Anomaly Detection Methodology of In-vehicle Network Based on Graph Pattern Matching

Mengsi Sun[1], Jiarun Wu[2]

[1] College of information science and technology, Jinan University, China

Correspondence: Mengsi Sun, College of information science and technology, Jinan University, China. E-mail: sunmengsi.jnu@gmail.com

Received: April 1, 2022 Accepted: April 22, 2022 Online Published: April 27, 2022

doi:10.5539/cis.v15n2p78 URL: https://doi.org/10.5539/cis.v15n2p78

Abstract

Vehicles are becoming more and more connected today, with many direct interfaces and infotainment units widely deployed in in-vehicle networks. However, the increase in interfaces and units can also lead to an increase in cyberattacks surfaces. As the de facto standard for the in-vehicle network protocol, the Controller Area Network (CAN) protocol provides an efficient, stable, and cost-effective communication channel between electric control units (ECUs). Nonetheless, it is increasingly threatened by cyberattack due to the lack of security mechanisms by design. This paper proposes a novel anomaly detection methodology based on graph pattern matching, which expresses CAN traffic in terms of graph structures. Given a base graph and window graph, we determine whether the window graph represents normal or anomaly by using the distance measure on the base graph. We have validated this anomaly detection methodology on public datasets and in an actual vehicle environment. Experimental results show that this methodology significantly improved the detection of unknown attacks and outperforms other CAN traffic-based approaches.

Keywords: in-vehicle network, cyberattack, anomaly detection, graph pattern matching

1. Introduction

Nowadays, the vehicle is changing from a traditional driving tool to a service system that provides people with many conveniences. The trend of vehicle intelligence is becoming more and more prominent. Modern vehicles are equipped with many electronic control units (ECUs), which manage the various functions of the vehicle and replace mechanical control units. In addition, the ECUs exchange information using in-vehicle networks such as Controller Area Network (CAN), LIN, MOST, and FlexRay. Among them, CAN is well-known and most deployed as the de facto standard for in-vehicle networks, which is designed by BOSCH and first used in cars in 1991 (Bosch & Robert, 1991). However, while CAN provides an efficient, stable, and cost-effective communication channel between ECUs, it lacks security features and is vulnerable to cyber threats. More direct interfaces and infotainment units have been deployed widely inside networks, making the vehicles more vulnerable to different attacks from inside and outside the vehicle. As a result, adversaries can easily connect devices to the CAN bus, sniff CAN traffic, and launch the attacks via On-Board Diagnostics port (OBD-II) (Mekki, Bouhoute, & Berrada, 2019), WIFI (Woo, Jo, & Dong, 2014), and Bluetooth (Pu, Zhu & et al., 2020), etc. Therefore, CAN security has become a genuine concern for vehicle manufacturers.

As potential attack surfaces inside vehicles continue to increase, more vulnerabilities and entry points are being discovered, and many security solutions for CAN bus have been proposed, such as message encryption (Cui, & Chen et al., 2020), message authentication (Jo, Jin, & Choi et al., 2019), and intrusion detection (Mter, & Asaj, 2011). The intrusion detection method operates as a countermeasure inside the vehicle. It works in both a passive and active manner and does not require changes to existing network and protocol specifications. Hence, it receives more widespread use than message encryption and authentication. Many state-of-art intrusion detection approaches have been proposed to secure CAN (Song, Kim, 2021, & Cheng, Bai et al., 2020), including message periodicity monitor, payload check, ECU fingerprinting approaches, and so on. CAN traffic has been the main focus of most researchers in their approaches because most of the message frames on the CAN bus are transmitted periodically. In addition, CAN traffic is more accessible to obtain than other data information (e.g., payload, signals, etc.).

This paper also proposes an anomaly detection methodology based on CAN traffic features. It is worth mentioning that the fundamental idea of the proposed methodology is to represent CAN traffic with the graph structure, unlike the traditional approaches. Graph-based anomaly detection techniques have been used widely in industries like finance, fraud detection, computer and social networking, data center monitoring, etc. (Akoglu, Tong, & Koutra, 2015), and have given good detection performance. The methodology based on graph pattern matching proposed in this paper consists of two phases, i.e., offline learning and online detection phases. In the offline learning phase, the proposed methodology constructs a base graph based on normal CAN traffic, which serves as a criterion for graph pattern matching, and obtains the ground truth values for detecting an anomaly. In the online detection phase, the proposed methodology constructs the CAN traffic in the sliding window to be detected as the window graph. Given the base graph and window graph, the proposed methodology performs graph pattern matching based on the graph distance defined in this paper and compares the distance value with the ground truth value to derive the detection results. In addition, graphs detected as normal are merged into the base graph in the online detection phase, i.e., the base graph is continuously improved during the real-time detection process. Experimental results show significant success in detecting different kinds of attacks with the proposed methodology.

The main contributions of this paper can be summarized as follows.

1. This paper uses graph pattern matching for the first time to detect the CAN bus's anomaly and defines the graph distance for graph pattern matching.
2. This paper implements and validates the proposed methodology not only on a public dataset but also on a real vehicle dataset, and experiment results demonstrate that the proposed methodology can effectively detect six types of attacks, and outperforms other CAN traffic-based approaches in terms of detection accuracy, precision, FPR, and F1-score.

The rest of this paper is organized as follows. Section 2 analyzes the current status of research on CAN bus security. Then Section 3 details the various stages in the proposed methodology, and experimental validation is presented in Section 4. Subsequently, compatibility and limitation issues about the proposed methodology are discussed further in Section 5, followed by the conclusion and future work in Section 6.

