A Real-time Anomaly-based Intrusion Detection System for Automotive Controller Area Networks

Luis F. P. D'Andrada[1], **Paulo Freitas de Araujo-Filho**[1], **and Divanilson R. Campelo**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

{lfpd,pfaf,dcampelo}@cin.ufpe.br

Abstract. *The Controller Area Network (CAN) is the most pervasive in-vehicle network technology in cars. However, since CAN was designed with no security concerns, solutions to mitigate cyber attacks on CAN networks have been proposed. Prior works have shown that detecting anomalies in the CAN network traffic is a promising solution for increasing vehicle security. One of the main challenges in preventing a malicious CAN frame transmission is to be able to detect the anomaly before the end of the frame. This paper presents a real-time anomaly-based Intrusion Detection System (IDS) capable of meeting this deadline by using the Isolation Forest detection algorithm implemented in a hardware description language. A true positive rate higher than 99% is achieved in test scenarios. The system requires less than 1μs to evaluate a frame's payload, thus being able to detect the anomaly before the end of the frame.*

1. Introduction

A modern automobile is comprised of dozens of Electronic Control Units (ECUs), which are responsible for controlling many electrical systems in a car, such as headlamps or even the steering. These ECUs are usually connected by in-vehicle networks, such as the Controller Area Network (CAN), the most pervasive intravehicular technology. CAN is a multi-master serial bus developed by Bosch in the 1980s and later standardized by the ISO (International Organization for Standardization). A CAN system covers both physical and data link layers of the ISO/OSI model. The physical layer defines how signals are transmitted, describing: the medium, i.e. if it is based on one or two wires; the Bit Encoding, i.e. the voltages CAN High and CAN Low; and the Bit Timing, i.e. when to sample during the bit. The data link layer is responsible for all other CAN characteristics such as frame formats, bit stuffing and arbitration. Since CAN is multi-master, all nodes can initiate a transmission at a given time. In order to decide which node has the priority, CAN takes advantage of its physical properties, giving higher priorities to the transmission of frames with lower ID (identifier) values. In this way, certain messages can be designed to have priority over others [Di Natale et al. 2012].

CAN has been the standard for intravehicular networks since its adoption. Back then, there were not security concerns in cars, since the vehicle was completely isolated, with no connection to the outside world. However, with the increasing connectivity present in modern automobiles, such as Bluetooth, WiFi and 4G/5G [Checkoway et al. 2011] [Miller and Valasek 2014], remote attacks on cars were proven to be possible [Miller and Valasek 2015] and thus the need to improve security in in-vehicle networks has become obvious.

[Liu et al. 2017] mention five attacking methodologies for in-vehicle networks that have been proved to be effective in the literature: 1) frame sniffing; 2) frame falsifying; 3) frame injection; 4) replay attack; and 5) denial of service (DoS). Frame sniffing, which is the foundation of other attacks, consists of observing and recording the traffic on a CAN bus in order to learn the details of CAN frames. Additionally, an attacker can send frames with randomized data in order to observe and gain knowledge about the system behavior. Having learned about system functions, the attacker can send frames designed to perform a specific attack. These frames are valid to the system, but they use fake data that can be misleading, characterizing a frame falsifying (or spoofing). Frame injection consists of injecting frames into the network through a malicious node, which could be a laptop connecting to the OBD port, a reprogrammed ECU or even a telematics system infected by a malware. A replay attack is simpler than the previously mentioned attacks, and consists of sending valid frames previously captured by sniffing at the appropriate time. Although it is a simple attack to execute, it can lead to non-negligible threats. Lastly, the DoS attack takes advantage of the CAN priority scheme in order to prevent genuine nodes to transmit their frames.

Intrusion Detection Systems (IDS) have been suggested as a mean of increasing in-vehicle network security [Miller and Valasek 2014] [Liu et al. 2017]. This approach is well suited to automotive CAN networks because it does not require the redesign of the system and it can run well in a scenario with limited resources and in real time. Other solutions, such as adding cryptographic features, would require a major redesign of the existing system and also would add network overhead. These characteristics make IDS systems a cost-effective and efficient way to increase in-vehicle network security.

[Dupont et al. 2019] classify NIDS (Network-based IDS) regarding the detection method (knowledge-based, anomaly-based and specification-based) and depth of inspection (flow-based inspection and payload-based inspection). A knowledge-based IDS, also called signature-based, compares attack signatures with observed events in order to identify attacks. This approach can achieve high detection rates in known attacks, but performs badly when detecting unknown attacks. On the other hand, an anomaly-based IDS builds a model from normal data and treats frames that deviate from it as anomalies, thus being able to detect unknown attacks with a higher precision. A specification-based IDS uses systems specifications instead of normal data in order to build a reference model. Regarding the depth of inspection, flow-based inspection takes advantage of messages' frequency in order to build a pattern for every CAN ID so that any flow that deviates from the normal pattern would be suspicious. A payload-based approach analyzes the packet's content, comparing it to a previously built model, which can be created using historical information or attack signatures, for example.

