature of attack patterns as well as the complexity of the data which can increase rapidly over time, making it difficult to scale traditional machine learning algorithms to handle data generated in real-time. In the next section, we introduce the use of incremental online machine learning algorithms in the context of attack detection in VANETs.

3. Materials and Methods

In this section, we first describe the proposed approach for attack detection after introducing incremental online machine learning. The data collection process is extensively detailed including experimental settings, simulation environment, and selected features. Tools and frameworks used for the implementation of incremental online algorithms are also described.

3.1. Incremental Online Learning

Recently, incremental online learning has received more attention, especially in the context of learning from data streams where data continually arrive, such as in IoT applications, spam filtering, attack detection, time series forecasting, etc. [5]. This evolution challenges traditional machine learning algorithms, which assume that they have access to the entire training dataset during the learning stage [3].

Using traditional machine learning algorithms consists of collecting large historical data, with the model being trained on the training set and validated on the testing set. In this way, the model is trained on a large dataset that has been collected and labeled beforehand [23]. This approach performs well when all the data are available up front since the model is trained on a large static dataset [6].

In contrast, incremental online learning algorithms involves building dynamic models which are trained in real-time, rather than on a fixed dataset all at once [4].

Formally, let D be a data stream including instances denoted by

$$\left(\vec{x}_i, y_i\right), (\mathrm{i} = 1, \ldots\ldots.\mathrm{t}, \ldots.) \tag{1}$$

where t represents a timestamp, $\vec{x}_i$ is a vector of features, and y_i is the target or the class.

Incremental online learning allows the model to learn continuously from each new data instance. The goal here is to continue to learn from new incoming data and update the model continuously [24]. The updated model can be used to predict a label y_i' for an unlabeled $\vec{x}_i'$. The primary difference between traditional and online learning is how the data are presented [25]. In the former, the dataset is static and entirely available, whereas data instances are presented sequentially over time in online learning. Incremental online learning has a number of characteristics including:

- Incremental learning allows for the use of data as they becomes available, creating models that are constantly up-to-date instead of having to process the entire dataset at once [5]. This can be very helpful in dynamic applications like VANET networks where data are produced in a streaming way [6];
- There is a need to process a small amount of data at a time to learn from it [25]. This is in contrast to traditional machine learning techniques, which need a lot of computing resources, particularly when training on big datasets [24];
- The ability to handle streaming data in real-time [6]. This makes it particularly useful in this area of research, where data are constantly being generated and need to be analyzed in real-time [4];
- Adaptive, because it can adjust to changing data patterns over time. This suggests that it is capable of learning and improving its predictions or decisions as new data arrive [23].

The proposed approach for detecting malicious nodes in VANETs is outlined in the following section.

3.2. *Proposed Method*

The process of the proposed method for attack detection based on incremental online learning is outlined in the diagram below (Figure 1):

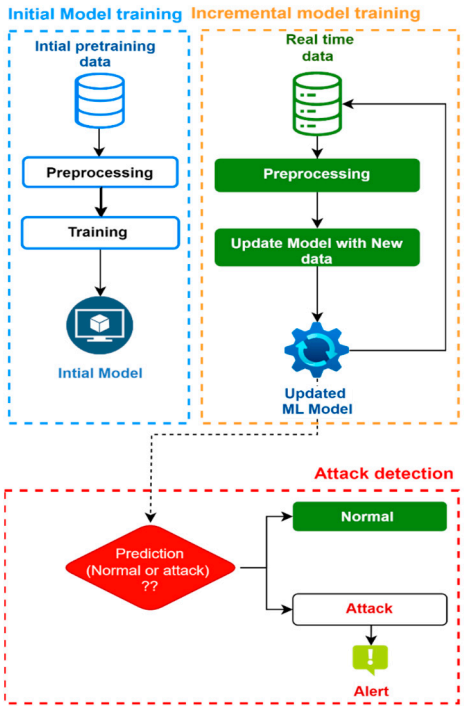

Figure 1. The process of the proposed approach for attack detection using incremental online learning.

The basic idea of the proposed approach is to use the data collected from the monitoring of the VANET nodes behavior in real-time and train an online model using incremental online learning algorithms. The incremental model performed is used for classifying

VANET nodes as normal or attacker. The primary steps of the proposed method are as follows:

- Initial model training

An initial model is trained on initial data to build a base model before online incremental learning is performed. This model will serve as the starting point for the online incremental learning process.

- Incremental model training

The model is incrementally updated after processing each new data instance provided in real-time. This process is repeated each time a new data instance is available, and the updating of the models occurs continually [6]. The model is updated or refined employing new data rather than being trained from scratch on the entire dataset. This ensures that the model is always up-to-date. The updated model is used for classifying VANET nodes as normal or malicious.

- Attack detection

This component performs online classification of unlabeled data, leveraging the updated trained model to classify new data. Online classification makes it possible to detect attacks in real-time by identifying patterns that indicate malicious activity. If the pattern of the data is matched with an attack, an alarm is triggered.

The proposed attack detection method involves data collection using a monitoring system designed to record various metrics and behavioral parameters within each vehicle node in the network. This includes recording the most relevant features to distinguish between normal nodes and black hole nodes in the network.

The following subsections provide technical details on how the data are collected and preprocessed. This includes network simulation environment, scenario-based data collection, and the most relevant features for attack detection.

3.3. Data Collection

3.3.1. Simulation Environment and Scenarios

In this study, the steps followed to simulate the VANET network are as follows:

Step 1: The preparation of a road network by importing a map from Open Street Map (OSM) [26]. The simulation zone is defined on a map form the Moroccan city El Jadida (Figure 2).

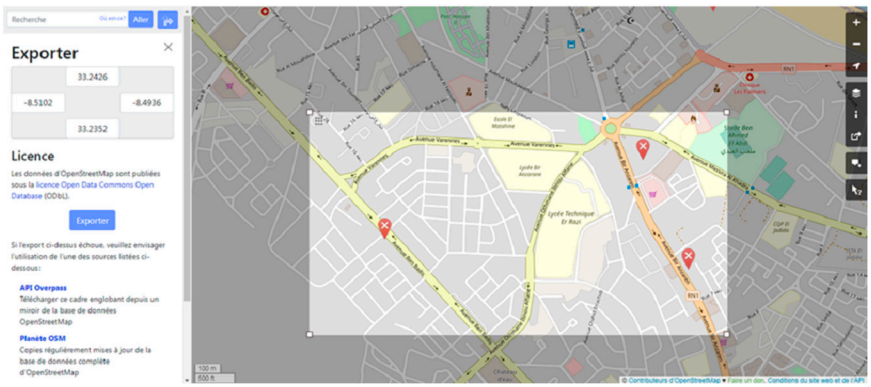

Figure 2. The simulation map from Open Street Map.

