Regular Paper

Long Short-Term Memory Networks for In-Vehicle Networks Intrusion Detection Using Reverse Engineered Automotive Packets

Araya Kibrom Desta[1,a)] Shuji Ohira[1,b)] Ismail Arai[1,c)] Kazutoshi Fujikawa[1,d)]

Received: October 28, 2019, Accepted: June 1, 2020

Abstract: Nowadays, vehicles are equipped with multiple Electronic Control Units (ECUs) each of which communicates with one another using a specification called Controller Area Network (CAN). CAN provides its own share of benefits in modernizing automobiles, but it also brought along a security issue to the automotive industry. CAN bus does not have any mechanism for encrypting or authenticating CAN payloads. As a countermeasure against these drawbacks, we have experimented on identifying intrusions in the CAN bus using Long Short-Term Memory Networks (LSTM). LSTM networks are trained with features extracted from reverse engineered packets. In a specific range of time windows, we have extracted three parameters, the number of packets, the bit flip rate and the average time difference that are used to train LSTM. The trained LSTM is later then used to predict all the three features which will be combined to a single anomaly signal using a root mean squared error. Depending on which side of the threshold appears the anomaly signal value, we managed to identify anomalies in an acceptable performance rate, up to F1 score of 98%. We have tested our methods with a variety of attacks on the CAN bus and demonstrated how effective our detection methods is.

Keywords: in-vehicle networks, CAN bus, intrusion detection, automotive security, neural networks, LSTM

1. Introduction

In the old days, automobiles used to be mechanical devices with no networking capability and a few electronic devices. Later in time, the number of Electronic Control Units (ECUs) inside automobiles have increased to some extent. But still, there was not any network that connects these ECUs. A wiring harness was being used to connect each of these devices until CAN was introduced by Bosch [2] as a multi-master, message broadcast system that specifies a maximum signaling rate of 1 Mbps. As of the CAN bus release, the number of ECUs in vehicles started to increase due to the simplicity of inter-networking. Nowadays, automobiles have up to 100 ECUs Kuwahara et al. [20], each of which communicates using the CAN, which is the de-facto standard of most present-day vehicles.

CAN was introduced to the automotive industry for the sole purpose of enhancing the driving experience. Before CAN was invented, ECUs used to communicate with one another by a point to point wiring system. After the introduction of the CAN bus to vehicles, the wiring harness between ECUs has reduced to less and simple wiring, which in return saves the overall cost and time. Additionally, the specification calls for high immunity electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing Corrigan [5].

Even though CAN provides its share of advantages in modernizing the automotive industry, it also brought along a new security hole in in-vehicle networks. CAN is vulnerable to various types of attacks due to its security weakness Koscher et al. [19]. CAN uses a broadcast type of communication method, whenever a message appears in the CAN bus it is accessible by all its connected nodes. This nature of the CAN bus enabled attackers to easily snoop on all communications or send packets to any other node on the network, which later on can help the attackers to reverse engineer the packets. Whenever two competing nodes want to use a CAN bus resource, the CAN bus handles the priority of the nodes using a message arbitration ID. A node with a lower arbitration ID will be given a higher priority to use the CAN bus resources. This property of the CAN bus makes it extremely vulnerable to a Denial-of-Service Attack (DoS). An attacker can continuously inject a packet with a lowest arbitration ID to the CAN bus, and this causes all other packets to be rejected from using the CAN bus. CAN bus also has no mechanism for encrypting or authenticating CAN packets.

By exploiting these security vulnerabilities of the CAN bus many security researchers have been able to control critical vehicle components. Miller and Valasek [23] illustrated on how they can control critical car components by remotely injecting packets to the target vehicle's CAN bus. Other security researchers have also demonstrated on how to remotely attack Tesla motors

[1] Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] kibrom_desta.araya.js3@is.naist.jp
[b)] ohira.shuji.ok2@is.naist.jp
[c)] ismail@itc.naist.jp
[d)] fujikawa@itc.naist.jp

of Tencent [26]. They discovered multiple security vulnerabilities and successfully implemented remote, aka none physical contact, control on Tesla Model S in both Parking and Driving Mode.

In order to avoid the security risks in the in-vehicle network, in this paper, we experimented on identifying intrusions in the CAN bus by using neural networks. And inspired by LSTM's design to overcome short term memory problems even in case of noisy, incompressible input sequences. That is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant (thus, neither exploding nor vanishing) error flow through internal states of special units. We have proposed an intrusion detection system using LSTM (LSTM-IDS) with the following contributions:

(1) We proposed LSTM-IDS with a suitable architecture of layers that is capable of learning to identify intrusions. We further studied how the architecture of layers have a big effect on the prediction capability of the neural network. The conventional method proposed by Taylor et al. [30] uses the bits in a packet to identify anomalies but they didn't take into consideration the meaning of each bit in the packet. Our methodology further reverse engineers the packets to learn about the counter bits, if any, of each bit in a packet and this helps in improving the intrusion detection from an average F1 score of 33% to 98% for the first car's arbitration ID 0C000027.

(2) The conventional method has no way of telling if messages are arriving at their intended time. Messages can be delayed due to different types attacks, a DoS attack in our case which we handled by also predicting the arrival time of the messages. If a message is delayed due to DoS attack, the conventional method will be bypassed.

(3) We fabricated different types of attacks to test if the trained network is competent enough to be used as an intrusion detection method. Our experimental results show that with the right combination of parameters, LSTM is suited for intrusion detection in in-vehicle networks. For some arbitration IDs, the proposed method can detect most of the attacks by a precision and recall values of up to 1.0.

(4) We also compared the detection performances for most arbitration IDs in the can bus with the conventional method that uses different attributes and network architecture to accomplish the task. When the number of counter bits found in a packet is higher, as in CAR B's arbitration IDs, 114, 119 and 18E, the detection F1 score doubles by up to 50% from the conventional method. Moreover, we discussed how this LSTM-IDS can be implemented on a real vehicle.

2. Related Work

The security vulnerabilities of a CAN bus can be improved by two methods. The first method is to use intrusion prevention techniques by tackling attacks before the packets arrive in the target host. A CAN bus has no authentication field or any mechanism to identify the source of a message. As the solution for these drawbacks Mundhenk et al. [24] have proposed a way to use a light message authentication for secure automotive networks. In their research, the keys required for secure communication with symmetric cryptography, such as the Advanced Encryption Standard (AES) or MACs, can be exchanged between ECUs. In Han et al. [14], a different approach to introducing MACs into FlexRay is presented. Groza and Murvay [11], Hazem and Fahmy [15], Groza et al. [12] are also among the different approaches presented on authenticating the CAN bus messages. Besides that, intrusions can also be prevented by encrypting CAN packets, Wu et al. [34]. This method proposes a security protocol for the CAN system based on AES-128 encryption and HMAC function. Woo et al. [32], Halabi and Artail [13], Farag [7] are also some of the researches that introduce the use of different encryption algorithms for encrypting the CAN bus. Despite the fact that these approaches can provide a way to enhance CAN security, the implementation of these methods in a real vehicle would be of a great risk due to the real-time communication requirements of in-vehicle networks. For this research, we are mainly dealing with detecting intrusions instead of enhancing the intrusion prevention mechanisms.

