One-Class Classifier for Malicious Node Detection in Blockchain-Based IoT Networks

Hans Gabriel H. De Castro*, Gabriel Kenneth L. Mariñas†, Cedric Angelo M. Festin‡, and Wilson M. Tan§
Department of Computer Science, University of the Philippines Diliman
*hhdecastro@up.edu.ph, †glmarinas@up.edu.ph, ‡cmfestin@up.edu.ph, §wmtan@up.edu.ph

***Abstract*—Blockchain is being explored as a substitute for central servers in IoT networks to enhance security by mitigating the limitations of the traditional architecture. But blockchain-based IoT networks are still prone to malicious nodes, which are attacked nodes in the network that transmit inaccurate data. The field of malicious node detection in blockchain-based IoT networks remains an area that requires further exploration. Hence, in this paper, we propose a system that uses One-Class Classification (OCC) algorithms for malicious node detection in blockchain-based IoT networks. Our system includes a Sensor Retention Policy (SRP) that decides whether a sensor should be removed from the network through trust points mechanics. 3 different OCC algorithms were considered for the system: (1) One-Class Support Vector Machine (OCSVM), (2) OCSVM using Standard Gradient Descent (SGD-OCSVM), and (3) Local Outlier Factor (LOF). Different variations of the system were created by varying the decision threshold parameter of each algorithm (the values considered are 0.0, -0.15, and -0.30) and by also considering the system variation with no SRP. We devised a set of test runs to determine the performance of each system variation based on the following metrics: (1) Modified F-score, (2) Average Detection Time, (3) Average Processing Overhead, and (4) Memory Consumption. The test results show that using an OCSVM algorithm with a decision threshold of -0.30 provides the highest recorded modified F-score of 0.98. Compared to the other OCC algorithms, the OCSVM algorithm is the most efficient in terms of the modified F-score and average detection time trade-off. The OCSVM algorithm provides the lowest average processing overhead (1.6 ns) and the lowest memory consumption (27.46 KB) compared to other OCC algorithms.**

***Index Terms*—Blockchain, Internet of Things (IoT), Malicious node detection, One-class classification (OCC)**

I. Introduction

Many companies are eager to utilize the *Internet of Things* (IoT) in their products and workplaces as it improves operational efficiency and creates new connected products and services [1]. However, as with any popular technology, IoT became a clear target of many cyberattacks in the past years. Nearly 20% of organizations have recorded cyberattacks on their IoT devices in the previous decade [2] with each breach costing around $255 thousand for a $5 million annual revenue company and more than $20 million for a $2 billion annual revenue company [3]. The commonly used *central server architecture* for IoT network connections is one of the primary causes of the numerous cyberattacks related to IoT [4]. An IoT network with a central server is at risk of having a *single point of failure* (the server) which opens the possibility for *Distributed Denial-of-Service* (DDoS) attacks and *False Data Injection Attacks* (FDIA).

One way of resolving these vulnerabilities is by using *blockchain* as the IoT network's architecture. The decentralized and immutable property of blockchain prevents DDoS attacks and FDIA [5]. There have been works on implementing such a system. Ramesh et al [6] used a private Ethereum network with Swarm / IPFS as a decentralized database to safely store sensor data from IoT devices. Meanwhile, Özyılmaz and Yurdakul [7] improve upon the idea by grouping adjacent IoT nodes into clusters and establishing a gateway for each cluster. In their work, *only* the gateways are nodes in the blockchain network, the IoT devices only need to transmit their data to the respective gateway of their cluster.

Though the use of blockchain eliminates the problems mentioned earlier, the system is still at risk since the IoT devices remain vulnerable to tampering. A tampered device can send false or erroneous data to the database without being caught [8]. Formally, we call the tampered device a *malicious node* and the term for finding it is called *malicious node detection*. There is much research on the topic of malicious node detection in an IoT network [9], but most of them are implemented in a central server architecture, such as the system developed by Jaint et al [10] that utilizes a weighted trust mechanic where each IoT nodes have their own trust points that increase when their data is similar to their adjacent nodes and decrease if otherwise. An IoT node is removed from its cluster when its trust points fall below a certain threshold. They used *response time* and *detection ratio* to analyze the performance of their system. Similarly, Dandi et al [11] used a trust points system (they use the term *reputation* instead of *trust points*), but they utilize the statistics of data transmission of the IoT nodes for updating their reputation. They compute the *transmission delay*, *data forwarding rate*, and *data packet loss rate* of each IoT node to determine if they are trustworthy.

