Vehicular Communications 29 (2021) 100338

Contents lists available at ScienceDirect

Vehicular Communications

www.elsevier.com/locate/vehcom

Convolutional neural network-based intrusion detection system for AVTP streams in automotive Ethernet-based networks

Seonghoon Jeong[a], Boosun Jeon[b], Boheung Chung[b], Huy Kang Kim[a,*]

[a] School of Cybersecurity, Korea University, Seoul 02841, Republic of Korea
[b] Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon 34129, Republic of Korea

A R T I C L E   I N F O

Article history:
Received 16 June 2020
Received in revised form 7 October 2020
Accepted 10 December 2020
Available online 2 February 2021

Keywords:
Automotive Ethernet
In-vehicle network
Network security
Replay attack
Intrusion detection system
Convolutional neural network

A B S T R A C T

Connected and autonomous vehicles (CAVs) are an innovative form of traditional vehicles. Automotive Ethernet replaces the controller area network and FlexRay to support the large throughput required by high-definition applications. As CAVs have numerous functions, they exhibit a large attack surface and an increased vulnerability to attacks. However, no previous studies have focused on intrusion detection in automotive Ethernet-based networks. In this paper, we present an intrusion detection method for detecting audio-video transport protocol (AVTP) stream injection attacks in automotive Ethernet-based networks. To the best of our knowledge, this is the first such method developed for automotive Ethernet. The proposed intrusion detection model is based on feature generation and a convolutional neural network (CNN). To evaluate our intrusion detection system, we built a physical BroadR-Reach-based testbed and captured real AVTP packets. The experimental results show that the model exhibits outstanding performance: the F1-score and recall are greater than 0.9704 and 0.9949, respectively. In terms of the inference time per input and the generation intervals of AVTP traffic, our CNN model can readily be employed for real-time detection.

1. Introduction

Connected and autonomous vehicles (CAVs) are an innovative form of *traditional vehicle*. A significant difference between traditional vehicles and CAVs is the following: In CAVs, a physical (PHY) layer is used for the in-vehicle network (IVN), and automotive Ethernet replaces the controller area network (CAN) and FlexRay. As a result, CAVs support the large communication throughput required by high-definition applications such as video-on-demand, intelligent transport systems, and advanced driver assistance systems (ADAS), which consume various sensor data and video streams [1]. The IEEE 1722 audio-video transport protocol (AVTP) is the essential mechanism by which automotive Ethernet ensures the reliable transport of time-sensitive and prioritized traffic (e.g., audio and video streaming, as the name suggests). Moreover, the IEEE 1722-2016 standard defines the AVTP for the transmission of time-sensitive control streams (including CAN and FlexRay messages) via automotive Ethernet; the latter was defined by the IEEE P1722a working group [2]. Accordingly, we consider that AVTP will be one of the most crucial protocols for IVNs in CAVs.

Currently, cyber-attacks against vehicles are rare (even with a lack of protection mechanisms) because the vehicles have inflexible applications and limited processing resources that are only sufficient for their mobility [3]. However, the attack surface and feasibility of attacks are substantial for CAVs because of their additional functions. For example, Automotive Grade Linux or Android OS on an infotainment device could inherit vulnerabilities or malware inbound owing to the connectivity of CAVs [4]. Intrusion detection is a useful method to prepare for cyber-physical attacks on IVNs. Although there have been several studies on intrusion detection systems (IDSs) for IVN security [3,5], intrusion detection for systems employing automotive Ethernet has not been studied. The interruption of a media stream owing to intrusion raises not only a usability issue but also security and safety issues in CAVs. Therefore, it is necessary to develop an IDS to address the gap in intrusion detection research with regard to automotive Ethernet.