The second approach for enhancing the security of the CAN bus is to implement intrusion detection mechanism in the CAN bus. This perspective can also be further split to signature based and anomaly based intrusion detection Casillo et al. [3]. Signature based systems detect attacks using a pre-defined knowledge base of attack signatures that is captured and created, and current network traffic is monitored for these signatures. Ivan Studnia [16] uses language theory to elaborate a set of attack signatures derived from behavioural models of the automotive calculators in order to detect a malicious sequence of messages transiting through the internal network. When a packet appears in the CAN bus, together with other information, it contains timing information, data field, and arbitration ID. Researchers have been illustrating on using timing information to identify intrusions in the CAN bus. Song et al. [28] used time intervals of CAN messages for intrusion detection. Their method uses the timing information of each ID by taking the fact that there is a unique time interval for each arbitration ID because each ECU connected to CAN bus sends messages regularly. When a new message appears on the CAN bus, their (Intrusion Detection System) IDS checks the arbitration ID and computes the time interval from the arrival time of the latest messages. According to their method, an anomaly message's time interval is shorter or longer than the normal range. Taylor et al. [29] experimented on using the frequency of messages to identify an anomalous sequence of messages. Their anomaly detector works by calculating statistics about ongoing network traffic and comparing them with historical values. They also trained a OCSVM detector with variables collected from the flow of the traffic. OCSVM was able to detect very short packet insertions with acceptable false alarm rates. Kuwahara et al. [20] also used CAN message frequencies for in-vehicle network intrusion detection. This approach analyzes the normal behavior of CAN messages in terms of their message frequency. They used two features, total counting feature and ID counting feature, which count the number of messages and available IDs in some time windows.

In this category, intrusions can also be detected using anomaly detection methods. Signature based approaches have the problem of out dating fast due to the numerous attack signatures that are

created everyday. In this category there are methods that proposes an anomaly detection method based on identifying the source of CAN messages or a way that suggest the exploitation of the physical layer feature of ECUs for intrusion detection. Kneib and Huth [17] proposes a system for identifying the origin of CAN messages using the physical characteristics of CAN signals. Foruhandeh et al. [8] propose an anomaly detection that monitors the CAN environment for secure updates of fingerprints to compensate for environment changes, such as the temperature and the supply voltage.

It's clear to see that most of the aforementioned research heavily focus on using timing information of CAN messages or the signature of attacks to detect intrusions. Furthermore, those that use the timing information techniques can only be implemented to periodic CAN messages. All the aforementioned methods fail to detect any anomaly sent with a non-periodic arbitration ID or a compromised ECU which attackers are capable of manipulating its timing information. This problem can be tackled by using information-theory based and machine learning based IDS methods. Wu et al. [33] have developed an entropy based IDS system. It uses a sliding window comprised of a fixed number of messages. Even though this approach works best in detecting DoS and spoofing attacks, IDs methods based on information entropy are generally ineffective in detecting attacks that modify the data field of CAN packets. And with the case of the two attack types, attackers can bypass the IDS if they know the algorithm is being used in a targeted vehicle.

Machine learning approaches have a better way of identifying anomalies that modify the data field of CAN packets. A Machine learning approach can be deployed to monitor how each bit in the data portion of a payload is changing through time in addition to the timing information. Nanduri and Sherry [25] and Taylor et al. [30] have proposed a method that can solve this issue in aircraft and vehicles respectively. Both of these methods focus on identifying intrusion by training a neural network that is capable of predicting the next packet data values, and its errors during prediction are used as a signal for detecting anomalies in the sequence. Our work is an improvement to the intrusion detection method research work by Taylor et al. [29] which takes the advantage of both using the timing information and the bit prediction method to further improve the prediction capability of a LSTM network. Using the frequency of the message and the arrival time of the message also helps us in interpreting the likely associations between the messages in the chosen window in addition to the bit flip rate predictions.

3. CAN Specification

CAN network is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The specification calls for a high immunity to electrical interference and the ability to self-diagnose and repair data errors. These features have led to CAN's popularity in a variety of industries including building automation, medical, and manufacturing.

A controller area network is ideally suited to the many high-level industrial protocols embracing CAN and ISO-11898:2003 as their physical layer. Its cost, performance, and upgradeability provide for tremendous flexibility in system design. In a CAN network, many short messages like temperature or RPM are broadcast to the entire network, which provides for data consistency in every node of the system Corrigan [6].

Fig. 1 CAN packet.

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a preprogrammed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic high in the identifier keeps on transmitting because it is the highest priority. Since every node on a bus takes part in writing every bit "as it is being written," an arbitrating node knows if it placed the logic-high bit on the bus.

In our detection method, we are mainly focusing on using the arbitration ID and the data field of a CAN packet as it is depicted in **Fig. 1**. CAN has two standards, the first standard with a 11-bit identifier, provides for signaling rates from 125 kbps to 1 Mbps. This standard was later amended with the "extended" 29-bit identifier. The standard 11-bit identifier field provides for 2^{11}, or 2,048 different message identifiers, whereas the extended 29-bit identifier provides for 2^{29}, or 537 million identifiers. Both of the standards are almost similar, except for some control and identifier fields. ECU's bus access is event-driven and takes place randomly. If two nodes try to occupy the bus simultaneously, access is implemented with a nondestructive, bit-wise arbitration, CAN_ID in Fig. 1, the node winning arbitration just continues on with the message without the message being destroyed or corrupted by another node. The lower the binary message identifier number, the higher its priority. An identifier consisting entirely of zeros is the highest priority message on a network since it holds the bus dominant the longest. And the data part in, Data Fig. 1, contains information that is disseminated to the ECUs depending on the driver's actions. Our proposed method uses these two parts of the packet and the time the packet appears in the CAN bus to train LSTM-IDS that can identify anomaly messages.

4. Attack Model

Dumping a CAN packet from a vehicle would show us a list of arbitration IDs and their respective data indexed by the time stamp. Our LSTM is trained with these three variables and one more variable, "number of messages." "Number of messages" is evaluated by counting the number of messages in a specific time window. To test the effectiveness of our trained network, we have simulated a variety of attacks on the CAN bus. Most of the attacks we have created consider the fact that packets are either removed or added deviating the sequence of the packets. We have fabri-

cated four types of attacks that adversaries would try to inject to the CAN bus. With these types of attacks, we have tested the prediction accuracy of our trained network. These types of attacks are listed below

Fuzzy Attack - The case wherein adversaries implement a bunch of packets to be injected in the CAN bus for the purpose of learning how different ECUs behave. To simulate this specific type of attack we have created a program that generates a packet with a size 8 bytes or less, depending on the data length code (DLC) of each arbitration ID, and inserted in a random time location. In particular, our fuzzy attack generator randomly selects a bit 1 or 0 to create one packet with the size of a targeted arbitration ID's DLC.

Insertion Attack - This attack is simulated by randomly selecting a packet from the list of packets dumped from each vehicle to be inserted in a different time location. The time for this attack messages is assigned by calculating the average arrival time from two of the neighboring messages.

Drop Attack - The case wherein attackers drop some payloads from a targeted ECU preventing the messages from being processed or used by a receiving ECU. This attack is simulated by removing two packets in a 1 second time window. Hence causing an unexpected situation in the in-vehicle networks.

Impersonation Attack - An Impersonation attack is an attack wherein adversaries drop some messages and send a fabricated replacement message to get control of critical car components. Our impersonation attack is same as insertion attack except a message has to be dropped from the CAN bus and it will be replaced by a randomly selected packet or it can also be manipulated on the fly without dropping the whole packet.

Denial of Service Attack (DoS) - An attack meant to force a targeted ECU to continuously back off from using a CAN resource, making it inaccessible from participating in the network. DoS attacks can be accomplished in vehicles by flooding a CAN with a message integrated with the lowest arbitration ID. To simulate this attack we have delayed the arrival time of messages by up to 5 ms.

5. Intrusion Detection Algorithm Using LSTM

To identify anomalies sent with a non-periodic arbitration ID or a compromised ECU, the detection methodologies should track how the data is changing through time. Data sequence learning methods can be used to track data of payloads through time. A simple strategy for general sequence learning is to use neural networks. By training a neural network about a sequence of messages, it will be able to predict forthcoming payloads using information which has already appeared in the CAN bus.