Step 2: The generation of mobility trace files using SUMO (Simulation of Urban Mobility) [9] which simulate realistic vehicle movements. SUMO provides a variety of python tools such as netconvert, randomTrips, poly-convert, and traceExporter used for the creation, execution, and evaluation of traffic simulations. The generated mobility trace file is then fed to the network simulator NS-3 (Figure 3).

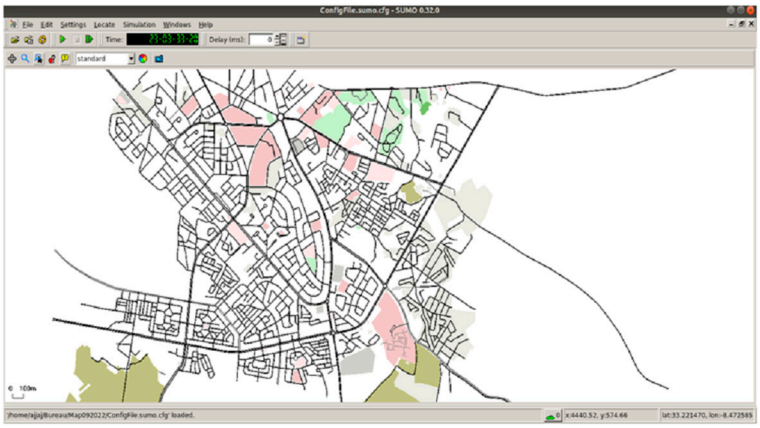

Figure 3. Network XML file edited by SUMO.

Step 3: Simulate the network components using the NS-3 network simulator, which is used to model the VANET communications' whole protocol stack. Simulation scenarios are configured in NS-3 using scripts written in the C++ programming language. Simulations are implemented using parameters exhibited below (Table 1).

Table 1. Simulation settings.

Parameter	Value
Platform	Linux, Ubuntu environment.
Simulator of network	NS3.29
Simulator of Mobility	SUMO-0.32.0
Routing protocol	AODV
Mac/Phy Layer	IEEE 802.11p
WiFichannel	YansWifi
Propagation model	friisLoss model
Transmission power	33 dbm
Transport protocol	UDP
Traffic type	CBR (constant bit rate)
Packet size	64 bytes
Number of vehicles	50
Runtime	360 s

We consider two VANET scenarios using the simulation parameters previously outlined in Table 1.

In the first scenario, 50 vehicles are involved, with 10 random source–destination pairs, which produce CBR traffic with fixed-size packets. The AODV routing protocol is used to route packets considering all nodes as legitimate vehicles. In this case, no black hole node is simulated.

In the second simulated scenario, we implement AODV routing protocol with black hole attacks wherein 1 to 3 nodes are selected to act as attackers and are configured at different times to launch the black hole attacks. The remainder of the nodes are trustworthy and have legitimate communications to the other vehicles in the network.

3.3.2. Definition of Features

During the simulation, a monitoring system is designed to record various metrics and behavioral parameters within each vehicle node during the normal scenario and the black hole attack scenario. This includes recording the most relevant information to distinguish between a normal node and a black hole node in the network.

Hereafter, a brief description of the characteristics of AODV routing protocol [27] and its vulnerability to black hole attack, which will help us selecting essential features relevant in detecting the black hole attack [7].

AODV relies on two fundamental mechanisms, namely route discovery and route maintenance. The protocol utilizes RREQ (Route Request) and RREP (Route Reply) messages for route discovery. When a source node needs to send data to a destination, it checks its routing table for a valid route. If there is no valid route available (possibly due to non-existence, expiration, or failure), the source node broadcasts an RREQ message to all neighboring nodes.

Each intermediate node forwards the RREQ message until it reaches either the destination or an intermediate node that has a valid route to the destination. The node with a valid route responds to the source node by sending a unicast RREP message along the reverse path.

Upon receiving the RREP message, the source node proceeds with transmitting data packets [27]. Regarding the mechanism of route maintenance in AODV, active routes are the only ones that are maintained. Nodes regularly communicate with their neighbors by sending HELLO messages in order to analyze the status of the links and identify whether a path is fresh or not.

If a link breaks, a Route Error (RERR) message is sent to the source node and any other nodes affected by the broken link. The nodes are informed of the link failure by this RERR message.

The AODV routing protocol's activities are illustrated in Figure 4 [7].

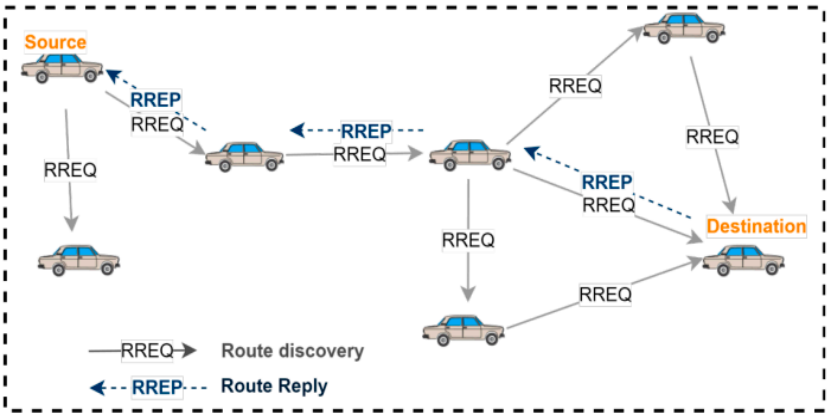

Figure 4. AODV routing protocol mechanism.

A severe kind of denial of service known as a "black hole attack" occurs when a rogue node purposefully intercepts routing request (RREQ) messages from nearby nodes.