Few works implement malicious node detection in blockchain-based IoT networks. One example is the work by Babu et al [12] where they created their custom consensus that uses a binary classifier to detect malicious nodes and to decide which block to append to the blockchain. Each edge device manages a sensor cluster, aggregates sensor data into lightweight blocks, broadcasts them for validation by other devices' classifiers, and if the majority agree the block is not malicious, it is appended to the blockchain; otherwise, the cluster is deemed malicious and barred from network participation. But the problem with their work is that a binary classifier is not suitable for malicious node detection, since

a binary classifier requires as much malicious training data as non-malicious training data, which is difficult to achieve since most training data are non-malicious. A better solution is to use a *One-Class Classification* (OCC) algorithm to detect malicious nodes since it only requires data from non-malicious nodes to train its classifier.

In aid of the concerns raised in the previous work, we developed a system that uses OCC algorithms for malicious node detection in blockchain-based IoT networks. We cluster the sensors based on their geographic proximity and assign a gateway for each cluster, similar to [7]. The system also includes a *Sensor Retention Policy* (SRP) that decides whether a sensor should be removed from the network through a trust points mechanics similar to [10] and [11], but the condition for the increase/decrease of an IoT node's trust points depends on the evaluation of the OCC algorithm on the node's transmitted data.

The rest of the paper is organized as follows: Section II outlines the system design, Section III details the system simulation and testing methodologies for various variations, Section IV presents the test results and evaluates the performance of each variation using different metrics, and finally, Section V concludes the work and discusses potential future avenues of research.

II. System Design

A. System Architecture

The architecture of the proposed system is shown in Fig. 1. A group of *sensor clusters* and *gateways* forms the system. The *sensors* are clustered based on their geographical proximity. It is based on the assumption that the geographical proximity of sensors leads to the consistency of their data reading (this assumption is validated in the *Simulation* section of this paper). These sensors are IoT devices that are capable of getting measurements from their physical environments such as temperature and relative humidity. They send their data to their cluster's assigned gateway. The gateway is a computer that has two primary components: an *Ethereum client* and a *Malicious Node Detection Program* (MNDP). The Ethereum client allows the gateway to store validated data from its sensor cluster to the blockchain. It is also used to read data from the blockchain. Before the sensor data gets stored in the blockchain, it must first pass through the MNDP. The MNDP itself has two components: *OCC Models Array* and a *Sensor Retention Policy* (SRP). The OCC Models Array contains twelve OCC models, each trained from each month of the year before the system is run. This means that, if the system will run on January 2012, then the OCC Models Array has a model trained from January 2011 data, a model trained from February 2011 data, and so on until December 2011. We have implemented the OCC Models Array this way so that the MNDP can adapt its evaluation based on seasonal changes that are already observable between months of the year. Our decision of partitioning our training data in monthly intervals strikes a rational balance between model accuracy (since the MNDP can adapt to seasonal changes) and storage efficiency

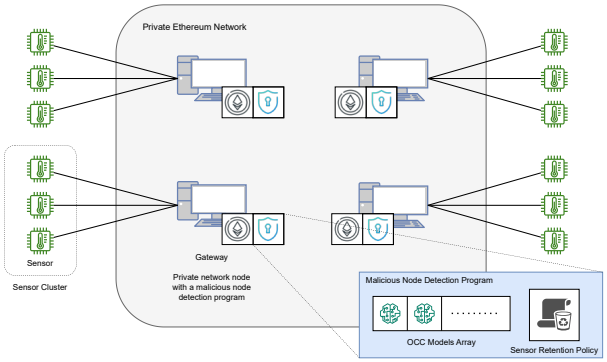

Fig. 1. System Architecture Diagram

(since partitioning the training data further will consume more storage from the gateway, e.g., if the training data is weekly partitioned, then the gateway must always store 48 trained OCC models in its storage drive). The sensor data is fed to one of the models in the OCC Models Array to determine if the data is malicious or not. If it is evaluated to be non-malicious, then the data will be stored in the blockchain using the gateway's Ethereum client. Otherwise, if the data is evaluated to be malicious, the sensor will be assessed by the SRP if it is legitimately malicious using a *trust points mechanic*. Based on the assessment of the SRP, the sensor that has sent the malicious data could be removed from its cluster.