An IDS can detect attacks, but it is not capable of preventing it. [Abbott-McCune and Shay 2016] and [Giannopoulos et al. 2017] suggest inserting a bit stuffing error during malicious frame transmission in order to prevent it from being successfully transmitted throughout the network. The IDS must respond fast enough in order to generate the bit stuffing error while the malicious frame is still being transmitted,

The purpose of this paper is to present an IDS capable of detecting anomalies in an automotive CAN network before the end of the transmission of a CAN frame. The proposed anomaly-based IDS is also payload-based, since it detects anomalous behaviors

by inspecting the content of the CAN frame. In order to detect anomalies, the Isolation Forest algorithm was chosen because of its low execution time, low memory requirements and simple implementation in a hardware description language. Our experiments show that the IDS is able to evaluate a CAN frame within 280 ns, in a worst case scenario. As a result, the anomaly can be detected before the end of the frame.

The remainder of this paper is described as follows. Section 2 presents the works related to this paper. Section 3 presents the IDS and its main characteristics. In Section 4, the evaluation scenario is described along with the main results obtained. Lastly, in Section 5 the conclusion and future works are discussed.

2. Related Works

There are several papers that propose IDS systems for CAN, but to the best of our knowledge none presents a real-time anomaly-based IDS that meets a time constraint in the order of few microseconds. This time constraint is necessary for the system to be able to prevent anomalous frames from being spread across the network, as will be discussed in Section 3.1.

[Kang and Kang 2016] propose an intrusion detection technique using a deep neural network (DNN). It achieves 99.9% of accuracy rate in the evaluated scenario and can respond in real time. However, since the time constraint is in the order of milliseconds, the solution is unable to respond before the end of a frame. In addition, deep neural networks demands high computational power, which can be unattractive for the automotive industry.

[Taylor et al. 2016] propose using a Long Short-Term Memory (LSTM) recurrent neural network (RNN). For each CAN ID, the RNN is trained to predict the payload of the next frame. The solution detects anomalies in each ID's individual frame stream independently. For most IDs and types of attacks tested, the solution of [Taylor et al. 2016] achieves an area under the ROC (receiver operating characteristic) curve greater than 0.99.

[Koyama et al. 2019] propose an anomaly detection based on messages' intervals and payloads, achieving 99.99% average accuracy across the four test vehicles. Since the proposed method detects anomalies in a tuple of three messages, it is unable to detect which message is anomalous. The response time in the test environment averaged 450μs, thus being unable to meet the required deadline.

Table 1. Comparison between papers

Papers	Data Used	Model	Single frame detection time
[Kang and Kang 2016]	CAN ID + Payload	Learned	$>$ 1ms
[Koyama et al. 2019]	Timing of frames + Payload	Learned	Not applicable
[Seo et al. 2018]	CAN ID + Payload	Learned	Not available
[Abbott-McCune and Shay 2016]	CAN ID	Specified	$\leq 10^{-6}$s
[Taylor et al. 2016]	Payload	Learned	Not available

[Seo et al. 2018] propose GIDS (GAN-based Intrusion Detection System), which uses the deep-learning model Generative Adversarial Nets. It builds two models, one that is trained for known attacks and another that is trained for unknown attacks. In order to train the models, it uses data captured from a Hyundai's YF Sonata using a raspberry pi 3 connected to a CAN bus through the OBD-II port. GIDS showed average accuracy of 100% for the first model and 98% for the second model. It is stated that this solution can be a real-time intrusion detection for the in-vehicle network, but it does not show any proof and no time constraint is defined.

[Abbott-McCune and Shay 2016] propose a solution capable of evolving to a working IPS [Abbott-McCune and Shay 2016]. It consists of checking whether the arbitration ID of a CAN frame matches the ones pre-programmed in the ECU and also if the ECU is transmitting data at the same time, thus detecting if a malicious node is trying to replay a message. It also detects whether the ECU is transmitting an ID that it is not pre-programmed to. In order to verify these inconsistencies, it requires one additional module per CAN node. Since it consists in a simple logic operation, the solution identifies these inconsistencies as soon as the arbitration field is sent. Although this solution responds fast enough to prevent a malicious frame transmission through the network, it cannot identify attacks based on the messages' content, thus not being able to detect attacks where the compromised ECU is the one supposed to send the malicious message ID.

Table 1 compares the aforementioned papers regarding the data used for their solution, whether the model is learned from a data base, or specified using system specifications. Also, it shows that many authors do not mention their solution's single frame detection time, which is an important information in order to act when an attack is detected.