Instead of sending these RREQ packets as intended to other nodes, the malicious node disrupts the route discovery process by rapidly providing a false route reply message (RREP) with the highest sequence number.

As a result, the malicious node receives data packets from the source node, assuming it is the best route, under the erroneous idea that the route discovery was successful. However, the misbehaving node captures all the routing packets and intentionally discards them, effectively preventing any successful communication [1].

Based on these unique behavioral characteristics of the black hole attack, we will choose features that are the most relevant to distinguish between a normal node and a black hole node in the VANET network.

Here are the key considered points when recording various metrics and behavioral parameters:

- Routing Behavior Analysis: this involves tracking the overall routing control packets in AODV and monitoring particular packets such as RREQ (Route Request), RREP (Route Reply), and RERR (Route Error). These parameters can contribute to the detection of the black hole attack, so they are summarized in the CTRLpackets, CountRREQ, CountRREP, and CountRERR features;

- Traffic Analysis: this involves monitoring the traffic characteristics and keeping track of the number of bytes that are sent or received by each node;
- Dropping ratio monitoring: observing the dropping rate of packets can help in the identification of nodes that selectively drop or discard incoming packets, indicating malicious behavior;
- Throughput monitoring: black hole attack typically drops a large amount of data, which may decrease significantly the throughput, making it an important metric to monitor for detection purposes.

The set of features recorded in our dataset are summarized in Table 2: CTRLpackets, Count RREQ, CountRREP, CountRERR, Throughput, Sent Pckts, Received Pckts, and DroppingRatio.

Table 2. Relevant features for black hole attack detection.

Feature Name	Feature Description
CTRLpackets	AODV routing control packets: a black hole node may advertise a fake and optimized route to the destination. Thus, it is essential to keep track of the routing control packets and detect any changes in their number advertised by a node.
CountRREQ	Number of Route Request messages that are used by the nodes to discover new routes to other nodes in the network. In the case of a black hole attack, the malicious node may not generate any RREQ messages because it is not interested in receiving any packets, and instead, it drops all the packets that it receives.
CountRREP	Number of Route Reply messages that are sent by nodes in response to RREQ messages to establish a route to the destination node. In the case of a black hole attack, the black hole sends fake RREP; if a large number of RREP messages are received from a single node, it could be an indication that the node is a potential black hole.
CountRERR	Number of Route Error (RERR) Messages: a black hole node may generate an abnormal amount of RERR messages, indicating that the node is not following the protocol's correct path discovery mechanism.
Throughput	Throughput measures the amount of data that can be transferred in a given time. A sudden decrease in throughput can be an indication of a black hole attack.
SentPckts	The number of sent packets. The attacker node captures and drops all routing packets, leading to a significant decrease in the number of successfully sent packets.
ReceivedPckts	The number of received packets. The rogue node selectively drops all the data packets received from other nodes, which may cause a decrease in the number of bytes that are received.
DroppingRatio	The dropping ratio measures the percentage of packets dropped by a node, and this feature can be used to detect if a node is dropping a higher number of packets than normal.

3.4. Data Preprocessing

In the context of our study, data normalization is employed to scale features to a common range between 0 and 1 or -1 and 1, which make it simpler to compare and evaluate the data.

Different normalization techniques can be applied, including min–max scaling, z-score normalization, and log transformation [28].

Z-score normalization is employed to transform each instance of the original data xi into x_i' using the following equation:

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{2}$$

where μ and σ denote the mean and standard deviation, respectively.

3.5. Incremental Online Algorithms

In the current study, we selected two incremental online classifiers, namely Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN) [11,12]. The algorithms are chosen based on their popularity in the online classification area and are freely available in the python framework sickit-multiflow [12].

Adaptive Random Forest (ARF): a modified version of the Random Forest (ARF) method developed to handle data streams. The algorithm includes important extended techniques that modify the way the trees are trained and sampled in order to adapt to streaming data: the online bagging and adaptive resampling techniques. In the standard Random Forest algorithm, each tree is trained on a bootstrap sample of the training data, which is obtained by randomly sampling with replacement from the original data. However, in data streams, new instances arrive continuously, and it is not feasible to store the entire training data in memory. The core idea underlying online bagging is to train a tree in a forest using a sample that is generated by randomly selecting instances from the data stream with replacement. The trees in the forest differ from one another due to the use of several samples, resulting in a varied ensemble. Adaptive resampling is another technique used to adapt to changes in the stream by updating the weight of instances according to their arrival times. Formally, instances that are more recent are given a higher weight, which enables the classifier to adapt to concept drift over time. This technique is used to update the weights of instances in the sample used to build the trees [6,11].

The pseudo code of the ARF algorithm is given in Algorithm 1. This algorithm begins by initializing a specified number of trees in an ensemble. Each new instance in the data stream is then sent to each tree in the ensemble for testing and training. ARF follows a "test-then-train" approach, where an incoming instance is first used to test the model's performance by making a prediction and estimating its accuracy. After testing, the instance is used to train the model. The training process in ARF utilizes online bagging, a technique that randomly selects a subset of features and performs splits based on these features.

Algorithm 1: Pseudo Code of Adaptive Random Forest

Inputs: n_trees: the number of trees in the ensemble, STREAM: the stream of data instances, f_s_size: size of the random subset of features to select for each split.
Outputs: arf_model: the trained ARF model

1. Initialize an ensemble with a specified number of trees.
2. Receive a stream of data instances.
3. For each new instance in the stream:
    a. Test the model's performance by making a prediction.
    b. Estimate the model's performance.
    c. If the performance is below the test threshold, train the model with the instance.
    d. Apply online bagging and select a random subset of features with f_s_size for tree training.
    e. Train all background trees on the current instance.
4. Repeat the above steps for all instances in the stream
5. The trained ARF model, which is the ensemble of trees, is the output.

K-Nearest Neighbors (KNN): a well-known classification algorithm used in machine learning that relies on the distance metric between data points. In online classification, the algorithm works by maintaining a set of labeled instances in memory, which are used to classify new incoming instances based on their proximity in the feature space. For each incoming data instance x, the algorithm computes the distance between x and each instance in the data stream using a distance metric which is typically the Euclidean distance. Subsequently, the K-Nearest Neighbors of x in D based on their distances are found. The class label of x is assigned based on the majority of its K-Nearest Neighbors. In KNN,

K is a hyper parameter that determines the number of nearest neighbors to consider for classification. As new instances are received, the algorithm updates its set of labeled instances and can change its classification decisions accordingly.