In the succeeding sections, we will discuss in greater detail the Sensor Retention Policy and how a sensor data flows within the system.

B. Sensor Retention Policy (SRP)

Since the classifier models are susceptible to errors during their training, it would be unwise to immediately remove a sensor once one of its data entries is evaluated to be malicious by a classifier model. It can misclassify non-malicious data as malicious. Hence, the SRP aims to address this issue by implementing a *trust points system* in the cluster. A trust point is a property of a sensor in a cluster that is assigned and monitored by the SRP. This property determines the likelihood of a sensor being malicious. A newly-registered sensor in a cluster has k initial trust points. k would be chosen such that it is low enough to quickly eliminate malicious sensors from a cluster but not too low that one-off non-malicious sensors are immediately removed from the cluster without giving any chance to increase their trust points. The trust points of a sensor get incremented after its five consecutive data entries is evaluated to be non-malicious. The trust points of a sensor get decremented every time one of its data entries are evaluated to be malicious. If a sensor's trust points reach 0, the SRP will send a message to the gateway to remove the sensor from the cluster.

C. Flow of Sensor Data

From the system architecture in Fig. 1, it was already shown that data from the sensor cluster are stored in the blockchain through the cluster's assigned gateway. However, not all data sent by the sensors are stored in the blockchain. Mechanisms were created to handle malicious sensors in a cluster as explained in the *Sensor Retention Policy* section. Algorithm (1) shows the flow of a single sensor data. The algorithm receives a single sensor data and its output depends on the evaluation of the SRP – it will either store the sensor data in the blockchain or it will drop the data, and it will also remove malicious sensors from the cluster.

Line 1 sets variable m to be the maximum possible trust points of a sensor. This is needed since, without it, a sensor that sends non-malicious data for a long period will be difficult to check for maliciousness since its trust points have heavily accumulated during that period. Line 2 and Line 3 are clear descriptions of what happens in Fig. 1. Line 4 and Line 9 show the different strategies of the SRP based on the evaluation of the classifier. If the classifier evaluates the data to be non-malicious (Line 4), Lines 5 to 8 store the data in the blockchain and increment the sensor's trust points if it meets the condition in Line 6. Otherwise, if the classifier evaluates the data to be malicious (Line 9), Lines 10 to 14 decrement the sensor's trust points and remove the sensor from the cluster if its trust points become zero.

Algorithm 1 Processing of a single sensor data

Input: Single sensor data
Output: Either the data is stored to the blockchain or the data is dropped.

1: $m \leftarrow 15$; ▷ Maximum trust points of a sensor.
2: Sensor sends data to gateway;
3: Gateway feeds data to classifier;
4: **if** the classifier evaluates the data to be not malicious **then**
5: Store the data to the blockchain;
6: **if** it is the fifth consecutive non-malicious data sent by the sensor and sensor's trust points $\leq m$ **then**
7: Increment the trust points of sensor by 1
8: **end if**
9: **else**
10: Decrement trust points of sensor by 1
11: **if** trust point of sensor == 0 **then**
12: Remove sensor from the cluster;
13: **end if**
14: **end if**

III. Methodology

A. Simulation Data

For the simulation of our proposed system, we need both non-malicious and malicious data. The non-malicious data is used to train our OCC models and both the non-malicious and malicious data were used to test the proposed system.