Convolutional neural networks (CNNs) are artificial neural networks that are often used in network traffic classification problems (e.g., classifying application protocols or intrusion detection). CNNs are extensively used because they can easily learn and predict one- or multidimensional data using a few parameters. In a CNN, a few sets of convolutional layers and pooling layers extract features from a given input. Finally, dense layers act as an estimator that returns a predicted class or a value using the extracted fea-

* Corresponding author.
E-mail addresses: seonghoon@korea.ac.kr (S. Jeong), bsjeon@etri.re.kr (B. Jeon), bhjung@etri.re.kr (B. Chung), cenda@korea.ac.kr (H.K. Kim).

https://doi.org/10.1016/j.vehcom.2021.100338
2214-2096/© 2021 Elsevier Inc. All rights reserved.

tures. Thus, CNNs simplify supervised learning without the need for much preprocessing of the input data.

In this paper, we propose an intrusion detection method using a deep learning model. Our method includes the feature generation process and a two-dimensional CNN (2DCNN) model. We designed the feature generator considering the observed characteristics of real AVTP traffic. The detection model distinguishes whether AVTP packets transmitted over automotive Ethernet are benign or injected on a packet-by-packet basis. Finally, we implemented an IDS using the proposed method to evaluate the performance of our method.

The main contributions of this study are as follows:

- We introduce six possible attack scenarios for automotive Ethernet. We also demonstrate an AVTP replay attack using real audio-video bridging (AVB) devices.
- We propose a novel intrusion detection method designed for automotive Ethernet. Our approach comprises the feature generator and the 2D-CNN model. To the best of our knowledge, this is the first attempt to solve the problem of the security of automotive Ethernet-based IVNs.
- We implement an IDS to evaluate the performance of our intrusion detection method by using a real dataset that we captured from a BroadR-Reach network. The evaluation results show that the IDS correctly classifies almost all AVTP packets, with very few false negatives. Furthermore, we confirm that our IDS is suitable for real-time detection.
- We provide automotive Ethernet intrusion datasets used for our experiment in the PCAP format. In the datasets, each AVTP packet is labeled as “benign” or “injected.” Readers can access our datasets in [6] to reproduce our experiment results, as well as use the datasets in further studies.

The paper is organized as follows. Section 2 introduces the background of automotive Ethernet and AVTP. Section 3 provides plausible attack scenarios targeted to automotive Ethernet. Section 4 describes the results of our replay attack performed on our real automotive Ethernet network as well as the proposed IDS design—including the feature generator and the artificial neural network used to classify AVTP packets. Section 5 shows the experiment results on the real AVTP traffic. Section 6 discusses the AVTP intrusion dataset, some limitations, and remediation strategies. Section 7 reviews previous studies on IDS designed for CAVs. Finally, Section 8 concludes the paper.

2. Preliminaries

2.1. Background of emerging automotive Ethernet

The CAN protocol is widely adopted for IVNs of traditional vehicles: its arbitration mechanism enables a simple bus topology, and its real-time message prioritization considers the importance of each application. However, there are two drawbacks to CAN-based IVNs: (1) security issues and (2) limited bandwidth which is not suitable for various multimedia streams (such as camera data and high-quality audio/video). *Traditional Ethernet* was once considered a successor of the CAN bus for IVNs. Ethernet is commonly used in local-area networks and provides efficient, low-cost, and high-bandwidth communication. However, the best-effort transmission that Ethernet provides is insufficient for CAVs: an unexpected delay or further traffic loss can be highly detrimental in vehicular applications. Thus, Ethernet protocols require some modifications in the PHY and the higher layers to enable their application to IVNs. Consequently, the concept of automotive Ethernet has emerged, and automotive Ethernet has been developed by standardization groups and industry experts.