The pseudo code below exhibits the core functionality of the KNN algorithm, which involves updating the model with new instances and making predictions based on the K-nearest neighbors.

Algorithm 2: Pseudo Code of K-Nearest Neighbors

Inputs: K: The number of nearest neighbors to consider, M: The memory size or maximum number of instances to retain, STREAM: the stream of data instances.
Outputs: Predicted label for each instance in the data stream

1. Initialize the KNN model:

   Set the number of nearest neighbors (K).
   Set the memory size (M).
   Initialize an empty memory buffer.

2. Receive a stream of data instances:

   For each new instance in the stream:

   a. Find the K nearest neighbors from the instances in the memory buffer using eEuclidien distance metric.
   b. Classify the new instance based on the majority class among the K-nearest neighbors.
   c. Update the model and memory buffer:

      If the memory buffer is not full (number of instances < M), add the new instance to the buffer.
      If the memory buffer is full, replace the oldest instance in the buffer with the new instance.

   d. Store the predicted label for the new instance.

3. Repeat step 2 for all instances in the data stream.

3.6. Prequential Evaluation

In the performance evaluation of machine learning models, the hold-out evaluation method is the commonly used approach, wherein the available data are split into two parts: a training set and a test set [3]. The model is trained on the training set, and its performance is evaluated on the test set to evaluate how well the model can perform on new unseen data.

Prequential evaluation, on the other hand, is a more recent and alternative approach to model evaluation used for data generated continuously in time. In prequential evaluation (or test-then-train evaluation), the basic idea is to use new data instances to firstly test the model and then to train it [29]. The process of prequential evaluation involves the following steps:

- Pretrain the model with initial data;
- Test and train: For each incoming data instance, the model is tested on the current instance and then trained using the same instance, using the partial_fit() method;
- Evaluate: The model's performance is tracked over time where metrics are updated over time.

For the prequential evaluation, several parameters are tuned amongst test step, max_samples, and pretrain_size, which refer, respectively, to the number of samples between each model test, the maximum number of samples to process from the data stream, and the number of samples to use for pretraining the model before starting the prequential learning process [12].

An example of pseudo code for prequential accuracy evaluation is outlined in the Algorithm 3 below:

Algorithm 3: Prequential Accuracy for Model Evaluation

Inputs: *D*: a stream of data (X,y), **Arf:** the classifier to be evaluated, **pretrain_size:** the number of samples for pretraining the model, **Ts**: test step, number of samples to process between each model test, **max_samples**: the maximum number of samples to evaluate.
Outputs: Acc_list: a list containing accuracy measures after each test step.

1. Initialize Acc_list to store accuracy measures
2. Initial counters CorrPred and TotalPred for correct predictions and total predictions.
3. Pretrain the model by calling model.partial_fit() on pretrain_size samples of the data stream.

```
for i in range(pretrain_size)
    (X, y) = D.next_sample()
    Arf.partial_fit(X, y)
end for
```

4. Repeat for max_samples iterations:

```
for i in range(max_samples):
      (X, y) = D.next_sample()
       prediction = model.predict(X)
      Arf.partial_fit(X, y)
      TotalPred+= 1
     if y == prediction:
    CorrPred+= 1
     if TotalPred % Ts == 0:
            accuracy = CorrPred/TotalPred
    Acc_list.append(accuracy)
    CorrPred = 0
    end if
end for
```

5. Return Acc_list

In this study, we use Accuracy, Precision, Recall, and F1-score metrics computed according to the following equations:

$$\text{Accuracy} = (\text{tp} + \text{tn})/(\text{tp} + \text{fp} + \text{fn} + \text{tn}) \tag{3}$$

$$\text{Precision} = \text{tp}/((\text{tp} + \text{fp})) \tag{4}$$

$$\text{Recall} = \text{tp}/(\text{tp} + \text{fn}) \tag{5}$$

$$\text{F1} - \text{score} = 2 * ((\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall})) \tag{6}$$

where tp, fp, fn, and tn are, respectively, number of instances correctly classified as positive, number of instances incorrectly classified as positive, number of instances incorrectly classified as negative, and number of instances correctly classified as negative.

Experiments are performed on Jupyter notebook, and Python 3.9.13 is used along with the Matplotlib 3.5.2, NumPy 1.20.0, and pandas 1.4.4 libraries. Implementation of incremental online algorithms is performed using scikit-multiflow 0.5.3, a Python library that provides a range of tools, algorithms, and evaluation metrics to handle data streams in real-time scenarios. Scikit-multiflow can be used along with other Python libraries like Numpy and SciPy, pandas and scikit-learn.

In the subsequent section, we will exhibit and discuss the results of evaluating the overall performance of the Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN) classifiers.

4. Results and Discussion

In this section, we will first analyze the mean Accuracy of classifiers under varying values of the pretrain_size parameter, which refers to the number of samples to use for

pretraining the model before starting the prequential learning process. We will then compare the overall performance of the classifiers according to Accuracy, Recall, Precision, and F1-score metrics. Plots that monitor the performance metrics of both classifiers over time are also given. Finally, the running time (training time and testing time) of both classifiers are assessed and compared.

The dataset used for evaluation includes eight features and two classes (Normal, BKH) with 4492 data samples as Normal and 4670 as black hole.

Regarding the choice of the pretrain_size parameter for prequential evaluation, we evaluated the mean Accuracy of both classifiers under varying values of the pretrain_size in order to choose the most appropriate value. Table 3 reports the obtained results.

Table 3. The mean Accuracy of classifiers under varying pretrain size values.

Classifier	Pretrain_Size			
	400	**600**	**1000**	**1200**
Adaptive Random Forest (ARF)	94.73%	94.77%	94.83%	94.90%
K-Nearest Neighbors classifier (KNN)	86.86%	87.13%	87.30%	87.83%

The results exhibited in Table 3 show that both classifiers Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN) record Accuracy above 86% for the different values of Pretraining size. It is also observed that Adaptive Random Forest (ARF) achieves higher Accuracy values than the KNN, which reaches almost 95% for the different values of the pretrain_size parameter. Further, we note that the overall Accuracy increases as the pretrain_size increases (varying from 400, 600, 1000 to 1200) for both classifiers.