2. Related Work

Each CAN message in the data link layer can extract four types of main information: timestamp, CAN ID, DLC and payload. So far, researchers have proposed various types of anomaly-based intrusion detection approaches by leveraging this information.

Timestamp: For the timestamp information, the time interval feature can be extracted. Song *et al.* (Song, Kim, & Kim, 2016) propose a lightweight detection method based on CAN message transmission time interval, which detects attacks by monitoring whether the change of time interval exceeds the threshold range. Cho *et al.* (Cho, Shin, 2016) propose taking the time interval of periodic CAN message as the fingerprint of sending ECU, and modeling and analyzing it as the basis for anomaly detection.

CAN ID: For the timestamp information, the CAN ID frequency, information entropy, similarity and ID location relationship features can be extracted. Attacks can be detected through monitoring these feature values, as they are unique across the network. Taylor *et al.* (Taylor, Japkowicz, & Leblanc, 2015) use the invariance of message frequency in the time window as the baseline for anomaly detection. Müter *et al.* (Müter, Asaj, 2011) proposed calculating the information entropy of ID in the time window as the feature of anomaly detection. When an injection attack occurs on the CAN bus, the change of information entropy will exceed the normal range. Kwak *et al.* take the cosine similarity between ID sequences on the calculated CAN bus as the feature of anomaly detection, and realized the effective detection of flooding attacks and fuzzy attacks. Marchetti *et al.* (Marchetti, Stabili, 2017) are the first to use the features of ID sequence as the parameters of anomaly detection, and they establish the transmission matrix according to the location relationship of the message ID, and regarded the ID pair that did not appear as an exception. In addition, the feature changes in ID value can also be used as the detection parameter. For example, a message with ID 0x000 is malicious since it can be used to occupy the bus and perform DoS attack.

DLC: A 4-bit field is used to identify the length of the data payload. Attacks can be detected by checking their value and range, as each ECU uses fixed byte size in the payload (Aliwa, Rana, & Perera et al., 2021).

Payload: For CAN ID information, payload range, fixed-rate and data consistency features can be extracted. Attacks can be detected if anomaly values and changes occur in the data field. Pawelec *et al* (Pawelec, Bridges, & Combs, 2019). model the 64-bit payload field for each AID and use prediction error to identify anomalies. Stabili *et al.* (Stabili, Marchetti, & Colajanni, 2017) propose Hamming distance-based detection approach. They analyze CAN payload and record each bit in the data field, and attacks are identified based on significant deviation from the calculated Hamming distance function.

The features in CAN ID can be extracted even without knowing the message specification; that is, we do not require knowledge of the syntax and semantics of the CAN messages, which are kept highly confidential by carmakers and their suppliers. Therefore, CAN ID is more often used as the parameter for anomaly detection in CAN bus.

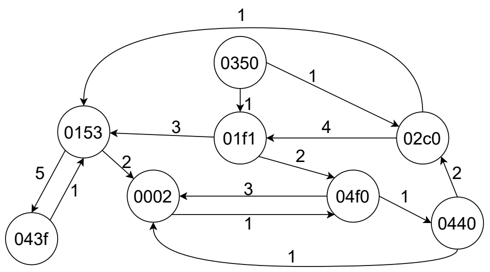


Figure 1. CAN traffic graph at a sliding window. The labels of the nodes are the message IDs, and the labels of the edges are number of times a message with the ID source of the edge is followed by a message with the ID destination of the same edge during the sliding window. For example, four messages with ID '01f1' followed messages with ID '02c0' at resp

3. Method

In this section, we present the framework of the proposed methodology. We first introduce the basic definitions involved in the proposed methodology and formally describe our problem. Secondly, we describe in detail the design of the proposed methodology at different phases, including the framework and algorithms, etc. Finally, we explain the definition of the detection time of the scheme.

3.1 Problem Definition

In this paper, we focus on a directed graph g with vertices and edges, and denote it as (V, E, LE, LV), where V is the set of vertices, and $E(\subseteq V \times V)$ is a set of edges. LV and LE are functions that assign labels for each vertex $u \in V(g)$ and edge $(u, v) \in E(g)$, respectively.

Definition 1 (Graph isomorphism). Given two graphs g_1 and g_2, g_1 and g_2 are isomorphic if there is a bijective function f: $V(g1) \to V(g2)$ such that $\forall u \in V(g1), f(u) \in V(g2) \wedge LV_1(u) = LV_2(f(u))$, and $\forall (u, v) \in E(g1), (f(u), f(v)) \in E(g2) \wedge LE_1((u, v)) = LE_2((f(u), f(v)))$.

Definition 2 (Maximum common subgraph). Given two graphs q and g, the maximum common subgraph (MCS), denoted as $MCS(q, g)$, is the connected subgraph s with the maximum number of edges common to the two graphs q and g, where s is isomorphic to the subgraph of each of the two given graphs.

Definition 3 (CAN traffic graph). A directed graph consisting of a sequence of CAN messages within a sliding window is denoted as CTG. As shown in Figure 1, the vertices of the CTG is a message ID, the position relationship between the message IDs in the sequence forms the edges of the CTG, and the label of the edge is the number of times a message with the ID source of the edge is followed by a message with the ID destination of the same edge during the sliding window.

Definition 4 (Base graph). A CAN traffic graph serves as a criterion to calculate the graph distance and is denoted as bg. Moreover, bg is continuously updated as the detection process proceeds.