3. Real-time Anomaly-based IDS for CAN

The purpose of the real-time anomaly-based IDS for CAN proposed in this paper is to be able to prevent anomalous frames to be successfully transmitted on the network. For this, a time constraint in the order of microseconds must be met. The proposed solution combined with another module that injects errors on a CAN network would form an Intrusion Prevention System (IPS), actively preventing the detected anomalies to be spread across the network. Such a module has already been proposed in solutions like CAN Stomping [Giannopoulos et al. 2017] and error injection techniques [Freitas de Araujo-Filho 2018]. Regarding the classification made in Table 1, the solution proposed in this paper uses data from payload, a learned model and a single frame detection time lower than $1\mu s$.

[Freitas de Araujo-Filho 2018] compares the performance of selected anomaly detection algorithms over a CAN dataset, and concludes that the Isolation Forest algorithm performs very well in this problem, achieving more than 99% of precision rate. As the Isolation Forest algorithm uses decision trees to detect the anomalies, it is possible to export a trained model to an FPGA (Field-programmable gate array) using a methodology similar to the one proposed in [Struharik 2011]. The methodology in [Struharik 2011] generates trees that have an upper bound for the execution time. For this solution, the upper bound is hT, where h is the tree height and T is the clock period for the FPGA.

An FPGA solution is chosen in order to meet the required deadline, but a CPU ap-

Field	Size(bits)	Description
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	CAN message identifier
RTR (Remote Request)	1	Dominant (0) for data frames and recessive (1) for remote request frames
IDE	1	Must be dominant (0)
r0	1	Reserved bit
DLC (Data Lenght Code)	4	Number of bytes of data
Data field	0-64	Data to be transmitted
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
EOF (End of Frame)	7	Must be recessive (1)

Table 2. CAN base frame format

proach could theoretically be possible. [Buschjäger and Morik 2017] compares the CPU and the FPGA approaches for Decision Trees and Random Forests, which are similar to Isolation Forest. They found that the FPGA approach is definitely the best in terms of energy consumption and provides acceptable throughput, considering a single tree. As for a forest, in case the model can fit the desired FPGA, the throughput tends to be better, considering the FPGA parallelization potential.

3.1. The time constraint

In order to prevent a frame from being correctly transmitted throughout the network, a possible solution is to introduce a bit stuffing error [Giannopoulos et al. 2017] before the end of the frame. According to the CAN specification [Bosch 1991], a bit stuffing error occurs when six equal bits are received consecutively. There are five data fields after the data transmission itself, summing up to 24 bits, since the last EOF (end of frame) bit is treated as `don't care` for frame receivers. Since six bits are needed to introduce the bit stuffing error, the detection must occur faster than the transmission of 18 bits, which results in a time constraint of 36μs for a 500 Kbps CAN network. Inducing other types of errors, such as a form error in EOF field, is also theoretically possible, although this has not been proposed as a mean of preventing a CAN frame transmission to the best of our knowledge.

3.2. Isolation Forest

In order to detect anomalies, the Isolation Forest algorithm takes advantage from the fact that anomalies are "few and different" [Liu et al. 2008]. The anomalies tend to be isolated closer to the tree root, so having a low average path length through the forest would mean that the sample is likely an anomaly.

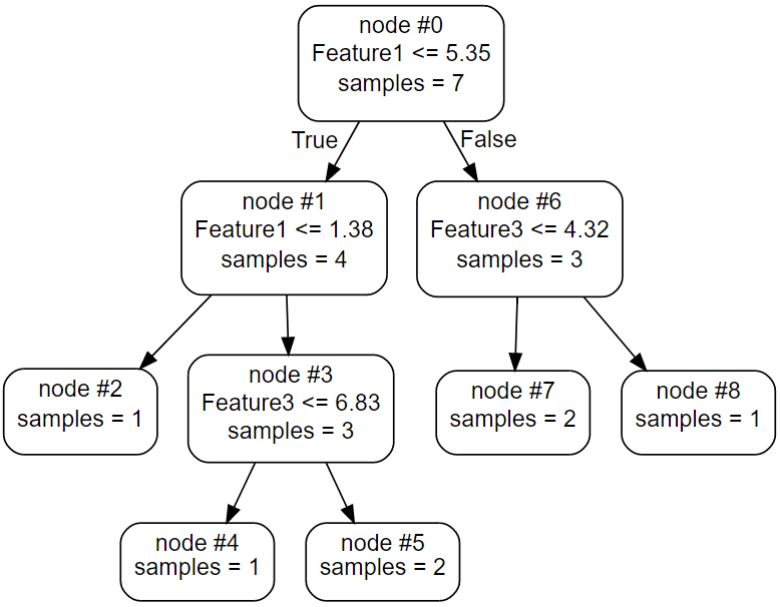

Figure 1. iTree example

3.2.1. The iTree

iTree (isolation tree) is a tree structure where every node is either an external-node with no child or an internal node with one test and two daughter nodes. A test consists of an attribute and a split value such that the test divides data points into the daughter nodes. A sample tree is shown Figure 1, where nodes $0, 1, 3$ and 6 are internal nodes and nodes $2, 4, 5, 7$ and 8 are external nodes.