The Accuracy results of Adaptive Random Forest (ARF) varies from 94.73% to 94.90%, presenting a slight increase, while KNN improves its Accuracy from 86.86% to 87.38% when increasing the pretrain_size from 400 to 1200.

Overall, the results of the carried-out experiments clearly indicate that using ARF achieves good Accuracy even with small pretraining data sizes, while the KNN's classifier Accuracy improves as the pretrain_size values increase.

This can be explained by the inherent characteristics of the Adaptive Random Forest (ARF) algorithm which is based on ensemble learning. ARF tends to leverage diverse models trained on different subsets of data to make a decision about the class label. This allows ARF to perform well in situations with a small number of pretraining samples. The K-Nearest Neighbors (KNN) classifier, on the other hand, relies on the nearest neighbors for classification.

This suggest that when increasing the pretraining data size, the KNN algorithm can achieve better performance.

Therefore, considering the above findings, we adopt {pretrain-size = 1200} for the evaluation of the classifiers in the rest of our study.

4.1. Performance Assessment of Classifiers

The mean Accuracy, Precision, Recall, and F1-score of each classifier are reported in Table 4

Table 4. The mean performance of ARF and KNN classifiers.

Incremental Classifier	Accuracy	Precision	Recall	F1-Score
Adaptive Random Forest (ARF)	94.90%	93.10%	96.81%	94.92%
KNN Classifier	87.38%	87.58%	86.62%	87.09%

From Table 4, we notice that both ARF and KNN classifiers achieve good performance. However, Adaptive Random Forest (ARF) outperforms the KNN classifier in terms of Accuracy, Precision, Recall, and F1-score.

ARF reaches 94.90%, 93.10%, 96.81% and 94.92% for Accuracy, Precision, Recall and F1-score, respectively, while KNN records a lower performance: the Accuracy is 87.38%, Precision is 87.58%, Recall is 86.62%, and F1-score is 87.09%.

These findings suggest that Adaptive Random Forest may be a more suitable choice for detecting malicious nodes in VANETs, offering improved predictive capabilities compared to the KNN algorithm.

Complementing the above results, the variations of the performance of both classifiers over time according to each performance metric are plotted.

Figures 5–8 illustrate the monitoring over time of the performance of the classifiers regarding Accuracy, Precision, Recall, and F1-score, respectively.

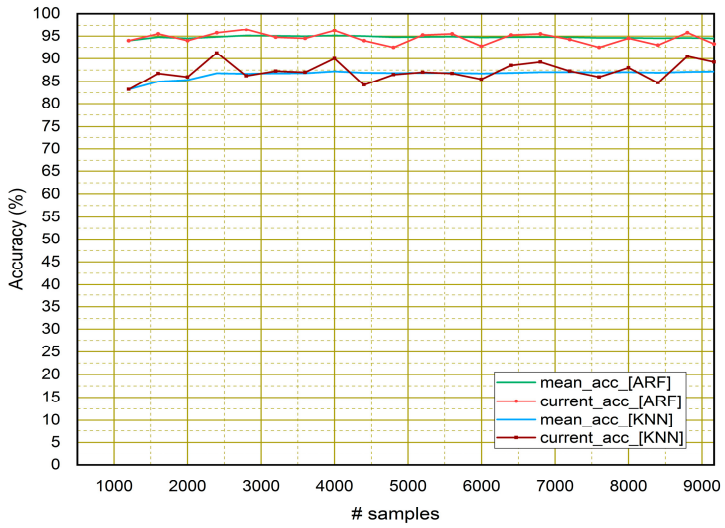

Figure 5. Accuracy of ARF and KNN classifiers over time.

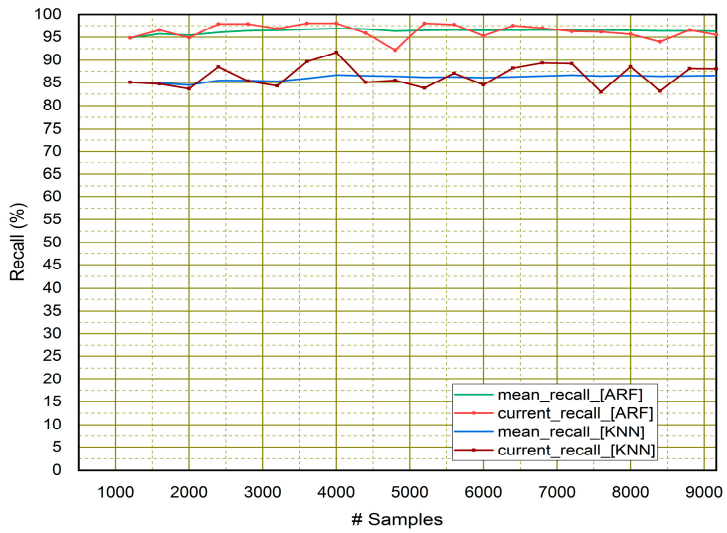

Figure 6. Recall of ARF and KNN classifiers over time.

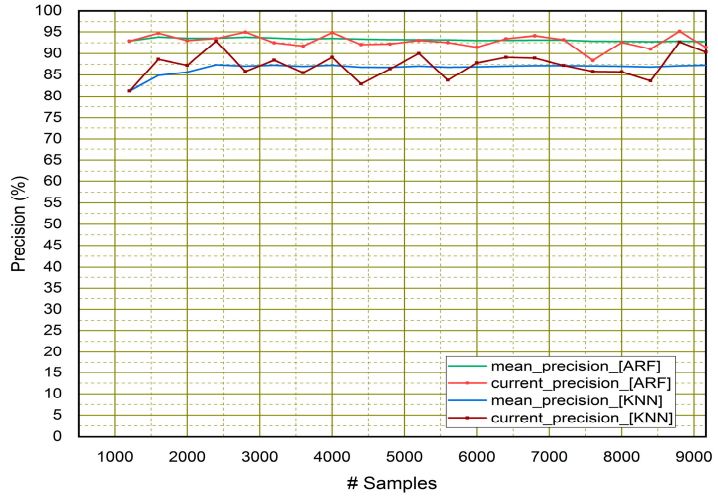

Figure 7. Precision of ARF and KNN classifiers over time.