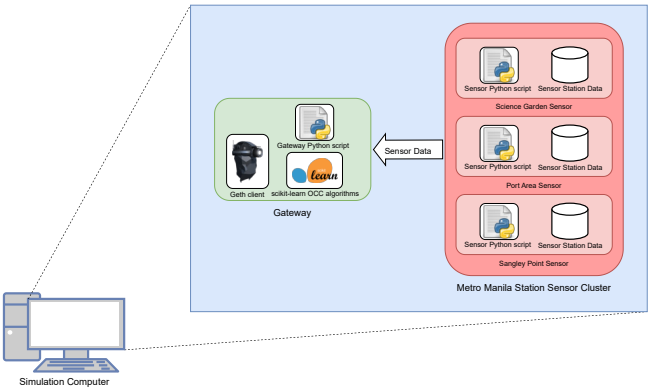

Fig. 2. Simulation Framework

1) Non-malicious Data: To simulate IoT devices sending sensor readings into the blockchain network, climatic data were obtained from three sensor stations of *The Philippine Atmospheric, Geophysical, and Astronomical Services Administration* (PAGASA). The three sensor stations chosen due to their geographical proximity are Science Garden station, Port Area station, and Sangley Point station. Daily weather data from January 1, 2011, to December 31, 2021 were recorded from each station. After performing some pre-processing in their dataset (such as handling invalid entries and removing features with high variance), the features that will be used in the simulation are *maximum temperature* (in °C), *minimum temperature* (in °C), *mean temperature* (in °C), *relative humidity* (in %), and *wind speed* (in m/s). We will allocate a part of the non-malicious data for model training and the rest for model testing.

2) Malicious Data: To simulate the IoT devices getting attacked by a perpetrator, we would need the malicious test data that the attacked device will transmit to the gateway. In our case, the malicious test data is generated by adding a random offset to the boundary values of the non-malicious test data. Specifically, a non-malicious test data entry d_i for date i generates the malicious test data entry d_i^* using the equation

$$d_i^* = d_i + \text{choice}(-1, 1) * \text{choice}(\min(d), \max(d)) + \text{choice}(-1, 1) * \text{randint}(1.0, 50.0) \quad (1)$$

where choice() and randint() corresponds to numpy's functions of similar name and d is the entire test dataset of a sensor station.

B. Simulation Framework

Fig. 2 shows a diagram of our simulation framework. We aim to simulate our proposed system (as previously described in the *System Architecture* section) by running a private Ethereum network with only a single cluster through the localhost of a Windows 10 Desktop PC with Intel Core I5-10400 Processor, 16 GB of RAM, and Windows 10 OS

installed on a 256 GB SSD. To set up the private Ethereum network, we created the initial block of the blockchain (also known as the *genesis block*) using a genesis file that serves as a blueprint for the network's configuration.

As mentioned in the *System Architecture* section, the signer of the private network will be the system's gateway. This gateway will form a cluster with three sensors, which we simulate using an Ethereum client and a Python process running on the simulation machine. We run a script to simulate each sensor station in the Metro Manila cluster. The sensor script performs the following tasks: it stores its station's daily sensor data and passes its data entry to the gateway as a dictionary object. We run a different script to simulate the cluster's gateway. The gateway script performs the following tasks: it implements the Malicious Node Detection Program by having an array of classifiers that classify the maliciousness of received sensor data using an OCC algorithm from `scikit-learn` [13] (a machine learning library in Python) and by implementing a Sensor Retention Policy as described in Algorithm 1. The gateway script also performs the storage/retrieval of non-malicious data to/from the simulation's Ethereum private blockchain by using the *web3.py* library [14] to interface with the running Ethereum client *geth*.

C. Metrics

To assess the feasibility and capability of our proposed system, it is valuable to measure four metrics: *Modified F-score*, *average detection time*, *average processing overhead*, and *memory consumption*.

1) Modified F-score: The modified F-score metric is defined as

$$F_{1,new} = \frac{NK + tp_{total} + tn_{total} - fp_{total} - fn_{total}}{2NK} \ , \quad (2)$$

where N is the number of test cases in a test suite, K is the number of sensors in a cluster, tp_{total} is the total number of true positives from the execution of a test suite, and tn_{total}, fp_{total}, fn_{total} are defined similarly to tp_{total}. We receive a true positive if a sensor is not scheduled to be attacked and it remains in the cluster until the execution of the test case ends (the definition of a test case is defined in the Tests section), we receive a true negative if a sensor is scheduled to be attacked and it is removed from the cluster before the execution of the test case ends, we receive a false positive if a sensor is scheduled to be attacked and it remains in the cluster until the execution of the test case ends, and we receive a false negative if a sensor is not scheduled to be attacked and it is removed from the cluster before the execution of the test case ends or if a sensor is removed from the cluster before its attack date.

The modified F-score metric has a range of $[0.0, 1.0]$. A system is considered highly accurate in detecting malicious sensors when its modified F-score approaches 1.0 and considered highly inaccurate when its modified F-score approaches 0.0.

We modified the F-score metric F_1 since it cannot reliably measure the system's accuracy when the system tends to remove its sensors immediately, wherein it gives a high F-score despite performing poorly. Because of this, the F-score definition must be modified so that its computation is no longer dependent on the accuracy of classifying individual data entries of its sensors, but rather on the accuracy of the system in maintaining/removing its sensors.