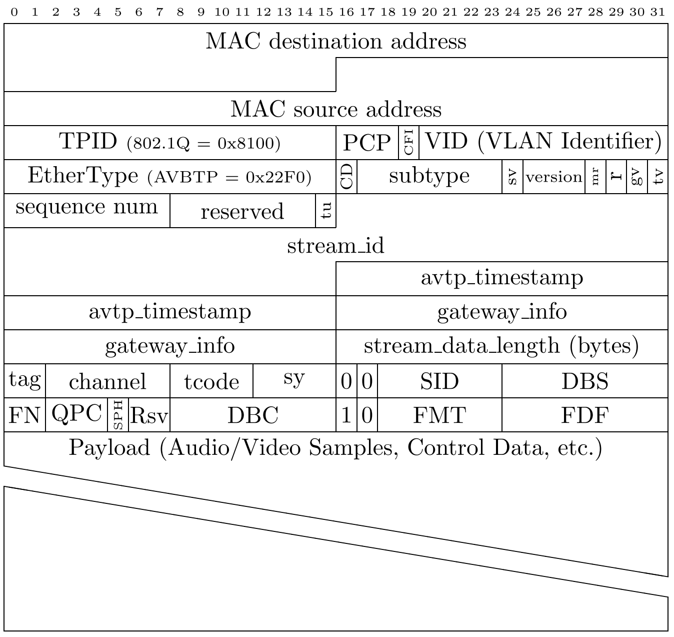


Fig. 1. Structure of stream AVTPDU — Values of `TPID`, `EtherType`, and `CD` fields are set to 8100h, 22F0h, and 0b, respectively.

2.2. Automotive Ethernet and IEEE 1722

Automotive Ethernet has innovations in the PHY and higher layers. Open Alliance’s special interest group designed and published BroadR-Reach technology by modifying the PHY employed in Ethernet to meet electrical requirements for automotive Ethernet. Subsequently, IEEE standardized 100BASE-T1, which succeeded BroadR-Reach with a few minor modifications. As a result, automotive Ethernet-based IVNs can be deployed with a twisted-pair cable; furthermore, automotive Ethernet exhibits a high bandwidth of up to 100 Mbps. However, changes in the PHY do not guarantee low-latency, time-sensitive, and prioritized communication based on existing protocols.

To guarantee the aforementioned features, Avnu Alliance implemented a set of technical standards named AVB. Notably, AVB enables the reliable transport of time-critical streams on automotive Ethernet. Specifically, four protocols—stream reservation protocol, generalized precision time protocol (gPTP), forwarding and queuing for time-sensitive streams, and AVTP—comprise AVB technology. The protocol headers are encapsulated in standard Ethernet headers when the packets are transmitted through automotive Ethernet. For example, an AVTP packet is encapsulated by the general MAC header as well as the VLAN header. The VLAN header functions as the prioritization enabler for automotive Ethernet; in addition, it enables network segmentation, as intended. With the support of these protocols, AVB-compatible devices become synchronized with each other and produce or consume streams regardless of topological complexity [7].

Fig. 1 shows the structure of a stream AVTP data unit (AVTPDU). AVTP is in charge of delivering the application payload. The payload can include audio and video streams, in addition to CAN or FlexRay messages for controlling in-vehicle electronic control units (ECUs). AVTPDU refers to an AVTP packet; AVTPDUs are categorized into *control AVTPDU* (value of the `CD` field is set as 1) and *stream AVTPDU* (value of the `CD` field is set as 0). AVB devices send control AVTPDUs to discover themselves or establish/disconnect a stream session. Meanwhile, in-vehicle switches learn the positions of AVTP listeners. After the session is established, the AVB talker transmits stream AVTPDUs containing application data. Then, the in-vehicle switches multicast the stream AVTPDUs. Stream AVTPDUs always