We trained a neural network to predict subsequent CAN payload data using previously seen data sequences. To train the network we need to prepare training data, unfortunately, we couldn't find public data that can fit our network. Due to this, we had to prepare our own benign and anomalous payload sequences.

The conventional method in Taylor et al. [30] can identify anomalies even if adversaries mimic the timing information of a compromised ECU. This is done by learning the flow of data to see if there are packets out of a correct sequence. Even though, this method can identify any fuzzy attack at 1.0 accuracy, it fails to detect delayed messages because it only uses the data portion of packets for anomaly detection. Messages can be delayed due to an insertion attack or a Dos attack. Our method further incorporates the arrival time of packets to see if each message is arriving in the CAN bus at the expected time. This helps us to further improve the detection accuracy and identify DoS attacks that can't be detected by the conventional method.

5.1 Long Short-Term Memory Networks - LSTM

We selected the Recurrent Neural Network (RNN) for our research because we wanted to see if RNNs are capable of connecting previous information to the present task, such as whether using previous payloads might inform the understanding of the present payload. But due to the long-term dependencies problem, RNNs are incapable of retaining information for a long time Bengio et al. [1]. The basic problem is that gradients propagated over many stages tend to either vanish(most of the time) or explode(rarely, but with such damage to the optimization). Recurrent networks involve the composition of the same function multiple times, once per time step. These compositions can result in extremely nonlinear behavior Learning [21]. Long Short-Term Memory Networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies Gers et al. [9]. The main contribution of LSTM is the capability of using self-loops to produce paths where the gradient flow for long durations. LSTM has multiplicative gates, which allows LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem, Graves [10]. For example, as long as the input gate remains closed (i.e., has an activation near 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate. The preservation over time of gradient information by LSTM is illustrated in **Fig. 2**. In the figure, shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open ('O') or closed ('-'). The memory cell 'remembers' the first input as long as the forget gate is open and the

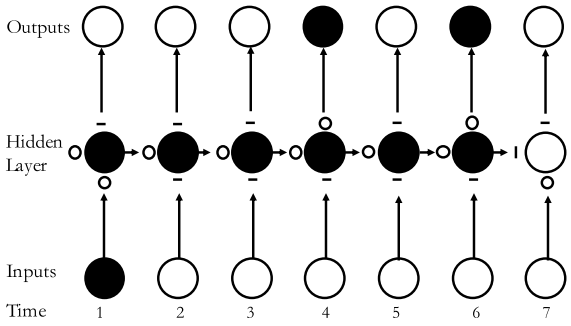


Fig. 2 Preservation of gradient information by LSTM.

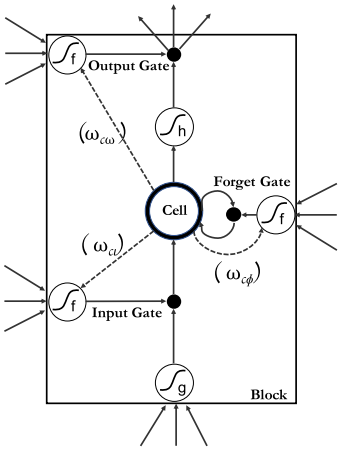


Fig. 3 Diagram for LSTM at the time step t including all the gates.

input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Figure 3 provides an illustration of an LSTM memory block with a single cell. The LSTM equations for calculating the corresponding gate values in a single node of a forward pass are shown in Eqs. (1)–(9). In Fig. 3, the subscripts ι, ϕ and ω refer respectively to the input gate, forget gate and output gate of the block. The subscript c refers to one of the C memory cells. The peephole weights from cell c to the input, forget and output gates are denoted $\omega_{c\iota}$, ω_{c_ϕ} and $\omega_{c\omega}$ respectively. ${s_c}^t$ is the state of cell c at time t (i.e., the activation of the linear cell unit). f is the activation function of the gates, and g and h are respectively the cell input and output activation functions. Let I be the number of inputs, K be the number of outputs and H be the number of cells in the hidden layer. Index h is used to refer to cell outputs from other blocks in the hidden layer, exactly as for standard hidden units. G defines the total number of inputs to the hidden layer, including cells and gates, and use the index g to refer to these inputs when a wish for distinguishing between the input types is not needed. Following the Fig. 3 and the conventions we explained above, the calculations for each of the gates is done using Eqs. (1)–(9). These equations are given for a single memory block only. For multiple blocks the calculations are simply repeated for each block, in any order.

Input gates

$$ {a_\iota}^t = \sum_{i=1}^{I} \omega_{i\iota} {x_i}^t + \sum_{h=1}^{H} \omega_{h\iota} {b_h}^{t-1} + \sum_{c=1}^{C} \omega_{c\iota} {s_c}^{t-1} \tag{1} $$

$$ {b_\iota}^t = f({a_\iota}^t) \tag{2} $$

Forget Gates

$$ {a_\phi}^t = \sum_{i=1}^{I} \omega_{i\phi} {x_i}^t + \sum_{h=1}^{H} \omega_{h\phi} {b_h}^{t-1} + \sum_{c=1}^{C} \omega_{c\phi} {s_c}^{t-1} \tag{3} $$

$$ {b_\phi}^t = f({a_\phi}^t) \tag{4} $$

Cells

$$ {a_c}^t = \sum_{i=1}^{I} \omega_{ic} {x_i}^t + \sum_{h=1}^{H} \omega_{hc} {b_h}^{t-1} \tag{5} $$

$$ {s_c}^t = {b_\iota}^t g({a_c}^t) \tag{6} $$

Output gates

$$ {a_\omega}^t = \sum_{i=1}^{I} \omega_{i\omega} {x_i}^t + \sum_{h=1}^{H} \omega_{h\omega} {b_h}^{t-1} + \sum_{c=1}^{C} \omega_{c\omega} {s_c}^{t-1} \tag{7} $$

$$ {b_\omega}^t = f({a_\omega}^t) \tag{8} $$

Cell outputs

$$ {b_c}^t = {b_\omega}^t h({s_c}^t) \tag{9} $$

Once, the network makes a forward pass updating the variables listed in the equation, the network again makes a back propagation depending on the loss function. The partial derivatives made to update the weights in the back propagation are omitted in this paper. For details of the forward and back ward phases, readers can refer to Ref. [10]. This research uses LSTM without peephole weights, hence in the above equations all the weights that connect the cells to any of the gates are set to zero.

5.2 Input Data Preprocessing

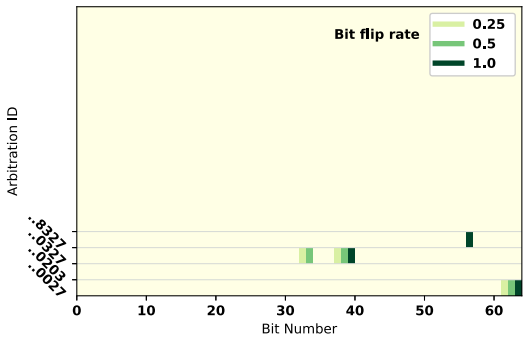
The purpose of our research is to identify an anomalous message in short time before attackers cause much damage on a targeted car. We implemented a way to identify anomalies in less than one second from the time the anomalies start to appear in the bus. We used 1 second so that we would be able to collect enough data for processing. When the look back value of LSTM is very short, LSTM did perform well due to all the different patterns it has to learn Wang et al. [31] Kong et al. [18]. For this reason, we have collected input and output values to our LSTM network in every 60ms seconds for car A and 75ms seconds for car B. We used different time ranges due to the behavior of each car's packets. The cars have a packet with two or more bits put together with the rest of the bits that are used as a counter.