3.2.2. Training stage

Isolation Forest builds an ensemble of iTrees for a given dataset taking on account two variables: the number of trees to build t and the sub-sampling size ψ. The sub-sampling size controls the trees size because the tree height limit l is defined as $\lceil(\log_2 \psi\rceil$. In an example shown in Figure 1, the sub-sampling size is set to 7, thus the tree maximum depth is 3, reached at nodes 4 and 5.

Given a dataset to build an iTree, we recursively divide it by randomly selecting an attribute and a split value until the tree reaches its maximum height or there are no different values in the subset.

3.2.3. Evaluating stage

At evaluating stage, an anomaly score must be produced. This score is derived from the expected path length $E(h(x))$ for each test instance. A single path length $h(x)$ for an instance x is calculated by counting the number of edges e from the root to a terminating node as instance x traverses through the iTree. However, since the iTree has a height limit, it is also added an adjustment factor that accounts for an unbuilt subtree beyond the limit.

Table 3. Data format for non leaf nodes

Field Name	Description	Size (bits)
Feature	Feature being evaluated	3
Is leaf node	Boolean	1
Threshold	Evaluation threshold value	8
Reserved	Bits reserved	4
Left	Adress of left child node	12
Right	Adress of right child node	12

Table 4. Data format for leaf nodes

Field Name	Description	Size (bits)
Reserved	Bits reserved	3
Is leaf Node	Boolean	1
Reserved	Bits reserved	12
Adjustment	Adjustment value	24

The anomaly score s is calculated as follows

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}}, \tag{1}$$

where $c(n)$ is the average path length of unsuccessfully searches in binary search trees [Liu et al. 2008].

3.3. Generating the trees

The detection algorithm for the proposed solution runs on an FPGA, but in order to train the iTrees it is much easier to use a higher level programming language in a PC environment. Once the trees are generated, it is necessary to encode them in order to export to the FPGA solution. Tables 3 and 4 show a possible way to encode the trees' nodes as it was also done in our experiment, which will be described in section 4.

3.4. Detecting anomalies in the FPGA

The hardware implementation is based on the design proposed in [Struharik 2011], where the author creates a methodology for implementing decision trees in hardware. In every clock cycle, the evaluated sample traverses a node of the tree, comparing the selected feature with the threshold. The threshold and the feature selection are information that should be stored in each node encoded data, as presented in table 3. At last, when all trees get to a leaf node, the nodes depths are added to the each adjustment value. All path lengths adjusted are summed up and shifted left in order to get the mean value, which is then compared to a threshold, thus evaluating the sample.

The path length algorithm implementation for an iTree is shown in Figure 2, obtained from Quartus software. The input signals are the data for the node being iterated, the features (i.e., CAN payload bytes), enable and clock; the output signals are the node address and the node depth with the adjustment factor added. The multiplexers in the left select which feature will be evaluated depending on node data (field Feature in Table 3). Current depth and current address are stored in registers at the right. It should be noted