Definition 5 (Window graph). A CAN traffic graph represents a graph constructed from CAN messages within the current sliding window and denoted as wg.

3.2 Overview of Anomaly Detection Framework

This paper proposes a novel methodology based on graph pattern matching that leverages CAN traffic features to detect attacks on the CAN bus. The graph distance is a metric for a matching pattern between two graphs. The graph distance, denoted as $Gdist(bg, wg)$, is defined as follows:

$$Gdist(bg, wg) = |E(wg)| - |E(MCS(bg, wg))| \tag{1}$$

where, both bg and wg has introduced in the problem definition. $|E()|$ is the function that calculates the number of graph edges, i.e., the sum of the label values of all edges. $MCS(bg, wg)$ is the maximum common subgraph of graph bg and graph wg, described in detail in the problem definition. $Gdist(bg, wg)$ indicates how many edges of the wg are not present in the bg.

As shown in Figure 2. the proposed methodology consists of two phases: one phase is the offline learning phase, in which we obtain the ground-truth value for the selected sliding window and the base graph bg for pattern matching. The other is the online detection phase, in which we use the base graph obtained in the offline learning phase to calculate graph distance with the window graph wg to detect anomalies in real-time. The detail is as follows.

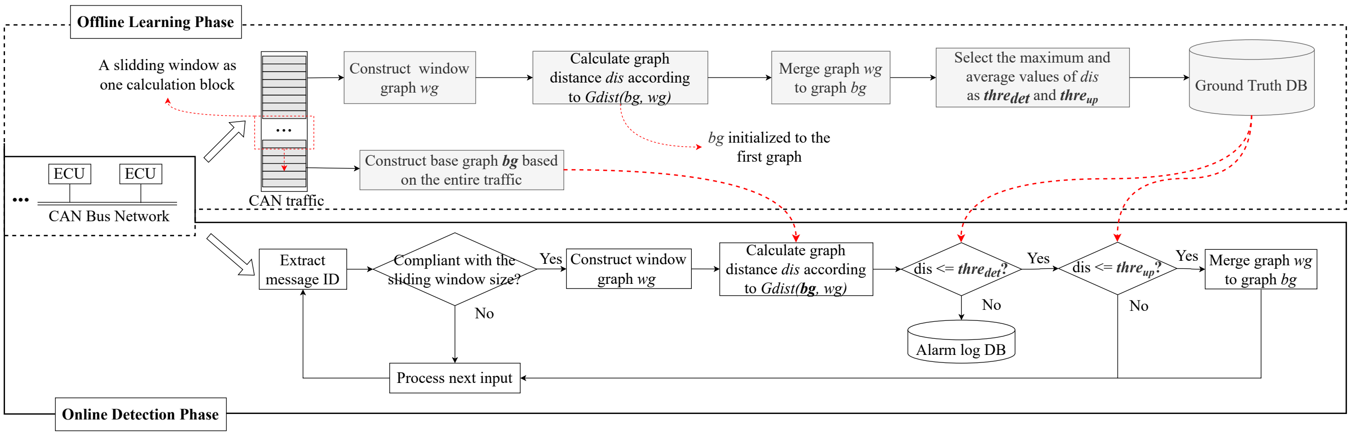

Offline Learning Phase
A slidding window as one calculation block
Construct window graph wg
Calculate graph distance dis according to Gdist(bg, wg)
Merge graph wg to graph bg
Select the maximum and average values of dis as thre_det and thre_up
Ground Truth DB
bg initialized to the first graph
Construct base graph bg based on the entire traffic
ECU
ECU
CAN Bus Network
CAN traffic
Extract message ID
Compliant with the sliding window size?
Yes
No
Construct window graph wg
Calculate graph distance dis according to Gdist(bg, wg)
dis <= thre_det?
Yes
No
dis <= thre_up?
Yes
No
Merge graph wg to graph bg
Alarm log DB
Process next input
Online Detection Phase

Figure 2. Overview of the anomaly detection methodology based on graph pattern matching in the CAN bus

Phase 1(Offline learning phase). This phase divides the normal CAN traffic (without attack messages) extracted from the CAN bus network into multiple equal-sized sliding windows and transforms the message ID sequence within the current window into a window graph wg. To obtain the ground truth, we first obtain the graph distance dis_k at t_k according to $Gdist(bg, wg)$, and calculate the $Gdist_{k+1}$ at t_{k+1} after the sliding window was updated, where bg is initialized to the first graph and continuously updated by merging new wg as the sliding window moves. The set of graph distances was $S(x) = dis_1, dis_2, , dis_n$, and we selected the value of max dis and average dis in $S(x)$ as the $thre_{det}$ and $thre_{up}$. Afterward, we saved the value in a DB as a Ground Truth in the predefined sliding window. In addition, in this phase, we construct the entire message ID sequence as a base graph, which is used to initialize the base graph in the online detection phase.

Algorithm 1: Construct CAN Traffic Graph (Offline Learning Phase)

```
   Input: CAN traffic within a sliding window $Tr$
   Output: a CAN traffic graph $g$
1  $pre\_msg \leftarrow msg_0$;
2  for each $msg \in Tr$ do
3      $cur\_id \leftarrow$ extract_CANID ($msg$);
4      $pre\_id \leftarrow$ extract_CANID ($pre\_msg$);
5      if $cur\_id \notin V(g)$ then
6          add a new vertex $cur\_id$ in $V(g)$;
7      end
8      if $(pre\_id, cur\_id) \notin E(g)$ then
9          add an edge $(pre\_id, cur\_id)$ in $E(g)$;
10     else
11         The label value of the edge $(pre\_id, cur\_id)$ is added by 1
12     end
13     $pre\_msg \leftarrow msg$;
14 end
15 return graph $g$;
```

Algorithm 1 shows the method of CAN traffic graph construction. Given CAN traffic within a sliding window, we first initialize pre_msg that is used to refer to the previous message. The ID of each message is extracted and added to the $V(g)$ as a vertex (Line 3-7). Then, an edge consisting of the previous message ID as the source vertex and the current message ID as the destination vertex is added to $E(g)$ (Line 8-12).