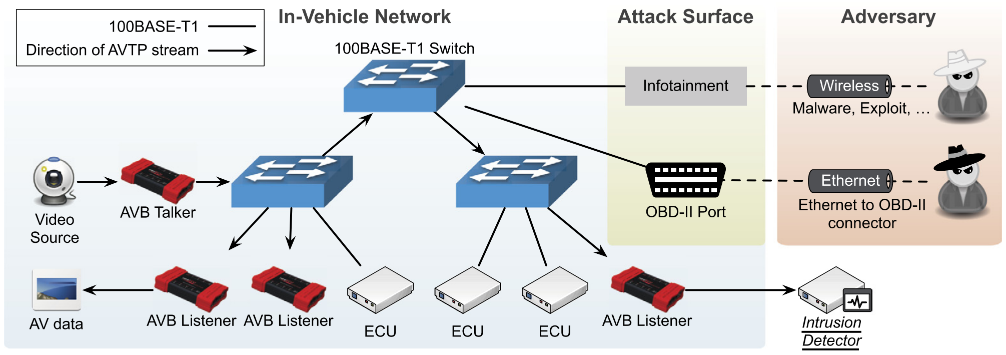


Fig. 2. Example of in-vehicle automotive Ethernet topology along with possible packet injection attack scenarios. AVTP streams flow from the AVB talker to the AVB listeners.

adhere to a VLAN tag to represent their priority among less time-critical packets. The `channel` field specifies the type of protocol for the subsequent `Payload` field.

3. Threats in automotive Ethernet

Several studies have evaluated threats to IVNs, especially with regard to the CAN bus. In contrast, there is a lack of research to identify threats to automotive Ethernet-based networks. In this section, we introduce threats targeting automotive Ethernet-based IVNs. This section does not elucidate all possible threats to IVNs but emphasizes the need to identify attacks on automotive Ethernet. We assume that the attacker has access to a target IVN. Fig. 2 shows an example topology of automotive Ethernet-based IVN in addition to possible scenarios of how adversaries can invade the target IVN. Attackers can invade the IVN through physical access using the OBD-II port and through remote access via compromised infotainment devices. Once an attacker gains such access, (s)he is capable of (1) passive attacks and (2) active attacks.

An attacker carries out a **passive attack** by monitoring inbound traffic on a particular node monitored by him/her. The goals of a passive attack include identifying available services, vehicle status, and nature of in-vehicle communications. For example, a passive attacker can identify the presence of a newly established AVTP stream from inbound control AVTPDUs. A passive attack is difficult to detect, as it does not affect any data within IVNs. Fortunately, the impact of a passive attack is relatively limited (compared with an attack on the CAN bus to which all nodes are connected). The reason is that in-vehicle switches transmit only broadcast packets to the monitoring node, whereas other packets reach only their designated destinations.

An **active attack** begins with a packet injection performed by the attacker. The attacker can inject arbitrary packets. Hence, injection attacks can be classified into several types depending on the attacker's purpose and the effects of the attack on IVNs. Here, we introduce five types of plausible active attacks (but further discovery and assessment are also needed):