We will use the modified F-score metric to determine which of our three candidate OCC algorithms – *One-class SVM* (OCSVM), *One-class SVM with Stochastic Gradient Descent* (OCSVM-SGD), and *Local Outlier Factor* (LOF) – will provide the highest accuracy for our system.

2) Average Detection Time: The average detection time metric measures how quickly can the system detects and remove malicious nodes in a cluster.

Detection time t_D is measured as:

$$t_D = \text{date of detection} - \text{date of attack} \ , \quad (3)$$

and average detection time $\bar{t}_D$ is measured as:

$$\bar{t}_D = \frac{\Sigma_{i=1}^{n} t_{D,i}}{n} \ , \quad (4)$$

where n is the total number of scenarios of detected attacked sensors. Because the entries in the dataset are recorded at daily intervals, the measurement of the average detection time will also be in days.

3) Average Processing Overhead: The average processing overhead metric measures the average additional time expended by incorporating the MNDP into the blockchain-based IoT network. The average processing overhead t_{apo} has the equation:

$$t_{apo} = \bar{t}_{\text{MNDP}} - \bar{t}_{\neg\text{MNDP}} \ , \quad (5)$$

where $\bar{t}_{\text{MNDP}}$ and $\bar{t}_{\neg\text{MNDP}}$ are the average time it takes to process a single data with MNDP and without MNDP, respectively. We compute both values in a similar manner by determining the time duration between these two events common in both system variations: when the gateway receives the data entry from the sensor, and when the gateway receives the transaction hash of the storing of the data entry to the blockchain.

4) Average Memory Consumption: The average memory consumption metric measures the amount of memory the OCC models array will consume once all of the twelve classifier models are trained. We can measure the memory consumption of the list of OCC models during the execution of the gateway's Python process using the `Pympler` memory profiling library.

D. Tests

To measure the four metrics discussed earlier, we will use our simulation framework to conduct numerous tests on different variations of our proposed system. By monitoring and analyzing the four metrics during the testing phase, we will identify the system variation that performs the best. Table I

TABLE I
SYSTEM VARIATIONS

with MNDP?	with SRP?	OCC Algorithm used	decision threshold (φ)	Label
Yes	Yes	OCSVM	-0.30	OCSVM30
Yes	Yes	OCSVM	-0.15	OCSVM15
Yes	Yes	OCSVM	0	OCSVM0
Yes	Yes	SGD-OCSVM	-0.30	SGD30
Yes	Yes	SGD-OCSVM	-0.15	SGD15
Yes	Yes	SGD-OCSVM	0	SGD0
Yes	Yes	LOF	-0.30	LOF30
Yes	Yes	LOF	-0.15	LOF15
Yes	Yes	LOF	0	LOF0
Yes	No	N/A	N/A	noSRP
No	No	N/A	N/A	noMNDP

shows the different variations of our system with an accompanied label for ease of identification. The first nine variations in the table are *with-SRP* variations. They are the result of combining three different OCC algorithms (OCSVM, SGD-OCSVM, and LOF) with three different *decision threshold* φ values ($\varphi = 0.0$, $\varphi = -0.15$, and $\varphi = -0.30$). The decision threshold is an OCC algorithm parameter often set to 0.0, which decides whether the MNDP classifies a data entry as malicious if the *decision function score* it receives from the classifier is greater than the decision threshold. But other than the decision threshold, we maintain initial trust points $itp = 15$ and maximum trust points $mtp = 30$ for all with-SRP variations. Meanwhile, the remaining two variations were versions of the system with no SRP and with no MNDP. We will compute the four metrics of the with-SRP variations and we will use the processing speed of *noMNDP* as the base processing speed in computing the processing overhead metric of the other variations.

To test the with-SRP variations, a collection of test cases called *test suite* was generated. Each test case is a complete run of a variation of the simulation framework from January 1, 2011 to December 31, 2021. The test cases differ by the number of attacked sensors, the date of their attack and the duration of their attack. Each with-SRP variations will undergo testing against this suite, enabling us to measure their performance on the four metrics. Table II shows some of the test cases in the test suite. The test suite comprises 100 test cases, with 10 cases having no attacked sensors, 30 cases having 1 attacked sensor, another 30 cases having 2 attacked sensors, and the remaining 30 cases having 3 attacked sensors. We generate each instance of sensor attack by randomly choosing their date of attack and their duration of attack. We prevent the date of attack to occur during the training period.