Phase 2(Online learning phase). In this phase, first, the ID of the CAN message read from the bus is extracted and stored, and this operation is repeated until the number of messages read satisfies the size of the sliding window we define. Second, the ID traffic within the sliding window is constructed into a CAN traffic graph according to Algorithm 1 and stored in the wg. Finally, the graph pattern matching is performed, i.e., the graph distance dis between bg and wg is calculated

Algorithm 2: Anomaly Detection Algorithm (Online Detection Phase)

```
Input: CAN message; the size of sliding window $window\_size$ ; ground truth $thre_{det}$; ground truth $thre_{up}$; Base
       graph $bg$
Output: Anomaly Detection Alarm
while $True$ do
    $i \leftarrow 0; S \leftarrow []$;
    $wg \leftarrow$ CAN traffic graph;
    while $i < window\_size$ do
        add current CAN message into $S$;
        $i \leftarrow i + 1$;
    end
    $wg \leftarrow$ Algorithm 1($S$);
    $dis \leftarrow Gdist(bg, wg)$;
    if $dis >= thre_{det}$ then
        return Anomaly Detection Alarm
    else
        if $dis <= thre_{ud}$ then
            for each $vertex \in V(wg)$ do
                if $vertex \notin V(bg)$ then
                    add this vertex in $V(bg)$;
                end
            end
            for each $edge \in E(wg)$ do
                if $edge \notin E(bg)$ then
                    add this edge in $E(bg)$;
                end
            end
        end
    end
end
return Anomaly Detection Alarm;
```

according to Equation 1. The condition to identify whether the window graph *wg* is normal or not is as follows:

$$wg = \begin{cases} dis \geq thre_{det}, & anomaly \\ dis < thre_{det}, & normal \end{cases} \tag{2}$$

where $thre_{det}$ is the ground-truth obtained from the offline learning phase. If the *wg* is detected as an anomaly, the detector sends the alert message to a driver or a manager and records the anomaly log at the Alarm Log DB. Otherwise, the detector determines again whether the *wg* can be used to update the *bg*, and the judgment conditions are as follows:

$$bg = \begin{cases} dis > thre_{ud}, & bg \\ dis \leq thre_{ud}, & wg + bg \end{cases} \tag{3}$$

where $thre_{ud}$ is also the ground-truth, and $wg + bg$ means to add vertices and edges in *wg* that do not exist in *bg* to *bg*. This condition means that if *dis* is not greater than $thre_{ud}$, *bg* is updated by merging *wg* to *bg*, otherwise, *bg* is not updated.

Algorithm 2 shows the process of detecting anomalies online in detail. Lines 4-7 correspond to the extraction and storage of CAN IDs, and Line 8 is the CAN traffic graph *wg* construction. In line 9, the graph distance *dis* between *wg* and *bg* is obtained. In Line 10-25, whether *wg* is normal is judged, and in line 13-24, when *dis* is not great than $thre_{ud}$, vertices and edges in *wg* that do not exist in graph *bg* will be added to *bg*, that is, merge *wg* to *bg*.

4. Results

In this section, we verify the feasibility of the proposed methodology through different experiments. First, we describe in detail the dataset used for the experiments. Second, we introduce the performance metrics used to evaluate the proposed methodology. Third, we perform detection experiments for different types of attacks and give the corresponding analysis of the experimental results. Fourth, we consider the computation overhead of the proposed methodology, and finally, we present a comparative analysis of the methodology with related work.

4.1 Dataset

Table 1. Description of validation dataset

DataSet	Attack Type	Total Messages	Normal Messages	Attack Messages
Car-Hacking (public)	Attack-free (normal)	988,987	988,872	–
	DoS Attack	3,665,771	3,078,250 (84%)	587,521 (16%)
	Fuzzy Attack	3,838,860	3,347,013 (87%)	491,847(13%)
	Gear Spoofing	4,443,142	3,845,890 (87%)	597,252 (13%)
	RPM Spoofing	4,661,702	3,966,805 (85%)	654,897 (15%)
Car-Intrusion (private)	Attack-free (normal)	90,808	90,808	–
	Flooding Attack	202,667	98,637 (49%)	104,030 (51%)
	Injection Attack	200,511	76,146 (38%)	124,365 (62%)
	Fuzzy Attack	268,324	93,817 (35%)	174,507 (65%)

In this paper, to verify the validity of the proposed methodology, we chose two different datasets for validation, one is a public dataset *Car-Hacking*[1], and the other is a private dataset *Car-Intrusion* collected in an actual vehicle environment by us.

Car-Hacking Dataset: The dataset is provided by Song *et.al* (Song, Woo, & Kim, 2020), and has been widely adopted by many excellent approaches, such as (Seo, Song, & Kim, 2019, Song, Kim, 2021). The dataset is generated by logging CAN traffic via OBD-II port. Each CAN message is represented by three main features:

- CAN ID: the identifier of CAN message.
- DATA[0-7]: the 8 data bytes of the message.
- Flag: it takes two values: T and R, which represent attack and normal messages respectively.