The values we used as inputs to the network and as outputs are listed below.

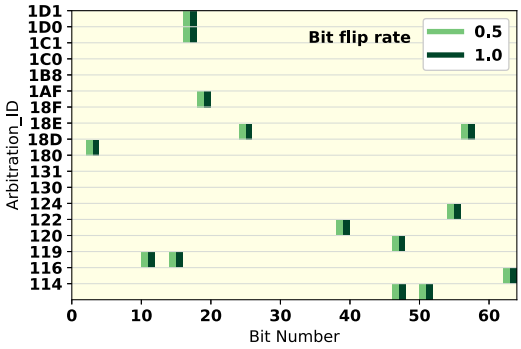
- **Number of Messages n -** This variable counts the number of messages that appear in the CAN bus in every 60ms for car A and 75 ms for car B. Counting the number of messages can give us an approximation of the number of messages that should appear in a certain time window. If the number of messages are way higher or lower than the predicted value, this can significantly affect our anomaly signal to better detect intrusions.
- **Average Time Difference T_{diff} -** In every range of milliseconds we collect messages associated with their respective arbitration IDs. Each of these payloads has timing information that shows the time the payloads appeared in the CAN bus. To calculate the average time difference we collected a bunch of messages with their particular timing information in the specified time range, then we took the average time difference between each packet. This also helps us in tracking if a packet with a specific arbitration ID is appearing in the CAN bus at the intended time.
- **Bit Flip Rate $b_0, b_1, \ldots, b_{63}$ -** A CAN payload can be 64 bits long or less depending on the arbitration ID. To calculate bit flip rate we employ an algorithm used by READ, Marchetti and Stabili [22]. The bit flip rate is evaluated for each bit of the payload, independently of its neighbors. Bit flip rate algorithm, Algorithm 1, counts the number of bit

Algorithm 1 Pseudo-code of bit flip rate calculation

```
1:  function BitFlip-Rate(messageList, DLC)
2:      payloadLen ← len(messageList)
3:      bitFlip ← array(DLC)
4:      previous ← messgeList[0]
5:      while item in (1...messageList) do
6:          for ix in range(DLC) do
7:              if item[ix] ≠ previous[ix] then
8:                  bitFlip[ix] + +
9:          previous ← item
10:     for ix = 0; ix < DLC; ix + + do
11:         bitFlip[ix] ← bitFlip[ix]/payloadLen
12:     return bitFlip
```



(a) Car A

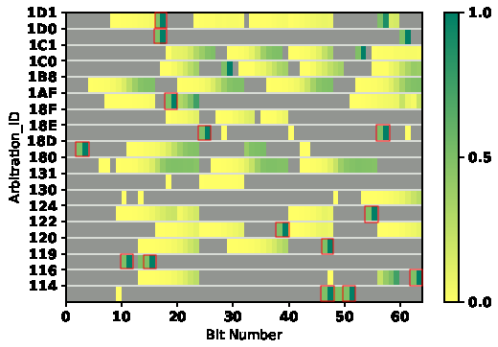


(b) Car B

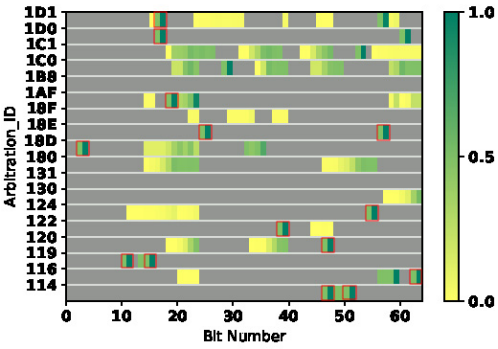
Fig. 4 Bits identified as a counter in the test cars. Car A has up to three counter bits with a counter bit flip rate of 0.25, 0.5 and 1.0, while car B has only two counter bits, 0.5 and 0.1.

flips (from 0 to 1 and vice versa) occurrences among consecutive messages collected in the specified instant of time, *messageList*. Then the bit flip rate is obtained by dividing the number of *bitFlip* to *payloadLen*. The result from this is an array of k elements each representing the bit flip rate for a single bit of the data field for a given ID, where k represents the number of bits included in the payload as defined by the value of the DLC field. The complete algorithm that calculates the bit flip rate is shown in Algorithm 1.

We used 75 ms for car B for the reason that the bit flip rate algorithm was not able to capture the transformation of the bits that are used as a counter because car B has more number of bits, up to three, used as a counter. **Figure 4** shows the counter bits available with each car. The input to the neural network will be the values calculated in the specified time for a range of 1 second.



(a) Driving mode



(b) Stationary mode

Fig. 5 The red strokes in both of the modes show the bits identified as counter bits.

Using the inputs collected in 1 second, we predict the next values.

The advantage of using bit flip rate as an input feature to our neural network is it helps us to convert the counter bits to constant bit flip rate values saving the neural network from learning the flipping of these specific bits. For instance, for car A's arbitration ID 0CF00327, Fig. 4 (a), it has 3 counter bits that count from 000 to 111. In the case of using bits as an input feature, the network has to learn how each of these bits are flipping from the previous counter to the next one but if we use bit flip rate these three bits will have a bit flip rate of 0.25, 0.5 and 1.0 for all the time windows in the training.

Driving mode of the vehicles influence the bit flip rate of the targeted arbitration IDs. But, this effect is only on the bits that are not identified as counter. The counter bits on both driving modes (Stationary, Driving) have not been affected. To prove this one, we collected 10 more minutes of data from our second car, car B. The first five minutes were collected while the car is stationary and the other 5 minutes were collected while the car is moving at a fluctuating speed. As it is shown in **Fig. 5**, the frames have higher bit flip rates when the car is moving than when the car is stationary. The red strokes show the bits identified as a counter. These bits have stayed constant no matter what the driving mode is.

During evaluation the driving mode (stopped/driving) was not considered. In practical cases both modes are inseparable and we need a system that can continuously look for anomalies independent of the driving mode. But from the perspective of IDS effectiveness, we can get better results when the car is in stationary mode as there will be less bit flips in the packets.

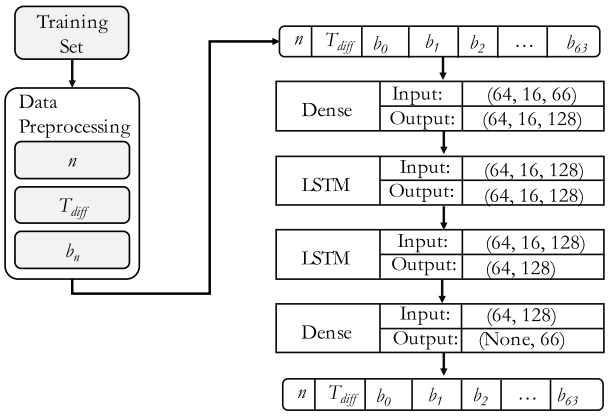


Fig. 6 IDS network architecture.

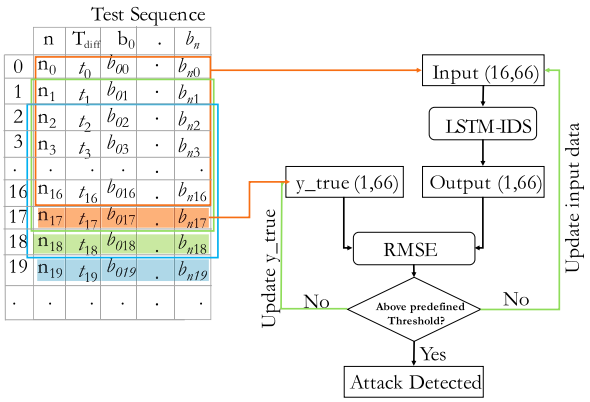


Fig. 7 Anomaly detection process.