Note that each system variation undergoes only one test suite run, and we collect the metric data based on the outcome of the test suite. The modified F-score metric $F_{1,new}$ is computed by:

1) For each sensor in each test case of a test suite, we determine the result of their run if it is true positive, true negative, false positive, or false positive based on the criteria defined in the *Metrics* section
2) Sum up the gathered run results to get tp_{total}, tn_{total}, fp_{total}, and fn_{total}
3) Use Equation (2) to compute the modified F-score where we set $N = 100$ and $K = 3$

Note that we can perform step 1 since we know the attack date of each sensor (given that the sensor is scheduled to attack) for each test case in a test suite.

For the average detection time metric $\bar{t}_D$, we can compute it by:

1) Counting all scenarios of detected attacked sensors and noting their date of detection and their date of attack
2) Using Equation (3) to compute the detection time t_D of each detected scenario
3) Using Equation (4) to compute the average detection time $\bar{t}_D$

Note that we do not consider in the computation of average detection time the scenario where the MNDP does not detect the attacked sensor since the equation for detection time requires the date of detection.

For the average processing overhead metric t_{apo}, we need to compute the average single-data processing time of a system with MNDP ($\bar{t}_{\text{MNDP}}$) and the average single-data processing time of a system without MNDP ($\bar{t}_{\neg\text{MNDP}}$) and use Equation (5) to compute it. To get $\bar{t}_{\text{MNDP}}$, we compute the average t_{MNDP} from all data entries processed within the execution of the test suite on a system with MNDP. The same thing can be done to compute $\bar{t}_{\neg\text{MNDP}}$ for a system without MNDP. We will use the *noMNDP* system to compute $\bar{t}_{\neg\text{MNDP}}$, and use this for the computation of t_{apo} for the rest of the system variations (since they all have an MNDP).

Finally, to determine the memory consumption metric, we need to calculate the memory consumption of the `models` list object when `len(models) == 12`, i.e., when all the 12 OCC models are trained. We do this for each test case and find their average.

IV. RESULTS AND ANALYSIS

To facilitate the identification of each system variation, the graphs presented below will utilize the labels provided in Table I.

TABLE II
TEST SUITE SNIPPET (ATTACK PERIOD IS IN *days* UNIT)

	Port Area Station		Sangley Point Station		Science Garden Station	
Test Case	**Attack Date**	**Attack Period**	**Attack Date**	**Attack Period**	**Attack Date**	**Attack Period**
1	None	None	None	None	None	None
...	...	...	...	...	...	...
98	Jan 05, 2015	1420	May 10, 2021	88	Jan 14, 2016	1715
99	Dec 8, 2019	731	Feb 17, 2019	831	Jun 22, 2015	1835
100	May 24, 2021	164	Oct 25, 2020	231	Mar 24, 2012	2950

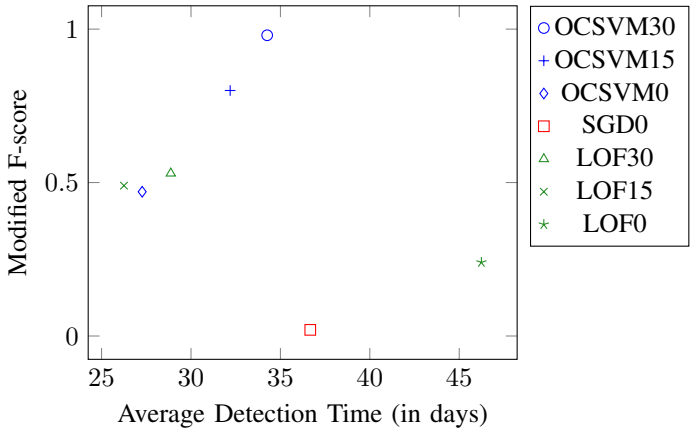

Fig. 3. Scatter plot showing the relationship between the average detection time metric and the modified f-score metric of the different system variations