- **Denial of service (DoS) attack.** An attacker may attempt a DoS attack in a specific ECU by flooding it with numerous VLAN-tagged packets. Automotive Ethernet is vulnerable to such an attack because it prioritizes network traffic and ensures a certain quality of service using a 3-bit *PCP* field in the VLAN header. Thus, the packets injected by an attacker can override other legitimate packets transmitted to the target ECU. Specifically, the flooder initiates a burst transmission to the switch of the target ECU. This causes packet loss on the network level; other ECUs cannot transmit any packets to the target ECU during the DoS attack. However, the DoS attack does not affect other ECUs. Note that each stream AVTPDU contains a VLAN header with a value of 3 in the *PCP* field (depicted in row 3 in Fig. 1). Consequently, an AVB listener may not receive an AVTP stream owing to the DoS attack.
- **Content-addressable memory (CAM) table overflow.** A CAM attack exploits the MAC address learning process of in-vehicle switches. Specifically, the attacker floods the network with packets having a random source MAC address until the target switch cannot accept new MAC address-port pair entries. This occurs when the CAM table of the target switch is full. As a result, all inbound traffic is flooded to all ECUs connected to the target switch. However, this threat exists not only in automotive Ethernet switches but also in general switches [8]. Commercial switches can be equipped with countermeasures for this attack [9]. Nevertheless, we should ensure that in-vehicle switches are robust against CAM table overflow attacks, considering that this is a period of transition from the use of the traditional CAN bus to automotive Ethernet. If an in-vehicle switch is vulnerable to the CAM table overflow attack, the impact of passive attacks will be very high because a single monitoring node can dump all transmitted packets.
- **Fuzzing.** In a fuzzing attack, random inputs are given to a target ECU in the form of a network packet. In this case, the attacker can scan active services, disrupt communications, and execute unexpected commands for a designated service. Furthermore, a fuzz packet can occasionally trigger an undocumented service mode (e.g., ECU firmware upgrade mode) or change parameters that are crucial for vehicle operation. The black-box fuzzing attack is easy to perform because it requires less effort to affect the target vehicle. Furthermore, a gray-box fuzzing attack conducted by an attacker with prior knowledge may cause a critical accident. Most of the traffic generated by a fuzzer contains a malformed payload, which makes such traffic easy to detect in an IVN implementation.
- **Command injection.** This attack is performed by an attacker who knows the specification of a target vehicle in detail. The attacker injects well-crafted payloads to the IVN of the target vehicle. Then, the target vehicle executes commands as assigned. The nature of command injection depends on the protocol or application targeted by an attacker. For example, packet injection may take place over a varying time interval, depending on the type of attack. Rezvani [10] discussed the working of an automotive Ethernet camera and demonstrated a command injection attack. He further exploited the *end of image function* of the target camera to override a video stream in accordance with his commands. To detect a command injection attack, an IDS needs to recognize the context of the communication or know the signatures captured from the injected packets. The risk of command injection attacks is very high because the target vehicle executes commands as intended by the attacker.

Table 1
Five active attacks on automotive Ethernet-based IVNs.

Attack	Target	Goal	Likelihood	Risk	Discriminability
DoS attack	ECU	Out-of-communication status for a target ECU	High	High	Easy
CAM table overflow	Switch	Disable switching and broadcast all legitimate traffic within IVNs	High	Low	Easy
Fuzzing	ECU	Execute an unexpected/hidden command on the target ECU	High	High	Easy
Command injection	ECU	Execute a command as intended by the attacker	Low	High	Moderate
Replay attack	ECU	Re-execute instructions executed at the time of traffic extraction	Middle	High	**Hard**

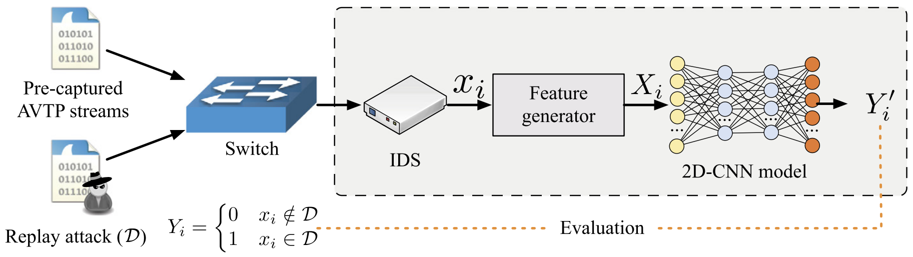

Pre-captured
AVTP streams
Switch
Replay attack ($\mathcal{D}$)
IDS
x_i
Feature generator
X_i
2D-CNN model
Y_i'
$Y_i = \begin{cases} 0 & x_i \notin \mathcal{D} \\ 1 & x_i \in \mathcal{D} \end{cases}$
Evaluation

Fig. 3. Summary of our system design.

- **Replay attack.** A replay attack enables an attacker to input a few commands even without prior knowledge of the attack target. To execute a replay attack, the attacker must possess a pre-captured packet dump. By replaying the packet dump to the IVN, the attacker can make the target vehicle execute certain instructions (which were captured during the traffic extraction). With regard to an IDS, detecting a replay attack is straightforward because there will be two ongoing sessions during such an attack, which is uncommon. However, it is challenging to determine which session or packets are being replayed by the attacker (because the injected packets were also generated by legitimate ECUs).