The dataset consists of attack-free traffic and the following types of attacks:

- DoS Attack: Injecting messages of CAN ID = 0000 every 0.3 ms.
- Fuzzy Attack: Injecting messages of totally random CAN ID and DATA values every 0.5 ms.
- Gear/RPM Spoofing Attack: Injecting messages of certain CAN ID related to Gear/RPM information every 0.1 ms.

Car-Intrusion Dataset: The dataset is generated in a real vehicle (XiaoPeng P5). We connect two PCs to the OBD-II port, gateway of the vehicle respectively through ValueCAN (a high-quality CAN network interface tool). As an attacker node, one PC generates and injects the attack messages to the CAN bus through gateway. The other PC collected one of the subnets traffic to monitor the status of the CAN bus.

Each CAN message also has three features consistent with *Car-Hacking*. The dataset consists of attack-free traffic and the following types of attack:

- Flooding Attack: Injecting messages of CAN ID = '0x1AF' every 0.1 ms.
- Injection Attack: Injecting messages of certain CAN ID stored in the DBC file every 0.2 ms.

[1]https://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset.

- Fuzzy Attack: Injecting messages of totally random CAN ID and DATA values every 0.3 ms.

Table 1 shows the detailed number of each attack type in these two datasets.

4.2 Performance Metrics

To evaluate the proposed methodology, we use different performance metrics, i.e. accuracy, recall, False Positive Rate (FPR), precision, and F1-score, which are defined by Equations 4, 5, 6, 7, 8 respectively. We define True Positive (TP) is the number of attack graphs classified as 'attack', False Positive (FP) is the number of normal graphs classified as 'attack', False negative (FN) is the number of attack graphs classified as 'normal', True negative (TN) is the number of normal graphs classified as 'normal'.

- Accuracy: It is the proportion of correctly classified samples among all classified samples, and calculated as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

- Recall: It is the proportion of correctly classified attack samples among the actual attack samples, and calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

- FPR: It is the proportion of samples classified as normal that are incorrectly classified as attack samples, which is calculated as follows:

$$FPR = \frac{FP}{FP + TN} \tag{6}$$

- Precision: It is the proportion of real attack samples among the samples classified as an attack, and calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

- F1-score: It represents the balance between precision and recall, and is generally used to measure classification performance when the class distribution is imbalanced in the test dataset, and calculated as:

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{8}$$

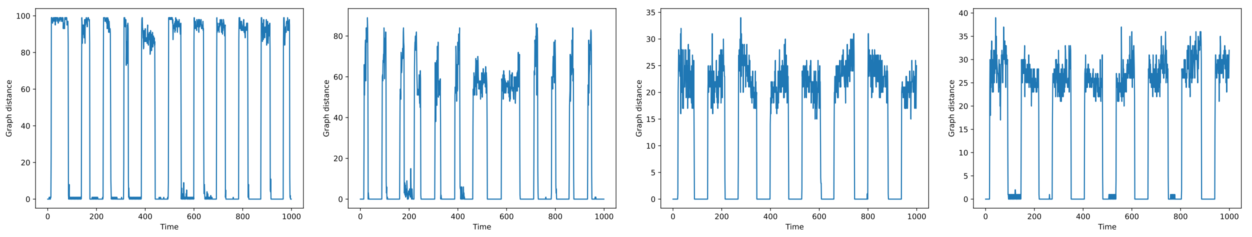
4.3 Anomaly Detection

Table 2. Experiment results

Dataset	Attack type	FPR(%)	Recall(%)	Precision(%)	F1-score(%)	Accuracy(%)
Car-Hacking	DoS Attack	0.0	98.98	100	99.49	99.69
	Fuzzy Attack	0.0	98.54	100	99.26	99.27
	Gear Spoofing	0.0	98.68	100	99.33	99.43
	RPM Spoofing	0.0	98.82	100	99.41	99.47
Car-Intrusion	Flooding Attack	0.0	99.69	100	99.85	99.8
	Injection Attack	0.0	99.88	100	99.94	99.9
	Fuzzy Attack	0.0	99.76	100	99.88	99.81

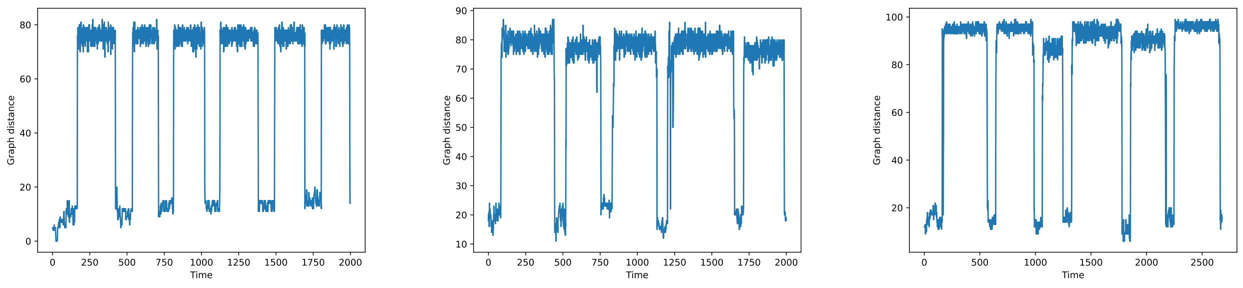
In this section, we conduct experiments on a public dataset and a real vehicle environment, respectively, to verify the reliability of the proposed methodology. Figure 3 and Figure 4 show partial experimental results based on the dataset *Car-Hacking* and *Car-Intrusion*, respectively, and when the window size is set to 100. It is easy to observe that the graph distance value increases sharply once the attack occurs, and when the attack ends, the graph distance value decreases and normalizes. In addition, the graph distance values calculated when an attack occurs are much larger than those calculated in the normal state, so it is easy to set thresholds to distinguish between normal and anomaly.