5.3 IDS Network Architecture

Our LSTM architecture is similar to the architectures used by the conventional LSTM except some changes in input features, network architecture and some parameter changes so as to fit with our training data. The Keras model Chollet et al. [4] we implemented is depicted in **Fig. 6**.

First we need to format our training data to a format that a neural network can understand. All our input data is numerical data, so we don't need any vectorization. But the input data is on a different scale. n is an integer that ranges from 0 to some unbound value and T_{diff} also varies accordingly. To bring all the input values to a comparable range, we used L_1 normalization that brings all the input values to a range between 1 and 0. Once we brought all the 66 features to a comparable range, a single tensor with a size of (t, 66) is created, t is the total sequence of training data. Next, we converted this tensor to input and output sequence. This is created by continuously iterating over the training data to convert it to overlapping input and output sequences. Each iteration goes through 16 (steps) consecutive sequences and predicts the subsequent data. A batch size of 64 is selected for the training, i.e during each batch the shape of the training data becomes (batch_size(64), look back(16), n_feature(66)). After we setup our input and output sequence, we trained the preprocessed data using the architecture shown in Fig. 6.

The architecture we selected for our IDS contains two non-recurrent hidden layers and two LSTM layers. The training first goes through a non-recurrent hidden layer with 128 units and later passes through two LSTM layers each with the 128 units. All of these three layers have a rectified linear unit (ReLU) activation function. The LSTM layers have also a 0.2 dropout that is used to overcome over-fitting during training. Later on, the output from the last LSTM layer is fed to a single dense layer with 66 units. The output from the last dense layer is a NumPy array with 66 units that has an output for each input feature.

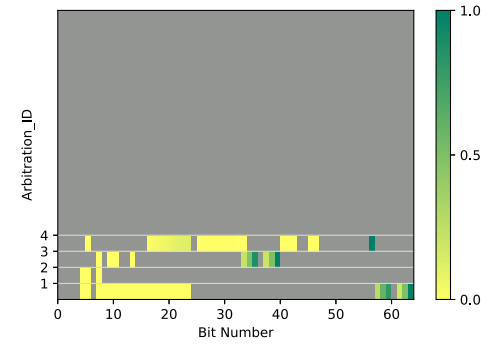
After we generate our trained model, the IDS phase passes through the steps shown in **Fig. 7**. First, we read the 1 second data collected from the available sequence of CAN data, let it be a dumped data as in offline detection or a live data being read from the CAN bus. All of the intrusion detection experiments are carried out in offline mode. Offline mode is one type of network intrusion detection system by which the dumped data is processed and passed through an IDS to decide if there has been any kind of manipulation in the collected packets. The 1 second data is first preprocessed here again in the case of live intrusion detection to convert the raw packet data to suitable input variables for the neural network. But, for this detection algorithm we implemented most of the experiments in an offline mode. Hence, we have preprocessed the sequence beforehand. The input to the neural network has a shape of (16, 66), the first variable shows the number of steps we used to predict the successive values and the second value shows the number of features we created from the data. The trained model then outputs a numpy array with a shape of (1, 66) which are the prediction results for all the input variables. Afterwards, we calculate the root mean square error (RMSE), Eq. (10), of the predicted results and the true value, y_true, true value is the actual value picked from the test data. We selected Root Mean Square Error (RMSE) so as to penalize higher errors because the mean square error gives a relatively higher weight to large errors than errors with a smaller deviation from the actual value.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y\_true_i - Output_i)^2} \tag{10}$$

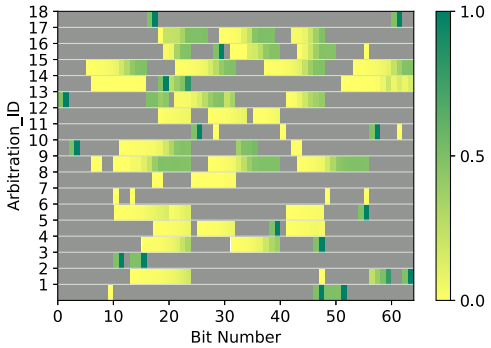
During testing, we have selected a predefined threshold value, depending on this value if the computed RMSE is above the predefined threshold value for each ID, an anomaly flag is generated implying the driver to take appropriate actions. However if the evaluated RMSE is within a normal range, we increment the pointers of each input variable by one to read the next sequence of data. This process starts when the engine of the testing car is started and continues up until the engine of the vehicle is stopped.

5.4 Training and Test Set Description

To train our neural network we collected data from two vehicles, car A and car B. From car A, we collected packets for more than 130 hours with a total of 300 million packets. This car has 42 arbitration IDs with a total average of 680 messages in a second and out of these messages about 400 packets of them are associated with four arbitration IDs, however the rest of the messages are distributed among the other 36 arbitration IDs. We selected the first 4 IDs because most of the frames, 400, from the 680 total frames in a second are associated with 4 arbitration IDs. We classified the IDs based on the frequency. The four arbitration IDs appear in the CAN bus at 100 times in a second which is big enough to see a pattern for an LSTM prediction. Since our IDS



(a) Car A



(b) Car B

Fig. 8 Bit flip rates of both cars. The grey color shows constant bits that don't flip throughout the whole training.

takes the advantage of bit flip rate, some of the selected IDs have these bits and some of them don't, **Fig. 8**.

During training no attack packets were introduced. We trained the network with an attack free sequence of training and validation data. The purpose of validation data here is to see how the network is reacting to previously unseen data. It serves no purpose here except to help us babysit the network to make it free of over-fitting or under-fitting problem. The attack packets are only introduced in the testing phase. Attack packets were put into the simulated environment after every second during testing. The LSTM makes a prediction after a second by first preprocessing the input data of 75ms for car A and 60 ms for car B. When we split the test data into two, the true value of the benign sequence stays as it is but for the simulated attacks the true value is changed depending on the attack type. For instance, in the case of a drop attack the true value is replaced with a subsequent frame and for a fuzzy attack the true value is replaced with a randomly generated packet.

For a practical intrusion detection mechanism, an anomaly should be detected way before it causes damage in the targeted host. So, we implemented our anomaly detector in a way that can look up data for a range of 1 second to predict a single future packet. But, we could not train the rest of the arbitration except the first four because we couldn't collect enough data that will be used as an input for prediction and due to the fact that LSTM performs better when the look back has a larger value Wang et al. [31], Kong et al. [18]. The neural network is trained to predict the bit flip rate in a range of 75ms seconds for car A. Figure 8 (a) shows the bit flip rate value of all experimented arbitration IDs in car A.

To further check the feasibility of our system, we also collected small data from a different car, car B. Car B has a comparatively higher number of messages, 2,575 packets per second, than car A. For this car, we trained 18 LSTM networks out of 68 arbitration IDs for the same reason we explained above. We drove this particular car for 30 minutes and collected 3.8 million packets. Figure 8 (b) shows the bit flip rate value of all experimented arbitration IDs in car B. We trained both of the vehicles with the 70%, 15%, 15% principle of training data, validation data and testing data sizes. And the testing data is further split into two, the first half for simulating the listed attacks and the other half is left intact.

6. Experiment Results and Discussion

6.1 Experiment Set Up

The data we collected from both cars are captured by manually connecting a Raspberry Pi 3 with PiCAN 3 device to each car. PiCAN 3 board provides CAN-Bus capability for the Raspberry Pi. It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver. For Car A, a total of 130 hours of driving data has been gathered to a sum of more than 300 million packets. Out of all these packets, 180 million of it falls in the category of the four arbitration IDs which we trained for our model. In addition, we used the same device to further prove our model in a second car, car B. For car B, we drove it around for only 30 minutes which counts to a total of 3.8 million packets.