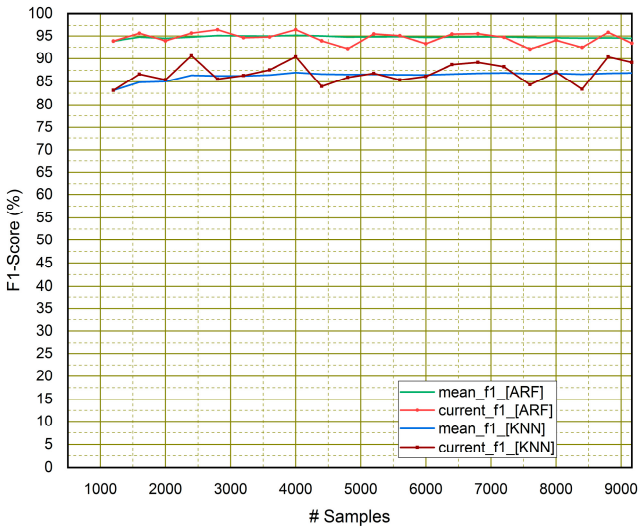

Figure 8. F1-score of ARF and KNN classifiers over time.

For each performance metric, the mean and the current values are plotted. We note that the mean Accuracy is obtained by averaging the Accuracy scores obtained in all test steps while the current Accuracy refers to the Accuracy of the classifier on the current test step.

For each classifier, Figures 5–8 exhibit both the mean and the current performance over time.

Figure 5 represents the monitoring of Accuracy for the ARF and KNN algorithms. First, we notice that the Accuracy of ARF is higher than that of KNN throughout the evaluation time, with a mean Accuracy ranging from 93% to 95% for ARF while KNN records a lower performance with a mean Accuracy between 83% and 87%.

Further, we observe that the plots of the mean Accuracy for both ARF and KNN appear to be stable throughout the evaluation time. This outcome is logical since the mean Accuracy is an average measure which computes the overall performance of the classifier over multiple test periods.

These outcomes suggest that both classifiers show consistent Accuracy over time. Similarly, examining the plots of the current Accuracy of both ARF and KNN, we point out that ARF achieves a higher Accuracy between 92% and 97%, outperforming KNN, whose current Accuracy ranged from 83% to 91%.

Overall, the plots of current Accuracy present more variations in comparison with those of mean Accuracy. This is because current Accuracy reflects the performance of the classifier on the most recent test period of time.

Considering the above results, we can conclude that monitoring both mean and current Accuracy over the evaluation time can be useful in monitoring the real-time performance of the classifiers and provide a more reliable estimate of the classifier's ability to detect misbehavior.

Similarly, Figures 6–8 capture the variations of Recall, Precision, and F1-score metrics over time. Both the plots of mean and current metric are monitored. ARF shows superior performance in comparison with KNN regarding all metrics, namely Recall, Precision, and F1-score.

In summary, the carried-out experiments demonstrate that Adaptive Random Forest (ARF) has superior performance than the K-Nearest Neighbors (KNN) algorithm regarding Accuracy, Recall, Precision, and F1-score metrics in the context of our study.

These findings can be attributed to the specificities of each algorithm. On the one hand, Adaptive Random Forest (ARF) is based on an ensemble approach that combines multiple decision trees to make predictions, which allows for making accurate predictions. Further, ARF's online bagging mechanism allows reducing the impact of poor or irrelevant features on the classification process. This technique makes ARF capable of making accurate predictions by handling imbalance in data and capturing the data patterns more accurately [11]. On the other hand, KNN relies on the calculation of a distance metric and makes predictions based on the majority of the K-Nearest Neighbors [12].

In what follows, we aim to analyze the training and testing time of the ARF and KNN algorithms.

4.2. Training and Testing Time

Figure 9 shows the variations of ARF and KNN models in terms of total running time, which include both training time and testing time.

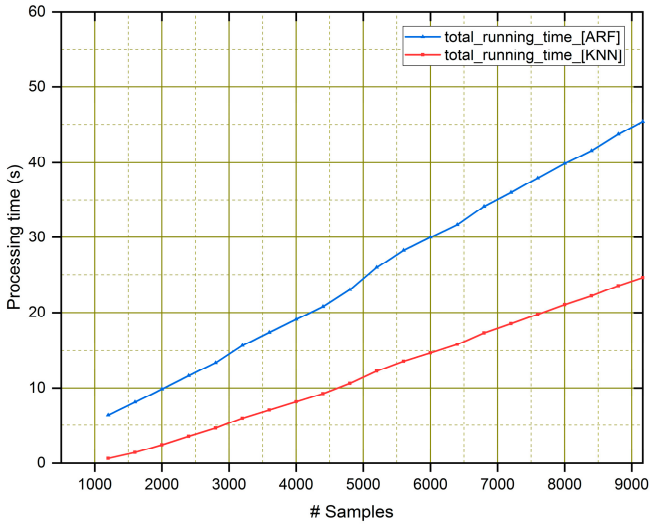

Figure 9. Training and testing time for ARF and KNN.

It is observed that ARF requires an average time of 56.53 s for training and testing, whereas the average training and testing time for KNN is 28.43 s.

These findings can be attributed to the fact that the two algorithms have fundamental differences. KNN relies on relatively simple techniques based on determining the K-Nearest Neighbors and assigning the most common class label as the new label for the data point [30]. ARFs, on the other hand, are ensembles of decision trees that combine

decisions from various trees and aggregate their results to obtain the final output [6,11]. Consequently, ARF might require higher training and testing time than KNN.

4.3. Comparison with State-of-the-Art Methods

In this section, we present a concise comparison of our proposed work with the state-of-the-art methods explored in this study as exhibited in Table 5.

Table 5. Comparison with state-of-the-art ML methods for attack detection.