Table **??** describes the specific experimental results corresponding to the above experiments. The experiment data show that the proposed methodology achieves good results for six different attacks, both in publicly available datasets and in



(a) (b) (c) (d)

Figure 3. Partial experimental results of the proposed methodology under *Car-Hacking* dataset with window size 100: (a) DoS attack. (b) Fuzzy attack. (c) Gear Spoofing attack. (d) RPM Spoofing attack.



(a) (b) (c)

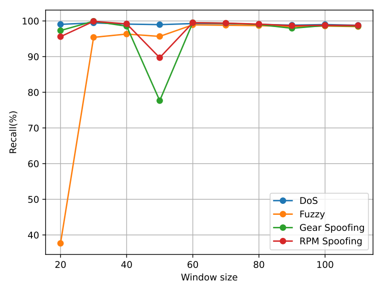
Figure 4. Partial experimental results of the proposed methodology under *Car-Intrusion* dataset with window size 100. (a) Flooding attack. (b) Injection attack. (c) Fuzzy attack.

a dataset collected in the real vehicle environment. For example, for all listed attack types, the proposed methodology achieves 100% of precision and 0% of FPR, and both F1-score and accuracy are above 98%. Even the worst TPR value achieves greater than 98.5%.

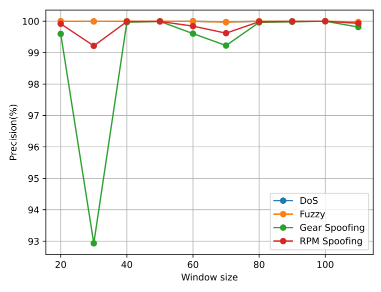
Furthermore, we further analyze the influence of the size of the sliding window on the detection performance. Figure 5 and Figure 6 present the analysis results under *Car-Hacking* and *Car-Intrusion* datasets, respectively. The results are presented as a function of Window Size and varying attack types. The first observation we can draw from Figure 5 is that the value of recall, precision, accuracy, and F1-score is very close to 100%, and the value of FPR is very close to 0% when Window Size = 40. The effectiveness results converge to the optimal value starting from Window Size = 40. When Window Size is $\in [60, 110]$, we record values greater than 99.5% in terms of recall, precision, accuracy, and F1-score metrics, and values less than 1% in terms of FPR, and this high-performance state is always maintained. In Figure 6, we can observe that the *Car-Intrusion* dataset is less challenging than *Car-Hacking* as the optimal performance result is reached when the Window Size is $\in [40, 100]$ except for two metrics, i.e., precision (Window Size $\in [70, 90]$, attack type = Flooding and Fuzzy) and FPR (Window Size $\in [70, 90]$, attack type = Flooding and Fuzzy). However, the results of these two metrics under the above parameter settings are all >= 98.04%, <= 7.58%, respectively.

4.4 Time for Detection

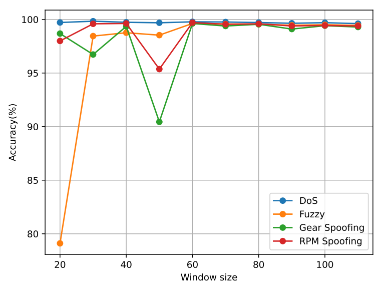
Figure 7 shows the grouped bar diagrams as the results of running the algorithm depending on the dataset and attack type. In calculating the detection time, the laptop specifications include Intel(R) Core(TM) i5-1135G7 CPU @2.40 GHz and Windows 10 for the operating system. For developing the algorithm, we used Python 3.9.2 for code compilation. Figure 7(a) shows the minimum, average and maximum time consumption of the proposed methodology under the dataset *Car-Hacking*. In the best case, the time cost in detecting DoS, Fuzzy, Gear Spoofing, and RPM Spoofing attacks can be as low as 0.021, 0.028, 0.029, and 0.029 ms, respectively. Typically, the average time cost to detect these attacks can reach 0.027, 0.036, 0.036, and 0.035 ms, respectively. In the worst case, the maximum values of time cost are only 0.124, 0.366, 0.28, 0.137 ms, respectively. Figure 7(b) also details the time consumption of the proposed methodology, but under dataset *Car-Hacking*. This figure shows that even in the real vehicle environment, the average and maximum values of detection time in detecting Flooding, Injection, and Fuzzy attacks are only 0.027, 0.026, 0.023 ms, and 0.082, 0.065, and 0.139 ms, respectively.