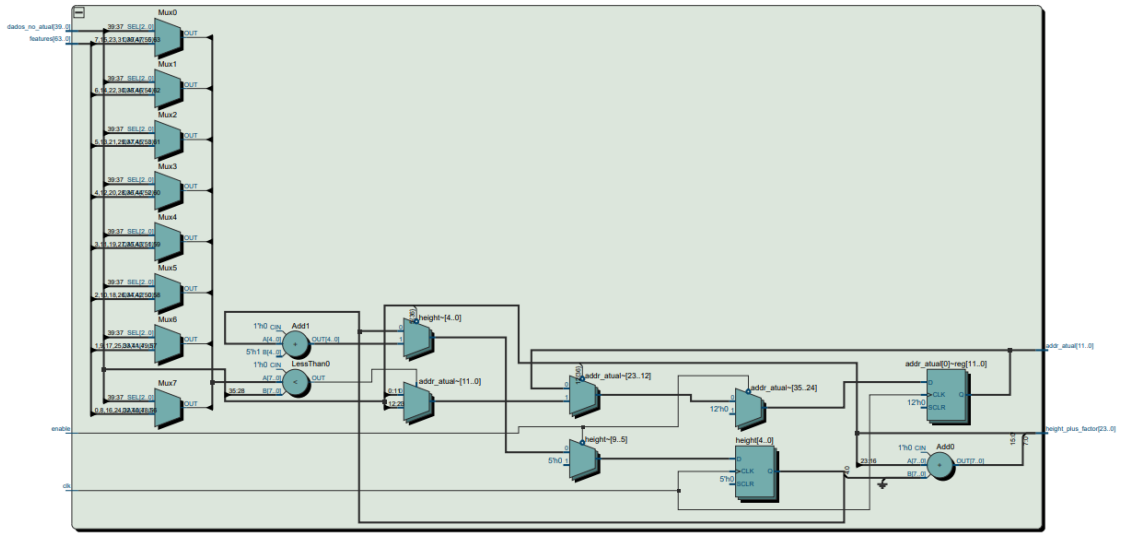

Figure 2. iTree implementation in hardware

that the forest is not trained on the FPGA, so the tree training algorithm is not considered, only the sample path length algorithm as defined by [Liu et al. 2008].

4. Experiment and Results

4.1. Experiment environment and setup

The equipment used in our experiments were: 1) a raspberry Pi 3 Model B; 2) Terasic DE2-115; 3) an i5 2.5Ghz with 8GB RAM Laptop; and a 4) Saleae Logic Pro 8. In the laptop, the Isolation Forest was trained and tested using Python and the scikit-learn library. The trained model was then retrieved by making some changes in the library code, using the format shown in Tables 3 and 4. Also using the laptop, the hardware description code with the forest data was developed and tested using Quartus software. In order to test the FPGA solution, test data was loaded in the raspberry Pi, which transmits it through General Purpose Input Output (GPIO) to the DE2-115 board. The result, i.e. an anomaly or not, was then sent to the raspberry as soon as a complete CAN frame was received. In order to measure the time needed for the FPGA to respond to the raspberry, the Saleae logic analyzer was used. This test scenario is showcased in Figure 3.

4.2. Dataset

As a proof of concept for the proposed methodology, a public CAN dataset is used. The dataset, which consists of CAN traffic collected from a Hyundai's YF Sonata, includes the attacks performed by the researchers of [Seo et al. 2018].

The dataset provides data for four types of attack: DoS (Denial of Service), fuzzy, spoofing gear and spoofing RPM. A DoS attack consists of injecting high priority frames into the network in order to prevent legitimate traffic to be sent. Corrupting a DoS frame wouldn't allow for legitimate traffic to be sent instead, thus this attack will not be considered. A fuzzy attack is usually a way to explore and search for vulnerabilities in the network. The attacker randomly modifies packets in order to look for unexpected behaviour. Spoofing attacks usually consist of impersonating a node in the network by sending messages with adequate sender addresses. Regarding CAN, since there is no sender address

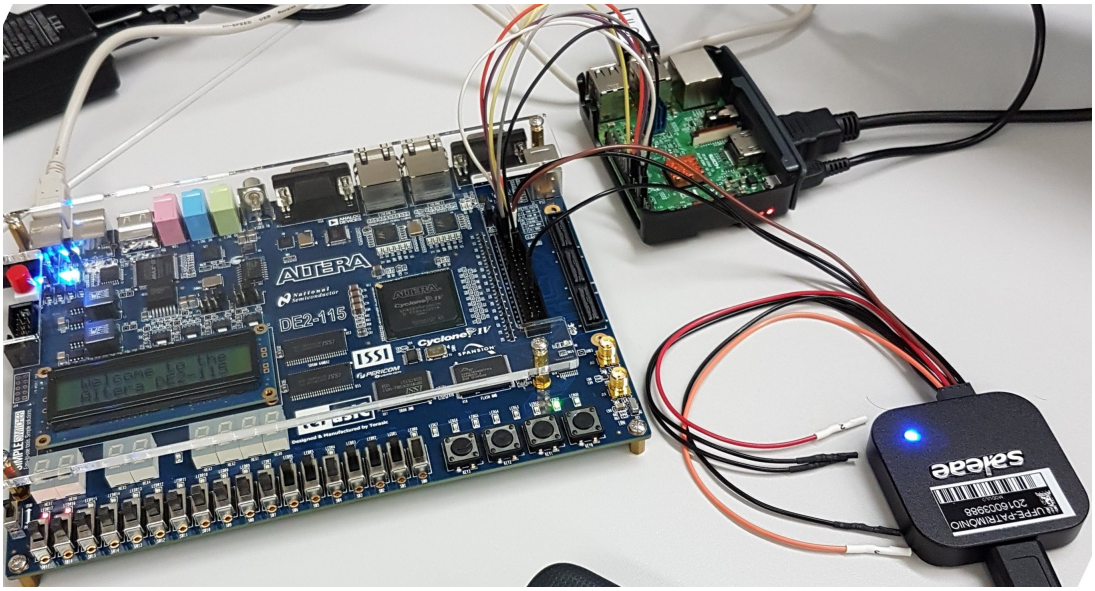

Figure 3. Test setup

field, the attacker spoofs the ID field, which is more related to the information being sent than the sender.