6.2 Evaluation Metrics

We used recall, precision, and F_1 score to show the performance results of our proposed method. These metrics are commonly used in binary classification problems. We also used these results to compare our proposed method with the conventional LSTM-IDS method. Recall shows how good the trained network is at identifying anomalies from a collection of our test data or the ability to find all the relevant cases, anomalies in our case. We used Eq. (11) to calculate the recall where TP stands "true positive" to indicate the anomalies identified as relevant and FN is for "false negative" to show the anomalies that were identified as benign but anomaly packets.

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

And precision shows how much of the identified anomalies were actually anomalies or how much of the identified anomalies where correctly labeled. We used Eq. (12) to calculate the precision. In the equation, false positive (FP) denotes the number of normal records that are identified as an anomaly.

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

Additionally, we used F_1 score to compare the proposed and conventional classification methods. F_1 score shows a measure of a test's accuracy. It considers both precision and recall of the test to compute the score. Equation (13) shows the formula used to evaluate the F_1 score of our experiment.

$$F_1 score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{13}$$

Table 1 Car A performance results for all attack types with an order of precision and recall in each row.

ArbID.	Metrics	Imp.	Drop	Ins.	DoS	Fuzzy
...0027	Precision	0.997	0.997	0.997	0.998	0.997
	Recall	0.921	0.995	1.0	1.0	1.0
...0203	Precision	0.997	0.999	0.999	0.9992	0.999
	Recall	0.461	1.0	0.480	1.0	1.0
...0327	Precision	0.997	0.997	0.997	0.998	0.998
	Recall	0.991	1.0	1.0	1.0	1.0
...8327	Precision	0.995	0.990	0.992	0.991	0.996
	Recall	0.760	0.406	0.4834	0.439	1.0

Table 2 Car B performance results for all attack types with an order of precision and recall in each row.

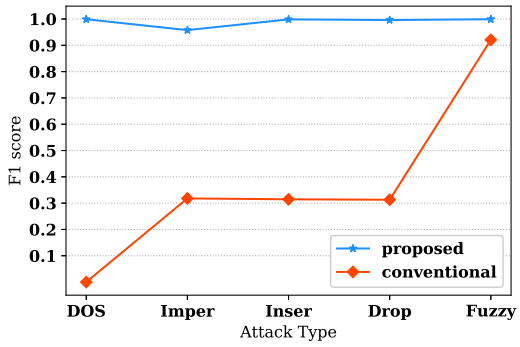
ArbID.	Metrics	Imp.	Drop	Ins.	DoS	Fuzzy
114	Precision	0.993	0.993	0.993	0.994	0.993
	Recall	0.767	1.0	1.0	1.0	1.0
116	Precision	0.931	0.926	0.922	0.939	0.993
	Recall	0.910	0.842	0.805	0.786	1.0
119	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.817	1.0	1.0	1.0	1.0
120	Precision	0.963	0.944	0.879	0.977	0.970
	Recall	0.792	0.515	0.223	0.446	1.0
122	Precision	0.931	0.800	0.8975	0.818	0.943
	Recall	0.818	0.242	0.530	0.198	1.0
124	Precision	0.952	0.913	0.928	0.897	0.957
	Recall	0.901	0.477	0.591	0.283	1.0
130	Precision	0.956	0.429	0.945	0.333	0.971
	Recall	0.667	0.023	0.523	0.015	1.0
131	Precision	0.918	0.545	0.913	0.583	0.929
	Recall	0.847	0.092	0.806	0.107	1.0
180	Precision	0.905	0.906	0.806	0.571	0.917
	Recall	0.857	0.865	0.376	0.120	1.0
18D	Precision	0.934	0.929	0.918	0.941	0.937
	Recall	0.948	0.873	0.754	0.777	1.0
18E	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.777	0.738	1.0	1.0	1.0
18F	Precision	0.922	0.654	0.932	0.820	0.935
	Recall	0.8233	0.131	0.9539	0.209	1.0
1AF	Precision	0.897	0.895	0.870	0.923	0.910
	Recall	0.856	0.841	0.659	0.822	1.0
1B8	Precision	0.895	0.870	0.788	0.907	0.904
	Recall	0.901	0.712	0.394	0.712	1.0
1C0	Precision	0.636	0.867	0.429	0.930	0.890
	Recall	0.216	0.804	0.093	0.848	1.0
1C1	Precision	0.869	0.889	0.611	0.920	0.905
	Recall	0.699	0.842	0.165	0.830	1.0
1D0	Precision	1.0	1.0	1.0	1.0	1.0
	Recall	0.712	1.0	1.0	1.0	1.0
1D1	Precision	0.949	0.900	0.861	0.940	0.964
	Recall	0.705	0.341	0.235	0.406	1.0

6.3 Experimental Results and Discussion

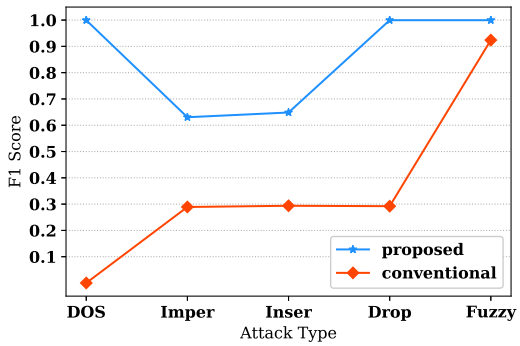
For each arbitration ID, we used different threshold values so as to get the best performance and depending on the property of the corresponding arbitration IDs it gets easier for the method to identify the anomalies for the IDs with a larger number of counter bits than the once which have no counter bits like in arbitration ID_...0203, **Table 1**, which got no counter bit at all, Fig. 4 (a).

Table 1 and **Table 2** show the performance results of our method for each arbitration ID. In the tables, each row contains performance values in an order of precision and recall for all the tested arbitration IDs.

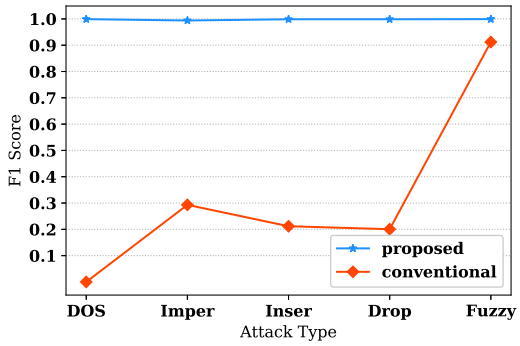
Car A's comparison graphs in **Fig. 9** (a)–Fig. 9 (d) shows how much using our method outperforms the conventional method, which only considers the raw packet bits. Using bit flip rate as an input helps us to smooth out the various transitions the neural network has to learn if we used a raw bit as an input. Moreover,