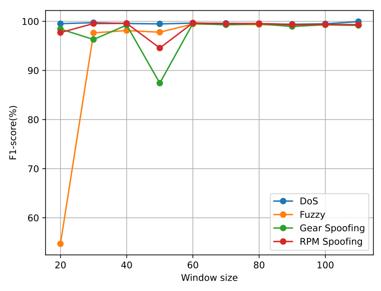
(a)



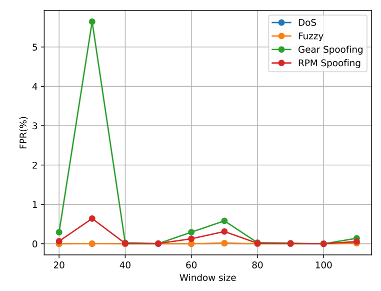
(b)



(c)

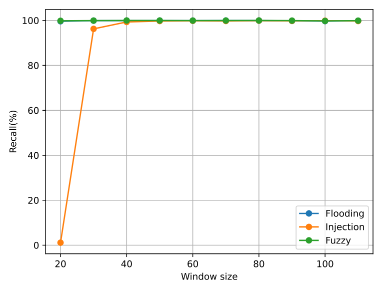


(d)

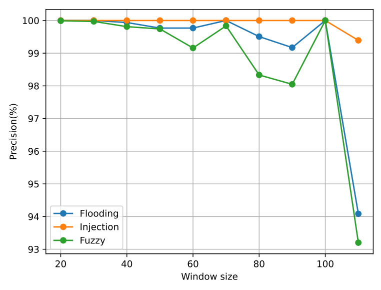


(e)

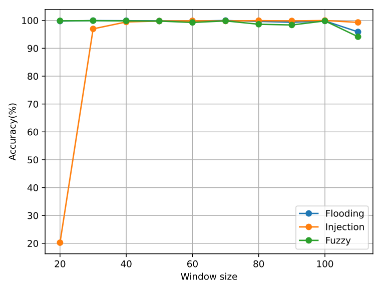
Figure 5. Experimental results by window size under *Car-Hacking* dataset: (a) Recall. (b) Precision. (c) Accuracy. (d) F1-score. (e) FPR.



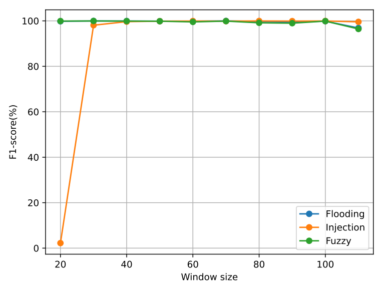
(a)



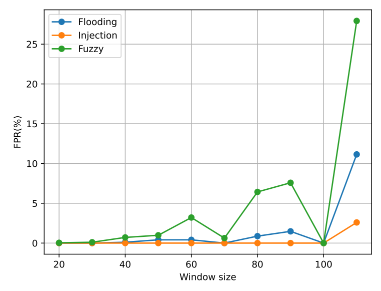
(b)



(c)

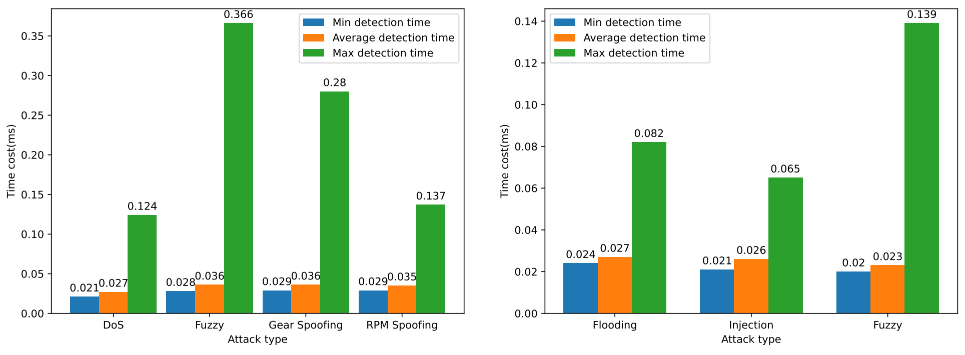


(d)



(e)

Figure 6. Experimental results by window size under *Car-Intrusion* dataset: (a) Recall. (b) Precision. (c) Accuracy. (d) F1-score. (e) FPR.



(a) (b)

Figure 7. Time cost for attack detection under different datasets with window size 100. (a) *Car-Hacking*. (b) *Car-Intrusion*

5. Discussion

The discussion below involves the proposed methodology's deployment, compatibility and limitation.

1. *Deployment:* In theory, the proposed methodology can be deployed in ECUs distributed across the different branches of the CAN bus or centralized in one of these ECUs, which is known as *Gateways*. Although the proposed methodology, under a centralized deployment scheme, has complete visibility of all the traffic that flows in distinct branches of the CAN bus, detection accuracy may decrease due to more noise or aperiodic IDs in CAN traffic. Therefore, the distributed solution is a better choice. At the same time, one additional ECU to each distinct branch of the CAN bus is sufficient for implementing the proposed methodology.

2. *Compatibility:* One of the advantages of the proposed methodology is that it does not require any modification in CAN protocol, so it complies with the existing standards. In addition, since it only needs to monitor CAN traffic, it can also work simultaneously with other approaches to secure the CAN bus, such as (Song, Kim, & Kim, 2016, Wu, Huang, 2018, Ohira, Desta & et al., 2020).

3. *Limitation:* There are some limitations to the proposed methodology. Although the experimental results show that the proposed methodology has better performance in detecting kinds of attacks, the minimum detection unit of this methodology is a message sequence which means that it cannot locate which message is malicious.