Table 1 summarizes five types of active attacks. We evaluate the *likelihood*, *risk*, and *discriminability* of the attacks based on the attack characteristics discussed previously. Here, we assign a high *likelihood* when no prior knowledge or effort is required before the attack. We assign a high *risk* if the target ECU can become out-of-order or can execute an attacker's command. Finally, *discriminability* indicates whether the injected packet is distinct from normal traffic (the packets can be easily detected if the injected packet is distinct). DoS attacks are relatively easy to detect because such attacks involve fixed payloads. CAM table overflow and fuzzing attacks are also easy to detect because they involve the injection of randomly generated packets. The detection of a command injection attack requires appropriate knowledge; therefore, it involves moderate difficulty. In contrast, replay attacks are very difficult to detect because the replayed packets have already been used in the target vehicle. Therefore, it is necessary to develop an IDS particularly for replay attacks.

4. System design

In this section, we propose an IDS designed to detect continuous replay attacks abusing stream AVTPDUs. In Section 4.2, we demonstrate an example of the replay attack on automotive Ethernet. Fig. 3 depicts the proposed system design.
IDS deployment. We assume that the IDS should monitor consecutive AVTP stream to detect intrusions. To this end, we describe how an IDS can be deployed in a CAV. Fig. 2 shows that the IDS is connected to one of AVB listeners. Because AVTP supports one-to-many streams, IDS can observe AVTP streams through an AVB listener without affecting other in-vehicle systems.

4.1. AVB packets

Before we develop an intrusion detection method for replay attacks on automotive Ethernet, we need to gather automotive Ethernet packets. To this end, we first implemented a testbed using a BroadR-Reach physical network to capture AVB-related packets. We connected an AVB talker and an AVB listener to a BroadR-Reach switch. Then, we configured our BroadR-Reach switch to forward all of the inbound packets to a monitoring port. Hence, we accrued all of the communications between the AVB devices.

Initially, the two devices synchronize through gPTP packets and send control AVTPDUs to establish a new session. After the connection is established, the AVB talker transmits one-way stream AVTPDUs. The stream AVTPDUs contain a compressed MPEG-2 TS live video stream from a USB camera directly connected to the AVB talker. The AVB listener plays the video by combining the received packets. All such communication processes are stored as one packet dump. We collected several packet dumps for the experiment (the pre-captured AVTP streams are shown in Fig. 3).

4.2. Adversary

We suppose that an attacker injects arbitrary stream AVTPDUs into the IVN. The goal of the attacker is to output a single *video frame*, at a terminal application connected to the AVB listener, by injecting previously generated AVTPDUs during a certain period. To demonstrate the attack, we extract 36 continuous stream AVTPDUs from one of our AVB datasets; the extracted AVTPDUs constitute one *video frame*. Then, the attacker performs a ***replay attack*** by sending the 36 stream AVTPDUs repeatedly.

To illustrate the effect of the replay attack, Fig. 4 shows the data collected from our testbed during this attack. As shown in Fig. 4(c), the adversary successfully compromises the video stream on the AVB listener side. The terminal application plays corrupted video footage when the replay attack occurs (because the AVB listener receives stream AVTPDUs from two talkers). The attack can have a critical impact if the terminal application is the ADAS; a CAV has no choice but to be blinded or make a wrong decision. Therefore, it is important to detect injection attacks and replay attacks.

4.3. Payload observations

In this section, we present some observations of stream AVTPDU payloads to explain their characteristics. We also elucidate the motivation for our design of the packet generator.