In our experiment, each dataset, one for each kind of attack, was divided into a training set, a validation set and a test set. The training set contains only normal data, whereas the validation and test sets contain both normal and attack data.

4.3. Isolation Forest training

Although the dataset contains other information such as time and CAN ID, the only features used are the eight bytes present in the CAN data field. Using the eight data bytes as our model's features allowed us to not spend time in preprocessing while still achieving state of art detection rates.

The model would have to fit the constraints imposed by the FPGA we use. At first, we used high values for the number of estimators (i.e., trees) and sub-sampling size. However, it became clear soon that the generated model would not fit the DE2-115 FPGA. As we tested different values of parameters, we found that a 10,000 sub-sampling size and 16 estimators would fit our FPGA and the generated model would still be able to detect anomalies with performance comparable to other solutions.

After the training process, we were able to produce an anomaly score for any sample and in our validation process we optimize a threshold so it is possible to classify the sample, whether it is normal data or an anomaly. In the test process we benchmark the generated model, obtaining statistics about its performance.

4.4. Exporting the model

To export the model it was necessary to encode the iTrees. For non-leaf nodes, the used format is described in Table 3 and for leaf nodes in Table 4. Since requiring only 40 bits per node, this model would require at most less than 1 megabyte, considering the selected model parameters.

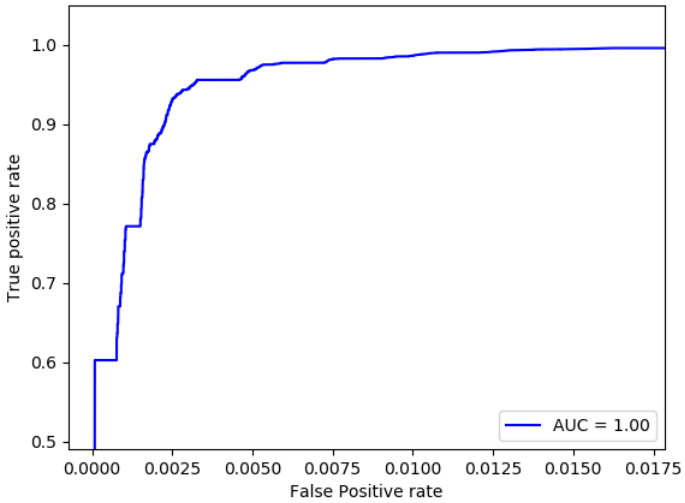

Figure 4. ROC Curve for fuzzy dataset

The encoded data for all trees are stored in RAM and is read during the algorithm execution in the FPGA. Our implementation of the detection algorithm in Verilog was tested using the scenario described in Section 4.1. The FPGA implementation behaves exactly equal to the Python implementation when detecting anomalies in the test dataset and, as expected, is able to respond much faster as will be shown in Section 4.5.

Regarding the use of resources, using the defined forest parameters (10,000 sub-sampling size and 16 trees), the implementation needed 91% of the DE2-115's total logic elements. When using 20,000 sub-sampling size and 16 trees, the usage of the logic elements would go higher then 100%, thus not being a feasible parameter configuration.

As mentioned in subsection 3.2.2, the tree height limit is defined as $\lceil (log_2 \psi) \rceil$, thus being 14 for our tree with $\psi = 10,000$. Therefore, since we use a 50 MHz clock in the described setup, the maximum evaluation time for a single sample is 280 ns.

4.5. Results

In order to measure the detection performance for the proposed method, a 10-fold cross-validation was performed for each dataset. During our validation step, we optimize the anomaly detection threshold based on results obtained using the validation dataset. We propose the use of 2 thresholds: 1) one that detects more anomalies, but also more false positives; and 2) another that detects less anomalies but minimizes false positives. This approach allow us to only actively prevent the frames that the system considers "more anomalous" using threshold 2 and take other actions if the frame surpass only threshold 1. However, as we can observe in Figures 5 and 6, this approach will not give any benefit in our spoofing attack scenario. After a certain threshold, the false positive rate grows and the true positive rate remains the same, meaning that changing the threshold above that point will only result in more false positives, so there is no real tradeoff. On the other hand, this approach can be used in the fuzzy attack scenario, since changing the threshold value affects both true and false positive rates as Figure 4 shows.