6. Conclusion

A novel graph distance-based anomaly detection methodology has been proposed in this paper. We propose to express the CAN traffic in a graph structure (i.e., CAN traffic graph) and perform pattern matching between base graph *bg* and window graph *wg* according to our defined graph distance algorithm. We divide the methodology into two phases, i.e., an offline learning phase and an online detection phase. The proposed methodology first obtains the ground-truth values for detecting attacks by learning from the normal dataset in the offline learning phase and then obtains the detection results by comparing the results computed in real-time with the ground truth values in the online detection phase. In addition, we also propose an update method of *bg* to ensure its validity. Finally, we validate the feasibility of the proposed methodology not only on a public dataset but also in an actual vehicle environment. The experimental results show that the proposed methodology can detect DoS, Fuzzy, Gear Spoofing, RPM Spoofing, Flooding, and Injection attacks with the precision of 100% and FPR of 0%. However, the TPR of this method has not reached 100%, that is, there is a case that the attack graph has not been detected, so this will become our next research work.

References

Akoglu, L., Tong, H., & Koutra, D. (2015). Graph-based anomaly detection and description: a survey. *Data Mining & Knowledge Discovery*, *29*(3). https://doi.org/626-688.10.1007/s10618-014-0365-y

Aliwa, E., Rana, O., Perera, C., & Burnap, P. (2020). Cyberattacks and countermeasures for in-vehicle networks. *ACM New York, NY, USA* (pp. 1-37).

Bosch, R. (1991). CAN specification version 2.0. *Rober Bousch GmbH, Postfach 1991 300240* (pp. 1-72).

Cheng, K., Bai, Y., Zhou, Y., Tang, Y., & Liu, Y. (2020). Caneleon: protecting can bus with frame id chameleon. *IEEE Transactions on Vehicular Technology*, *PP*(99), 1-1. https://doi.org/10.1109/TVT.2020.2990417

Cho, K. T., & Shin, K. G. (2016). Fingerprinting electronic control units for vehicle intrusion detection. *Proc. 25th USENIX Secur. Symp.* (pp. 911-927).

Cui, J., Chen, X., Zhang, J., Zhang, Q., & Zhong, H. (2020). Toward achieving fine-grained access control of data in connected and autonomous vehicles. *IEEE Internet of Things Journal*, *PP*(99), 1-1. https://doi.org/10.1109/JIOT.2020.3041860

HJ, J., HK, J., HY, C., W Choi, & Lee, I. (2019). Mauth-can: masquerade-attack-proof authentication for in-vehicle networks. *IEEE Transactions on Vehicular Technology*, *PP*(99), 1-1. https://doi.org/10.1109/TVT.2019.2961765

Marchetti, M., & Stabili, D. (2017). Anomaly detection of CAN bus messages through analysis of ID sequences. *IEEE Intelligent Vehicles Symposium (IV)*. IEEE.

Mekki, A. E., Bouhoute, A., & Berrada, I. (2019). Improving driver identification for the next-generation of in-vehicle software systems. *IEEE Transactions on Vehicular Technology* (pp.7406-7415). https://doi.org/10.1109/TVT.2019.2924906

Mter, M., & Asaj, N. (2011). Entropy-based anomaly detection for in-vehicle networks. *Intelligent Vehicles Symposium* (pp.1110-1115). IEEE.

Ohira, S., Desta, A. K., Arai, I., Inoue, H., & Fujikawa, K. (2020). Normal and malicious sliding windows similarity analysis method for fast and accurate ids against dos attacks on in-vehicle networks. *IEEE Access*, PP(99), 1-1. https://doi.org/10.1109/ACCESS.2020.2975893

Pawelec, K., Bridges, R. A. , & Combs, F. L. (2019). Towards a CAN IDS Based on a Neural Network Data Field Predictor. *Proc. ACM Workshop Automot. Cybersec. (AutoSec)* (pp. 31-34). https://doi.org/10.1145/3309171.3309180

Pu, Z., Zhu, M., Li, W., Cui, Z., & Wang, Y. (2020). Monitoring public transit ridership flow by passively sensing wi-fi and bluetooth mobile devices. *IEEE Internet of Things Journal*, *PP*(99), 1-1. https://doi.org/10.1109/JIOT.2020.3007373

Seo, E., Song, H. M., & Kim, H. K. (2019). Gids: gan based intrusion detection system for in-vehicle network. *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)* (pp.1-6). https://doi.org/10.1109/PST.2018.8514157

Song, H. M., & Kim, H. K. (2021). Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data. *IEEE Transactions on Vehicular Technology*, *PP*(99), 1-1. https://doi.org/10.1109/TVT.2021.3051026

Song, H. M., Kim, H. R., & Kim, H. K. (2016). Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. *International Conference on Information Networking*. IEEE.

Song, H. M., Woo, J., & Kim, H. K. (2020). In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications* (pp.100198.1-100198.13). https://doi.org/10.1016/j.vehcom.2019.100198

Stabili, D., Marchetti, M., & Colajanni, M. (2017). Detecting attacks to internal vehicle networks through Hamming distance. *2017 AEIT International Annual Conference* (pp.1-6).

Taylor, A., Japkowicz, N., & Leblanc, S. (2015). Frequency-based anomaly detection for the automotive CAN bus. *2015 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE.

Woo, S., Jo, H. J., & Dong, H. L. (2014). A practical wireless attack on the connected car and security protocol for in-vehicle can. *IEEE Transactions on Intelligent Transportation Systems* (pp.993-1006). https://doi.org/10.1109/TITS.2014.2351612

Wu, W. F., & Huang, Y. Z. et al. (2018). Sliding window optimized information entropy analysis method for intrusion detection on in-vehicle networks. *IEEE Access* (pp.45233-45245). https://doi.org/10.1109/ACCESS.2018.2865169

Copyrights