Study	Approach	Real-Time	Dataset	Tools	Performance Metrics	Continuous Learning
[14]	SVM and Logistic Regression	No	VeReMi	PYTHON tools	F1-score	No
[16]	Binary classification with Naïve Bayes, decision tree and Random Forest	No	VeReMi	PYTHON tools	Accuracy	No
[17]	Random Forest and a posterior detection based on coresets	No	CICIDS2017	MATLAB	Accuracy	No
[18]	Hybrid optimization-based Deep Maxout Network (DMN)	No	BoT-IoT data and NSL-KDD data	PYTHON tools	Precision and Recall	No
[19]	Adaptive Neuro Fuzzy Inference System (ANFIS) and Convolutional Neural Networks (CNN)	No	CICIDS 2017	PYTHON tools	Precision Sensitivity Recall Specificity	No
[22]	ML methods for classification, KNN and RF	No	VeReMi	PYTHON tools	Accuracy F1-Score	No
[21]	SVM	No	Generated data	SUMO and OMNeT++ PYTHON tools	TPR, FPR, and ACC	No
[31]	Distributed multi-layer classifier	Yes	Generated data	OMNET++ SUMO	Accuracy	No
Our work	Incremental Online classification using Adaptive Random Forest (ARF) and K-Nearest Neighbors (KNN) classifiers	Yes	Generated data	NS-3 SUMO Python tools	Accuracy Recall Precision F1-score Training time Testing time	Yes

The comparison is conducted based on several parameters such as the real-time detection, the dataset used, the involved tools, the considered evaluation metrics, and the online learning approach.

The results of this comparison indicate that the current study may offer a novel strategy, involving incremental online learning for detecting routing attacks in VANETs.

First, a key benefit of our approach is the ability to monitor the behavior of nodes in the network in real-time. This allows identifying potential threats in time and acting appropriately.

Further, we build our own dataset for attack detection, which incorporates essential features, which are relevant in capturing the behavior of VANET nodes under black hole attacks.

Additionally, the adopted simulation methodology allows reliable VANET simulations built upon accurate mobility models that were generated from realistic maps using popular tools like OSM, SUMO, and NS-3.

Finally, our proposed approach allows continuous and real-time learning thanks to the incremental learning process, which allows for the use of data as it becomes available, creating models that are constantly up-to-date instead of models that are trained on fixed datasets. This suggests that it is capable of learning and improving its predictions or decisions as new data arrive.

5. Conclusions

The current study introduced a new approach for detecting routing attacks in VANETs, leveraging incremental online learning algorithms trained on data generated in real-time. Our study focused on assessing the performance of two algorithms, namely Adaptive Random Forest (ARF) and K-Nearest Neighbors, with regard to Accuracy, Precision, Recall, and F1-score metrics. The training and testing time of both classifiers were also analyzed. Our research particularly addressed the detection of black hole attacks, which pose a significant threat to the AODV routing protocol. Data used for attack detection are collected from simulating realistic VANET scenarios using two well-known simulators, namely SUMO and NS-3. Further, essential features, which are relevant in capturing the behavior of VANET nodes under a black hole attack, are monitored over time.

The findings show that incremental learning is a promising solution in time-critical applications like attack detection in highly dynamic environments such as VANETs, as it allows continuous and real-time learning. Further, the results show that Adaptive Random Forest (ARF) can be successfully applied to classify and detect black hole nodes in VANETs. ARF outperformed KNN with respect to all performance measures. However, ARF required more time for both training and testing in comparison with KNN.

While these conclusions have shown promising results, the detection of intrusions using online incremental machine learning remains a challenging problem due to the constantly evolving nature of attacks and the need to continuously adapt the models to new threats. It is worth noting that the choice of features used to represent the network traffic data has a significant impact on the performance of the classifier. Further, the performances of incremental learning classifiers seem to vary depending on various factors like the dataset used, the complexity of the problem, as well as the type of the incremental learning framework used.

As a next step, we intend to lead an in-depth study by tuning the algorithms' parameters in order to enhance their performance as well as the overall required time for training and testing.

Finally, it is crucial to acknowledge the challenges of real-time attack detection in VANETs, owing to the highly dynamic nature of the network and the rise of sophisticated attack strategies. Accordingly, our next priority is to implement a hybrid model that incorporates ARF with other well-known algorithms like online SVM for more effective and precise detection of potential threats and attacks in VANETs.

Author Contributions: Conceptualization, S.A. and S.E.H.; methodology, S.A. and M.-A.E.H.; validation, M.H.; formal analysis, S.A., M.H. and S.E.H.; investigation, S.A. and S.E.H.; resources, S.A. and S.E.H.; data curation, S.A. and S.E.H.; writing—original draft preparation, S.A., M.H. and S.E.H.; writing—review and editing, S.A., M.H. and S.E.H.; visualization, S.A., M.-A.E.H. and S.E.H.; supervision, M.H.; project administration, M.H. and S.E.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