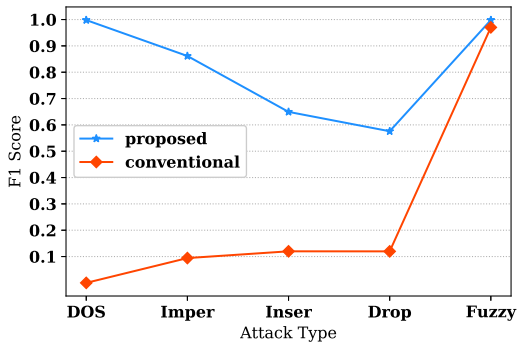
(a) ID_0C000027



(b) ID_0CF00203



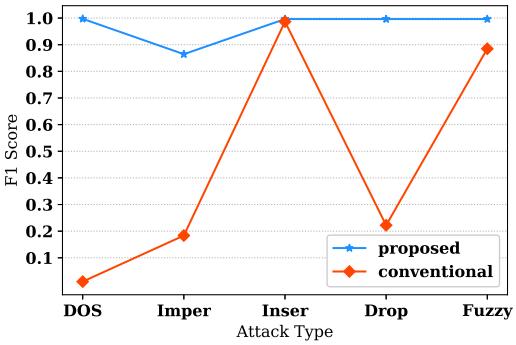
(c) ID_0CF00327



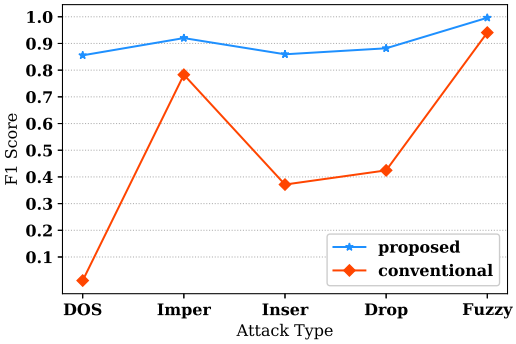
(d) ID_18FF8327

Fig. 9 Car A, comparison graphs between the proposed and the conventional method for all attack types.

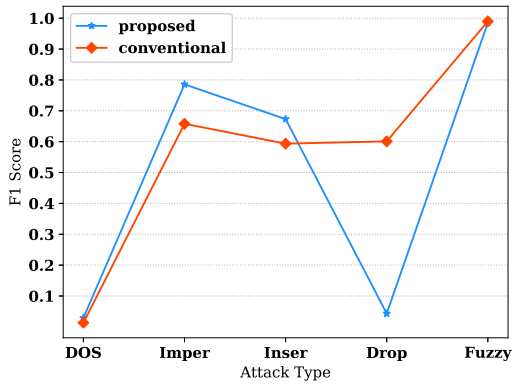
our LSTM's input includes the average time difference and the number of packets to help us get an improved detection accuracy. In the comparison figures, the detection performance of the arbitration IDs with a higher number of counter bits is more ac-



(a) Arbitration ID 114 has 4(2+2)counter bits



(b) Arbitration ID 116 has 2 counter bits



(c) ID 130 has no counter bits

Fig. 10 Car B, F1 score Comparison results between the conventional method and the proposed method.

curate than in the case of no counter bit. **Figure 10** shows car B's F_1 score of comparison between the conventional and proposed methods. The conventional method didn't detect any of the DoS attacks, Fig. 9 and Fig. 10, because their method doesn't consider if messages are arriving at an expected time range. A DoS attack is simulated by delaying packets to a range of 0.05 ms. Delaying these messages doesn't make any difference in the conventional method but the number of messages and the average arrival time variables of the proposed method are affected when messages are delayed. The Proposed method can detect with a precision and recall values as shown in Table 2. The conventional method has no way of telling if a message is delayed due to different attacks, a DoS attack in our case, because the conventional method only considers the packet bits.

As it can be seen in Fig. 9, each arbitration ID has different F1

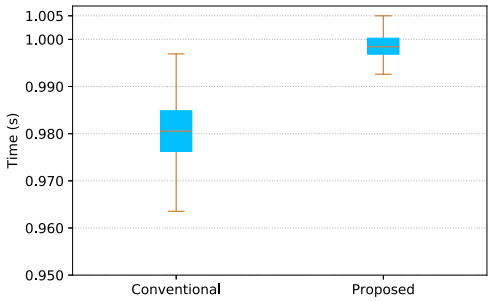


Fig. 11 Execution Time for both conventional and proposed methods.

scores for even the same type of attack. This is because some of the IDs a have higher number of counter bits compared to the others. The trained LSTM predicts better for the IDs with a higher number of counter bits than the ones which have no counter bits.

Implementation of LSTM-IDS in the in-vehicle network would require a fast processor that is capable of collecting bit flip rates in n time window and make a prediction in as short a time as possible. But, there is always a significant delay from the time when the packets appear in the CAN bus to the time when these packets are collected for analysis. **Figure 11** shows the execution time of our detection process. The values in the box plot show the execution time of the model simulated in Ubuntu 18.04 OS, Intel Xeon E5-1620 CPU and GM200 (GeForce GTX TITAN X) 3072 CUDA cores GPU that has a clock speed 33 MHz and a RAM size of 16 GB. We implemented the LSTM-IDS in this computer by using a Oliver Hartkopp [27] that contains CAN drivers and a networking stack for Linux. We played the dumped file in the terminal with the vcan0 interface and created a program that collects the packets through the SocketCAN API and does the prediction. As we can see it in the figure, our proposed method has some delay due to the calculations overhead.

When our system enters the detection phase, one or two messages appear in the CAN bus that will be skipped unless we use a buffering mechanism. For these messages to not be skipped while we are testing a time window we implemented a buffering mechanism that can keep these messages in a buffer until the prediction is completed. Even though this method leaves no packets behind, it also destroys our goal of identifying intrusions in a short time because if packets start to pile up in the buffer it would start to take a longer time to identify the anomalies. Therefore, if the system is going to be deployed in a real vehicle, we recommend two ways. One is to only run a single proposed method's IDS and let the packets that appear during the processing delays be skipped. The other is to run two of the proposed method's IDS concurrently with the second one starting a little late. While the first IDS is checking for intrusions, the second IDS can start collecting next testing sequences starting from the time when the first IDS entered processing.

7. Conclusion

In this paper, we presented on how anomalies can be detected using LSTM for in-vehicle networks. The detection method works for targeted attacks meant for a specific ECU. LSTM networks are trained to predict forthcoming payloads and related attributes by looking at information that has already appeared in the

CAN bus at some instant in time. The predicted values are compared with actual values, that are either sent by an intruder or the benign ones and depending on how close our prediction is with the actual payload, we managed to effectively identify anomalies in an improved accuracy compared to the conventional LSTM-IDS. To train the network we reverse engineered the messages to identify the counter bits associated with each packet which improved the overall detection method to up to 98%. Through the overall evaluation, we proved that further including data from timing and count of packet in a time window has a great advantage in detecting anomalies. Although the proposed method can identify anomalies at a better accuracy than the conventional method, it has poor performance for those arbitration IDs which contain no counter bits.

In this paper, we had to implement an LSTM network for each arbitration ID due to the different DLC lengths associated with each arbitration ID. But for the case of our first car, CAR A, it uses CAN 2.0B and all the messages have the same DLC size. Therefore for our future work, we would implement a single LSTM network or any other related neural network that takes all the arbitration ID's packets as an input to predict a forthcoming packet. This would help us implement an overall IDS that doesn't leave out any packet behind.

References

[1] Bengio, Y., Simard, P. and Frasconi, P.: Learning long-term dependencies with gradient descent is difficult, *IEEE Trans. Neural Networks*, Vol.5, No.2, pp.157–166 (1994).

[2] Bosch, R.: Bosch can specification (Sep. 1991).

[3] Casillo, M., Coppola, S., De Santo, M., Pascale, F. and Santonicola, E.: Embedded intrusion detection system for detecting attacks over can-bus, *2019 4th International Conference on System Reliability and Safety* (*ICSRS*), pp.136–141, IEEE (2019).

[4] Chollet, F. et al.: Keras (2015), available from ⟨https://keras.io⟩.

[5] Corrigan, S.: Introduction to the controller area network (can), Texas Instruments (2016).