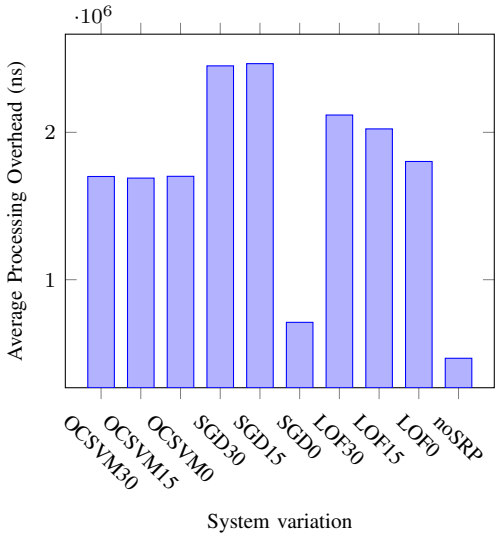

Fig. 4. The average processing overhead of the different system variations

A. Modified F-score and Average Detection Time

The modified F-score metric and the average detection time metric are both influenced by the *decision threshold* parameter (φ), so in Fig. 3 we show the relationship between the two metrics using a scatter plot. Note that *SGD15*, *SGD30*, and *noSRP* are not included in the graph since they are unable to perform malicious node detection and, thus, fails their average detection time metric. *SGD0* is the only SGD-OCSVM-based system variation with a recorded average detection time. It may seem to imply that *not* reducing φ for an SGD-OCSVM-based system leads to better performance in detecting malicious nodes. But as we can observe from Fig. 3, *SGD0* has the lowest modified F-score among all the different system variations, so whether we reduce its φ parameter or not, its performance in detecting malicious nodes is poor compared to other system variations that use different OCC algorithms. In the context of the LOF-based system variations, we can observe two patterns when we reduce φ: firstly, the modified F-score metric exhibits an increase, and secondly, the average detection time undergoes a substantial decrease while exhibiting a gradual increase with further reduction of φ. It means, for an LOF-based system, reducing the φ parameter would be beneficial only up to a certain point since further reduction leads to a trade-off between the two metrics since the modified F-score increases at the expense of the average detection time from increasing. A similar pattern is observed with OCSVM-based system variations, wherein reducing the φ parameter leads to an increase in both metrics. But since an OCSVM-based system can reach a near 1.0 modified F-score at a relatively minimal increase in average detection time just by reducing φ to -0.30, we can assert that among all the OCC algorithms, OCSVM obtains the highest benefit in varying the φ parameter in terms of the modified F-score metric and the average detection time metric.

B. Average Processing Overhead

Fig. 4 compares the average processing overhead of all the system variations listed in Table I. Besides *NoSRP*, *SGD0* has the lowest average processing overhead. But since *SGD0* also has the lowest modified F-score among all the variations, the authenticity of its metric result is up for debate. Hence, ignoring *SGD0*, the OCSVM variations has the lowest average processing overhead among other variations of 1.6 ns, with the LOF and the SGD-OCSVM variations trailing behind.

C. Average Memory Consumption

Fig. 5 shows the average memory consumption of the generated OCC Models Array of each OCC algorithm. The SGD-OCSVM algorithm has the highest average memory consumption among all the OCC algorithms with 883.86 KB of consumed memory, the LOF algorithm trails behind with an average memory consumption of 268.48 KB, while the OCSVM algorithm has the lowest average memory consumption among all the OCC algorithms with 27.47 KB of

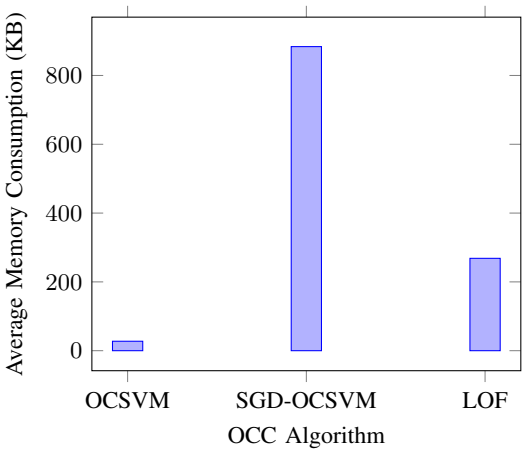

Fig. 5. The average memory consumption of the different OCC Algorithms

consumed memory, which is 32 times *less* than the SGD-OCSVM algorithm. Hence, in terms of having a lower memory footprint, the OCSVM algorithm has a significant advantage over the other OCC algorithms.