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

References

1. Ajjaj, S.; El Houssaini, S.; Hain, M.; El Houssaini, M.-A. A New Multivariate Approach for Real Time Detection of Routing Security Attacks in VANETs. *Information* **2022**, *13*, 282. [CrossRef]
2. Banafshehvaragh, S.T.; Rahmani, A.M. Intrusion, Anomaly, and Attack Detection in Smart Vehicles. *Microprocess. Microsyst.* **2023**, *96*, 104726. [CrossRef]
3. Mchergui, A.; Moulahi, T.; Zeadally, S. Survey on Artificial Intelligence (AI) Techniques for Vehicular Ad-Hoc Networks (VANETs). *Veh. Commun.* **2022**, *34*, 100403. [CrossRef]
4. Nallaperuma, D.; Nawaratne, R.; Bandaragoda, T.; Adikari, A.; Nguyen, S.; Kempitiya, T.; De Silva, D.; Alahakoon, D.; Pothuhera, D. Online Incremental Machine Learning Platform for Big Data-Driven Smart Traffic Management. *IEEE Trans. Intell. Transp. Syst.* **2019**, *20*, 4679–4690. [CrossRef]
5. Losing, V.; Hammer, B.; Wersing, H. Incremental On-Line Learning: A Review and Comparison of State of the Art Algorithms. *Neurocomputing* **2018**, *275*, 1261–1274. [CrossRef]
6. López, J.M. Fast and Slow Machine Learning. Ph.D. Thesis, Université Paris-Saclay–Télécom Paristech, Paris, France, 2019.
7. Malik, A.; Khan, M.Z.; Faisal, M.; Khan, F.; Seo, J.-T. An Efficient Dynamic Solution for the Detection and Prevention of Black Hole Attack in VANETs. *Sensors* **2022**, *22*, 1897. [CrossRef]
8. Ajjaj, S.; El Houssaini, S.; Hain, M.; El Houssaini, M.-A. Performance Assessment and Modeling of Routing Protocol in Vehicular Ad Hoc Networks Using Statistical Design of Experiments Methodology: A Comprehensive Study. *ASI* **2022**, *5*, 19. [CrossRef]
9. Documentation-SUMO Documentation. Available online: https://sumo.dlr.de/docs/index.html (accessed on 21 September 2021).
10. Ns-3 | a Discrete-Event Network Simulator for Internet Systems. Available online: https://www.nsnam.org/ (accessed on 21 September 2021).
11. Gomes, H.M.; Bifet, A.; Read, J.; Barddal, J.P.; Enembreck, F.; Pfharinger, B.; Holmes, G.; Abdessalem, T. Adaptive Random Forests for Evolving Data Stream Classification. *Mach. Learn.* **2017**, *106*, 1469–1495. [CrossRef]
12. Montiel, J.; Jesse, R.; Bifet, A.; Talel, A. Scikit-Multiflow: A Multi-Output Streaming Framework. *J. Mach. Learn. Res.* **2018**, *19*, 2914–2915.
13. Karagiannis, D.; Argyriou, A. Jamming Attack Detection in a Pair of RF Communicating Vehicles Using Unsupervised Machine Learning. *Veh. Commun.* **2018**, *13*, 56–63. [CrossRef]
14. Singh, P.K.; Gupta, S.; Vashistha, R.; Nandi, S.K.; Nandi, S. Machine Learning Based Approach to Detect Position Falsification Attack in VANETs. In *Security and Privacy*; Nandi, S., Jinwala, D., Singh, V., Laxmi, V., Gaur, M.S., Faruki, P., Eds.; Springer: Singapore, 2019; Volume 939, pp. 166–178. ISBN 9789811375606.
15. Singh, P.K.; Gupta, R.R.; Nandi, S.K.; Nandi, S. Machine Learning Based Approach to Detect Wormhole Attack in VANETs. In *Web, Artificial Intelligence and Network Applications*; Barolli, L., Takizawa, M., Xhafa, F., Enokido, T., Eds.; Springer International Publishing: Cham, Switzerland, 2019; Volume 927, pp. 651–661. ISBN 978-3-030-15034-1.
16. Sonker, A.; Gupta, R.K. A New Procedure for Misbehavior Detection in Vehicular Ad-Hoc Networks Using Machine Learning. *Int. J. Electr. Comput. Eng. IJECE* **2021**, *11*, 2535. [CrossRef]
17. Bangui, H.; Ge, M.; Buhnova, B. A Hybrid Machine Learning Model for Intrusion Detection in VANET. *Computing* **2022**, *104*, 503–531. [CrossRef]
18. Kaur, G.; Kakkar, D. Hybrid Optimization Enabled Trust-Based Secure Routing with Deep Learning-Based Attack Detection in VANET. *Ad Hoc Netw.* **2022**, *136*, 102961. [CrossRef]
19. Karthiga, B.; Durairaj, D.; Nawaz, N.; Venkatasamy, T.K.; Ramasamy, G.; Hariharasudan, A. Intelligent Intrusion Detection System for VANET Using Machine Learning and Deep Learning Approaches. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 5069104. [CrossRef]
20. Sharma, A. Position Falsification Detection in VANET with Consecutive BSM Approach Using Machine Learning Algorithm. Ph.D. Thesis, Faculty of Graduate Studies through the School of Computer Science, Windsor, ON, Canada, 2021.
21. Zhang, C.; Chen, K.; Zeng, X.; Xue, X. Misbehavior Detection Based on Support Vector Machine and Dempster-Shafer Theory of Evidence in VANETs. *IEEE Access* **2018**, *6*, 59860–59870. [CrossRef]
22. Ercan, S.; Ayaida, M.; Messai, N. Misbehavior Detection for Position Falsification Attacks in VANETs Using Machine Learning. *IEEE Access* **2022**, *10*, 1893–1904. [CrossRef]
23. Rojas, J.S.; Rendon, A.; Corrales, J.C. Consumption Behavior Analysis of over the Top Services: Incremental Learning or Traditional Methods? *IEEE Access* **2019**, *7*, 136581–136591. [CrossRef]
24. Jin, B.; Jing, Z.; Zhao, H. Incremental and Decremental Extreme Learning Machine Based on Generalized Inverse. *IEEE Access* **2017**, *5*, 20852–20865. [CrossRef]
25. Almeida, A.; Brás, S.; Sargento, S.; Pinto, F.C. Time Series Big Data: A Survey on Data Stream Frameworks, Analysis and Algorithms. *J. Big Data* **2023**, *10*, 83. [CrossRef]
26. OpenStreetMap. Available online: https://www.openstreetmap.org/ (accessed on 4 September 2023).
27. Das, S.R.; Belding-Royer, E.M.; Perkins, C.E. Ad Hoc On-Demand Distance Vector (AODV) Routing. Available online: https://tools.ietf.org/html/rfc3561 (accessed on 20 December 2020).
28. Singh, D.; Singh, B. Investigating the Impact of Data Normalization on Classification Performance. *Appl. Soft Comput.* **2020**, *97*, 105524. [CrossRef]

29. Hidalgo, J.I.G.; Maciel, B.I.F.; Barros, R.S.M. Experimenting with Prequential Variations for Data Stream Learning Evaluation. *Comput. Intell.* **2019**, *35*, 670–692. [CrossRef]
30. AlQabbany, A.O.; Azmi, A.M. Measuring the Effectiveness of Adaptive Random Forest for Handling Concept Drift in Big Data Streams. *Entropy* **2021**, *23*, 859. [CrossRef] [PubMed]
31. Rashid, K.; Saeed, Y.; Ali, A.; Jamil, F.; Alkanhel, R.; Muthanna, A. An Adaptive Real-Time Malicious Node Detection Framework Using Machine Learning in Vehicular Ad-Hoc Networks (VANETs). *Sensors* **2023**, *23*, 2594. [CrossRef] [PubMed]