[6] Steve Corrigan: Introduction to the controllerareanetwork (can) (2016), available from ⟨http://www.ti.com/lit/an/sloa101b/sloa101b.pdf⟩ (accessed 2019-02-26).

[7] Farag, W.A.: Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms, *2017 7th International Conference on Modeling, Simulation, and Applied Optimization* (*ICMSAO*), pp.1–5, IEEE (2017).

[8] Foruhandeh, M., Man, Y., Gerdes, R., Li, M. and Chantem, T.: Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks, *Proc. 35th Annual Computer Security Applications Conference*, pp.229–244 (2019).

[9] Gers, F.A., Schmidhuber, J. and Cummins, F.: Learning to forget: Continual prediction with lstm, *9th International Conference on Artificial Neural Networks: ICANN*, pp.850–855 (1999).

[10] Graves, A.: Supervised sequence labelling, *Supervised Sequence Labelling with Recurrent Neural Networks*, pp.5–13, Springer (2012).

[11] Groza, B. and Murvay, P.S.: Security solutions for the controller area network: Bringing authentication to in-vehicle networks, *IEEE Vehicular Technology Magazine*, Vol.13, No.1, pp.40–47 (2018).

[12] Groza, B., Murvay, S., Van Herrewege, A. and Verbauwhede, I.: Libra-can: A lightweight broadcast authentication protocol for controller area networks, *International Conference on Cryptology and Network Security*, pp.185–200, Springer (2012).

[13] Halabi, J. and Artail, H.: A lightweight synchronous cryptographic hash chain solution to securing the vehicle can bus, *2018 IEEE International Multidisciplinary Conference on Engineering Technology* (*IMCET*), pp.1–6, IEEE (2018).

[14] Han, G., Zeng, H., Li, Y. and Dou, W.: Safe: Security-aware flexray scheduling engine, *2014 Design, Automation & Test in Europe Conference & Exhibition* (*DATE*), pp.1–4, IEEE (2014).

[15] Hazem, A. and Fahmy, H.A.: Lcap-a lightweight can authentication protocol for securing in-vehicle networks, *10th escar Embedded Security in Cars Conference, Berlin, Germany*, Vol.6 (2012).

[16] Nicomette, V., Kaaniche, M., Laarouchi, Y., Studnia, I. and Alata, E.: A language based intrusion detection approach for automotive embedded networks, *International Journal of Embedded Systems, Inderscience* (2018).

[17] Kneib, M. and Huth, C.: Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks, *Proc. 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp.787–800 (2018).

[18] Kong, W., Dong, Z.Y., Jia, Y., Hill, D.J., Xu, Y. and Zhang, Y.: Short-term residential load forecasting based on lstm recurrent neural network, *IEEE Trans. Smart Grid*, Vol.10, No.1, pp.841–851 (2017).

[19] Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al.: Experimental security analysis of a modern automobile, *2010 IEEE Symposium on Security and Privacy*, pp.447–462, IEEE (2010).

[20] Kuwahara, T., Baba, Y., Kashima, H., Kishikawa, T., Tsurumi, J., Haga, T., Ujiie, Y., Sasaki, T. and Matsushima, H.: Supervised and unsupervised intrusion detection based on can message frequencies for in-vehicle network, *Journal of Information Processing*, Vol.26, pp.306–313 (2018).

[21] Goodfellow, I., Bengio, Y. and Courville, A.: *Deep Learning* (2016).

[22] Marchetti, M. and Stabili, D.: Read: Reverse engineering of automotive data frames, *IEEE Trans. Information Forensics and Security*, Vol.14, No.4, pp.1083–1097 (2018).

[23] Miller, C. and Valasek, C.: Remote exploitation of an unaltered passenger vehicle, White Paper of *Black Hat USA Conference* (2015).

[24] Mundhenk, P., Steinhorst, S., Lukasiewycz, M., Fahmy, S.A. and Chakraborty, S.: Lightweight authentication for secure automotive networks, *2015 Design, Automation & Test in Europe Conference & Exhibition* (*DATE*), pp.285–288, IEEE (2015).

[25] Nanduri, A. and Sherry, L.: Anomaly detection in aircraft data using recurrent neural networks (RNN), *2016 Integrated Communications Navigation and Surveillance* (*ICNS*), p.5C2-1, IEEE (2016).

[26] Keen Security Lab of Tencent: Keen security lab blog, car hacking research (2016), available from ⟨https://keenlab.tencent.com⟩ (accessed 2019-02-26).

[27] Thuermann, U. et al.: Oliver Hartkopp, Controller area network protocol family (aka socketcan), available from ⟨https://www.kernel.org/doc/Documentation/networking/can.txt⟩ (accessed 2019-02-26).

[28] Song, H.M., Kim, H.R. and Kim, H.K.: Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network, *2016 International Conference on Information Networking* (*ICOIN*), pp.63–68, IEEE (2016).

[29] Taylor, A., Japkowicz, N. and Leblanc, S.: Frequency-based anomaly detection for the automotive can bus, *2015 World Congress on Industrial Control Systems Security* (*WCICSS*), pp.45–49, IEEE (2015).

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

[31] Wang, Q., Guo, Y., Yu, L. and Li, P.: Earthquake prediction based on spatio-temporal data mining: An lstm network approach, *IEEE Trans. Emerging Topics in Computing* (2017).

[32] Woo, S., Jo, H.J. and Lee, D.H.: A practical wireless attack on the connected car and security protocol for in-vehicle can, *IEEE Trans. Intelligent Transportation Systems*, Vol.16, No.2, pp.993–1006 (2014).

[33] Wu, W., Huang, Y., Kurachi, R., Zeng, G., Xie, G., Li, R. and Li, K.: Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks, *IEEE Access*, Vol.6, pp.45233–45245 (2018).

[34] Wu, Y., Kim, Y.J., Piao, Z., Chung, J.-G. and Kim, Y.-E.: Security protocol for controller area network using ecandc compression algorithm, *2016 IEEE International Conference on Signal Processing, Communications and Computing* (*ICSPCC*), pp.1–4, IEEE (2016).

Araya Kibrom Desta is currently pursuing Ph.D. degree with the Graduate School of Science and Technology, Nara Institute of Science and Technology. He received M.E. from the Graduate School of Information Science, Nara Institute of Science and Technology in 2019. His research interests include intrusion detection systems and cyber-physical systems security.

Shuji Ohira is currently pursuing Ph.D. degree with the Graduate School of Science and Technology, Nara Institute of Science and Technology. He received M.E. degree from the Graduate School of Science and Technology, Nara Institute of Science and Technology in 2020. His research interests include embedded system security and cyber-physical systems security.

Ismail Arai is an associate professor affiliated with Information Initiative Center, Nara Institute of Science and Technology since 2016. He received M.E. and Ph.D. in engineering from the Graduate School of Information Science, Nara Institute of Science and Technology in 2004 and 2008 respectively. From 2008 to 2011, he was a postdoctoral fellow with The Research Organization of Science and Engineering, Ritsumeikan University. At Department of Electrical and Computer Engineering, National Institute of Technology, Akashi College, he was an assistant professor, a lecturer, and an associate professor 2011–2013, 2013–2015, 2015–2016 respectively. He is a member of IPSJ, IEICE, ACM, and IEEE. His research topics are transportation data analysis, pedestrian navigation systems, cybersecurity, and open data.

Kazutoshi Fujikawa received M.E. and Ph.D. degrees in information and computer sciences from Osaka University in 1990 and 1993, respectively. Currently, he is a professor of Information Initiative Center, Nara Institute of Science and Technology. His research focuses on multimedia communication systems, digital libraries, ubiquitous computing, and mobile networks. He is a member of IPSJ, IEICE, ACM, and IEEE.