V. Conclusion and Future Work

In this paper, we tackle the malicious node problem in blockchain-based IoT networks by implementing a one-class classifier-based malicious node detection program. This program can serve as a protection layer for a blockchain-based IoT network to ensure the validity of the data being stored in the blockchain. We tested 3 different OCC algorithms and different decision threshold parameter values on our system. We found that the OCSVM algorithm with $\varphi = -0.30$ provides the highest modified F-score of 0.98 and compared to other OCC algorithms, the OCSVM algorithm provides the most efficient trade-off between the modified F-score and the average detection time metric. We also found that the OCSVM algorithm provides the lowest average processing overhead (1.6 ns average of all OCSVM variations) and the lowest memory consumption (27.46 KB) compared to other OCC algorithms. All of these findings provides us with the conclusion that the OCSVM algorithm outperforms the other OCC algorithms in terms of our target metrics and in integration with the system, provides a practical approach to malicious node detection in blockchain-based IoT networks.

For future work, we recommend to use a dataset with finer sample intervals to provide more practical detection time results. It would also be worthwhile to explore the possible benefits of scheduled and/or on-demand retraining of the models in the OCC Models Array. Since we want the system to run indefinitely, we need to guarantee the accuracy of the trained models even after running the system for a long period. To do this, we can consider two new mechanisms. First, we can refresh the models in the OCC Models Array by retraining a new model every start of a month. The second mechanism handles the case when the sensors in the cluster are affected by a legitimate spike/dip in their readings caused by natural phenomena such as heat waves or typhoons. We expect the system to detect these sudden changes and not just assume that their emitted data are malicious. To do so, we can introduce a new condition in the SRP that triggers retraining of all the models in the OCC Models Array when the majority of the sensors in the cluster is evaluated to be malicious even though, upon manual investigation, none of them are legitimately malicious.

References

[1] "Iot spotlight report 2020," Vodafone, Tech. Rep., 2020. [Online]. Available: https://explore.vodafone.com/iot-spotlight-report-2020

[2] "Leading the iot: Gartner insights on how to lead in a connected world," Gartner, Tech. Rep., 2017. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[3] "Are your company's iot devices secure? internet of things breaches are common, costly for u.s firms," Altman Vilandrie & Company, Tech. Rep., 2017. [Online]. Available: https://www.altmansolon.com/wp-content/uploads/2020/07/IoT-Security-Whitepaper.pdf

[4] A. Al Sadawi, M. S. Hassan, and M. Ndiaye, "A review on the integration of blockchain and iot," in *2020 International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, 2021, pp. 1–6.

[5] I. Romashkova, M. Komarov, and A. Ometov, "Demystifying blockchain technology for resource-constrained iot devices: Parameters, challenges and future perspective," *IEEE Access*, vol. 9, pp. 129 264–129 277, 2021.

[6] V. K. C. Ramesh, Y. Kim, and J.-Y. Jo, "Secure iot data management in a private ethereum blockchain," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 369–375.

[7] K. R. Ozyilmaz and A. Yurdakul, "Designing a blockchain-based iot with ethereum, swarm, and lora: The software solution to create high availability with minimal security risks," *IEEE Consumer Electronics Magazine*, vol. 8, no. 2, pp. 28–34, 2019.

[8] G. S. Ramachandran and B. Krishnamachari, "Blockchain for the iot: Opportunities and challenges," *CoRR*, vol. abs/1805.02818, 2018. [Online]. Available: http://arxiv.org/abs/1805.02818

[9] L. K. Ramasamy, F. Khan K. P., A. L. Imoize, J. O. Ogbebor, S. Kadry, and S. Rho, "Blockchain-based wireless sensor networks for malicious node detection: A survey," *IEEE Access*, vol. 9, pp. 128 765–128 785, 2021.

[10] B. Jaint, V. Singh, L. K. Tanwar, S. Indu, and N. Pandey, "An efficient weighted trust method for malicious node detection in clustered wireless sensor networks," in *2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, 2018, pp. 1183–1187.

[11] W. Dandi and H. Jian, "Blockchain-based node data detection scheme for the internet of things system," in *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*, 2021, pp. 206–209.

[12] A. T., S. Babu, and B. S. Manoj, "A machine learning consensus based light-weight blockchain architecture for internet of things," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2022, pp. 1–6.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] "web3.py docs." [Online]. Available: https://web3py.readthedocs.io/en/stable/