The Area Under the Receiver Operating Characteristics (AUROC) and its standard deviation (SD) in the 10-fold cross validation for each dataset are shown in Table 5. An

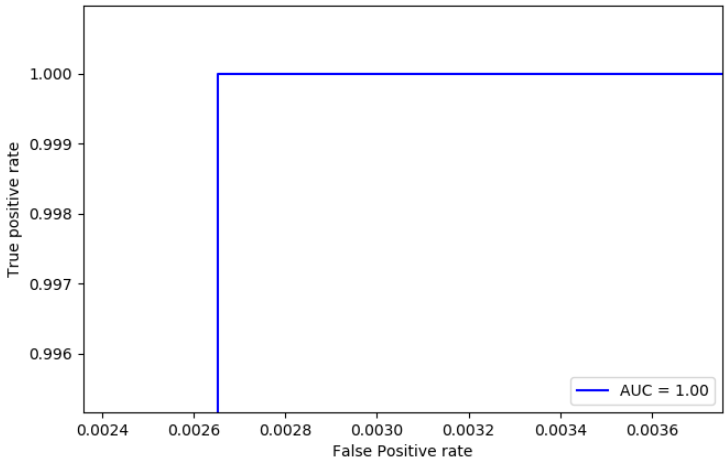

Figure 5. ROC Curve for Spoofing gear dataset

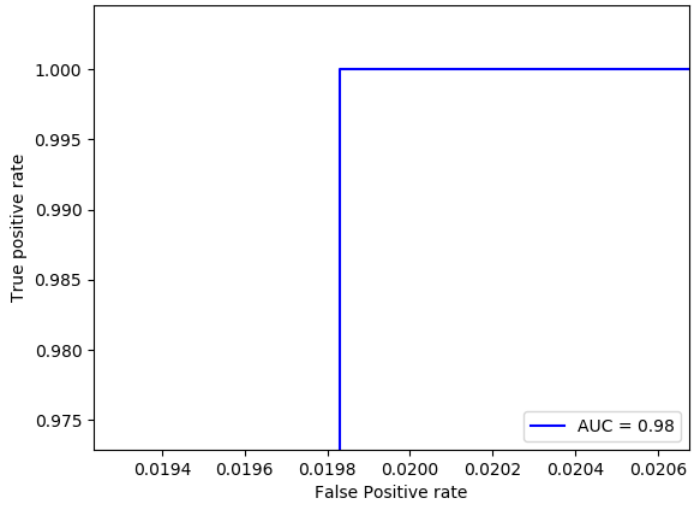

Figure 6. ROC Curve for Spoofing RPM dataset

AUROC close to 1 is the desirable value, since it would mean a perfect separation between the 2 classes, normal data or anomaly data. For all datasets the proposed system was able to achieve an AUROC higher than 0.98.

The mean and standard deviation of the other metrics obtained in the cross validation for each dataset are shown in Tables 6, 7 and 8, respectively. In case of the fuzzy dataset, we show the results obtained for both thresholds. The utilized metrics were precision, recall and accuracy. The precision rate indicates the ratio of correctly detected anomalies among predicted anomalies, which includes false positives, whereas recall indicates the ratio of correctly detected anomalies among all anomalies in test set.

As GIDS also uses the same dataset in its experiment [Seo et al. 2018], it is possible to compare directly its performance for each type of attack, as shown in Table 9. In Table 9, Proposed IDS (1) refers to the solution with Threshold 1, whereas Proposed IDS (2) means the solution with Threshold 2. Table 9 shows that, despite being able to meet the required time constraint to build an IPS, the proposed solution also have close detection performance to other solutions.

Table 5. ROC Area Under Curve (AUC)

	ROC AUC	
	Mean	**SD**
Fuzzy	0.998	<0.001
Spoofing (RPM)	0.984	0.014
Spoofing (Gear)	0.996	0.001

Table 6. Results for Fuzzy dataset

	Threshold 1		Threshold 2	
	Mean	**SD**	**Mean**	**SD**
Precision	0.872	0.026	0.996	0.002
Recall	0.997	0.001	0.704	0.079
Accuracy	0.979	0.004	0.959	0.01

Table 7. Results for (Spoofing) Gear dataset

	Mean	**SD**
Precision	0.980	0.009
Recall	1	0
Accuracy	0.997	0.001

Table 8. Results for (Spoofing) RPM dataset

	Mean	**SD**
Precision	0.918	0.070
Recall	1	0
Accuracy	0.986	0.012

Table 9. Performance comparison with GIDS

Attack type	Solution	Recall	Precision	Accuracy
Fuzzy	GIDS	0.996	0.973	0.98
	Proposed IDS (1)	0.997	0.872	0.979
	Proposed IDS (2)	0.704	0.996	0.959
Spoofing (RPM)	GIDS	0.99	0.983	0.98
	Proposed IDS	1	0.918	0.986
Spoofing (Gear)	GIDS	0.965	0.981	0.962
	Proposed IDS	1	0.98	0.997

Although the algorithm for the hardware implementation guarantees that the maximum detection time is hT, a time measurement was performed using a logic analyzer at a sample rate of 500 million samples per second. The FPGA takes 0,28μs to analyze a data frame, which is the time between the start of a detection (Enable signal) and the end of a detection (Done signal) as shown in Figure 7.

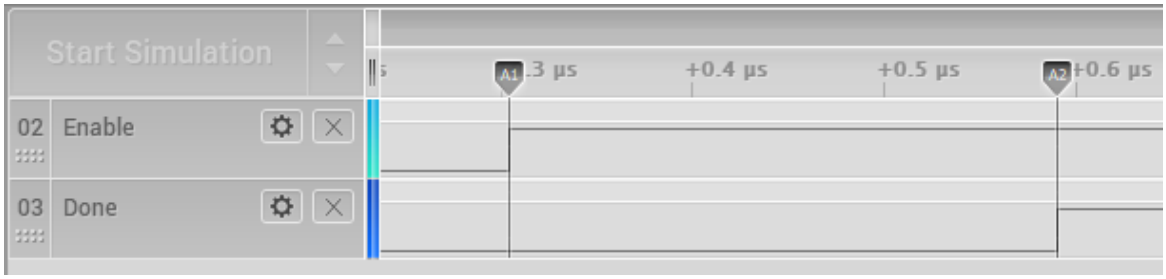

Figure 7. Detection time

5. Discussion and Future Work

In this paper, we propose a real-time anomaly-based IDS that meets a deadline of a few microseconds. Although many other anomaly-based IDS systems had been proposed for CAN, none of them meets the required deadline. Our IDS presents a detection performance similar to other proposed IDS and a feasible and cost-effective deployment in the automotive scenario, since only an FPGA is required. Other presented solutions can require multiple additional hardware or more computational power in order to work.

For a 500 kbps CAN network, responding in less than 36μs allows the IDS to effectively prevent malicious frames to be sent across a CAN network if combined with a error injection solution. Although to corrupt a suspicious frame has been suggested as a mean of preventing an attack, studies regarding the impact of this approach in automotive systems are still needed, especially considering that an anomaly-based IDS/IPS system is prone to false positives.

Isolation forests can achieve higher precision rates. The presented results were limited by the FPGA resources and methodology to export the tree. As mentioned in Section 4.4, our target FPGA limited the tree maximum size chosen as parameter to the isolation forest. Being able to use bigger trees and/or more features would boost the performance. For example, a feature that is a differential payload for each ID could enable the system in detecting attacks otherwise unidentifiable.

The presented approach could also be extended to CAN FD (CAN with Flexible Data Rate) protocol. However, the number of features would increase because of the increased payload size of CAN FD. With more features, it also may be necessary to use bigger trees. Regarding the time constraint, since CAN FD can transmit the data field and the CRC (cyclic redundancy check) field at higher rates, the deadline would be tighter, but still meetable since our proposed IDS could respond in less than 1μs.

References

Abbott-McCune, S. and Shay, L. A. (2016). Intrusion prevention system of automotive network can bus. In *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, pages 1–8.

Bosch, R. (1991). Can specification version 2.0. *Published by Robert Bosch GmbH (September 1991)*.

Buschjäger, S. and Morik, K. (2017). Decision tree and random forest implementations for fast filtering of sensor data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(1):209–222.

Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al. (2011). Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco.

Di Natale, M., Zeng, H., Giusto, P., and Ghosal, A. (2012). *Understanding and using the controller area network communication protocol: theory and practice*. Springer Science & Business Media.

Dupont, G., Hartog, J. d., Etalle, S., and Lekidis, A. (2019). Network intrusion detection systems for in-vehicle network-technical report. *arXiv preprint arXiv:1905.11587*.

Freitas de Araujo-Filho, P. (2018). Contributions to in-vehicle networks: error injection and intrusion detection system for can, and audio video bridging synchronization. Master's thesis, Universidade Federal de Pernambuco.

Giannopoulos, H., Wyglinski, A. M., and Chapman, J. (2017). Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine*, 12(4):60–68.

Kang, M. and Kang, J. (2016). A novel intrusion detection method using deep neural network for in-vehicle network security. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, pages 1–5.

Koyama, T., Shibahara, T., Hasegawa, K., Okano, Y., Tanaka, M., and Oshima, Y. (2019). Anomaly detection for mixed transmission can messages using quantized intervals and absolute difference of payloads. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 19–24. ACM.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

Liu, J., Zhang, S., Sun, W., and Shi, Y. (2017). In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, 31(5):50–58.

Miller, C. and Valasek, C. (2014). A survey of remote automotive attack surfaces. *black hat USA*, 2014:94.

Miller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015:91.

Seo, E., Song, H. M., and Kim, H. K. (2018). Gids: Gan based intrusion detection system for in-vehicle network. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6.

Struharik, J. (2011). Implementing decision trees in hardware. In *2011 IEEE 9th International Symposium on Intelligent Systems and Informatics*, pages 41–46. IEEE.

Taylor, A., Leblanc, S., and Japkowicz, N. (2016). Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139. IEEE.