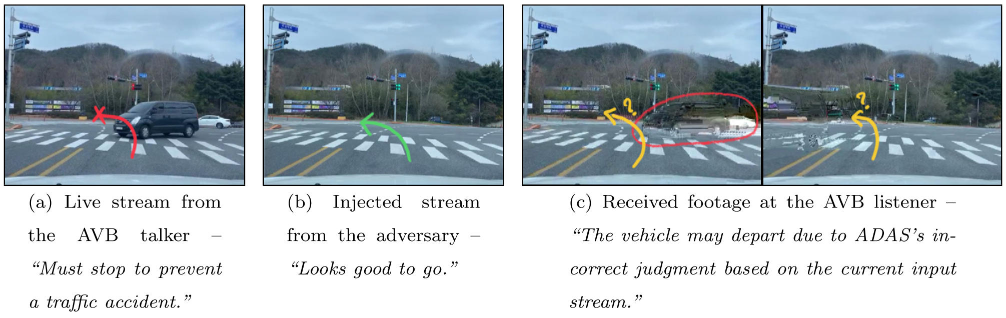

(a) Live stream from the AVB talker – "Must stop to prevent a traffic accident."
(b) Injected stream from the adversary – "Looks good to go."
(c) Received footage at the AVB listener – "The vehicle may depart due to ADAS's incorrect judgment based on the current input stream."

Fig. 4. Demonstration of a replay attack (captured from our BroadR-Reach network). The terminal application plays a distorted video, as well as a little bit of the residual video; however, it mostly plays the scene intended by the attacker, as depicted in (c). Assuming that an autonomous driving system is connected to the AVB listener, a traffic accident is impending in this example scenario.

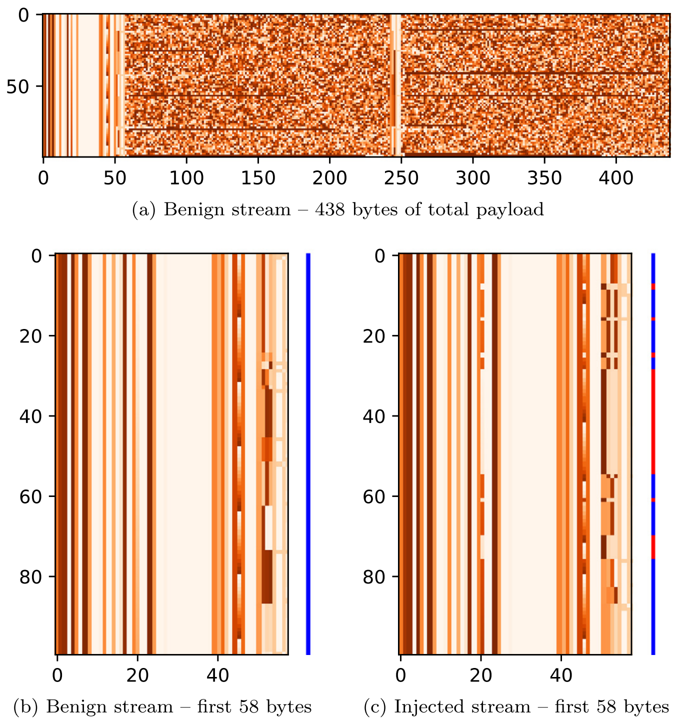


Fig. 5. Visualization of 100 continuous stream AVTPDU payloads. The vertical bars in (b) and in (c) represent whether the stream is benign (blue) or injected (red). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Fig. 5 shows the visualization of 100 continuous-stream AVTPDU payloads captured from our experiment testbed. Each pixel represents a byte value: light and dark cells indicate low and high values, respectively. Values are given in the order of incoming AVTPDUs (from the top row to the bottom row). Each stream AVTPDU that we captured has the same size (438 bytes).

We found some interesting aspects in the first 58 bytes of the stream AVTPDUs Fig. 5(b). First, each header of the stream AVTPDUs, shown in Fig. 1, has a fixed value, except for the `sequence num`, `DBC`, and `Payload` fields. Second, the values of the `sequence num` and `DBC` fields sequentially increase by 01h and 10h, respectively, and are set to 00h when it overflows. Third, values in the offset range of 51–58 (the first eight bytes of the `Payload` field) change occasionally. This pattern repeats every time a new video frame is received.

In contrast, we found no meaningful pattern in the byte offset range of 59-438, although a light vertical line (offset 248) and a dark horizontal line were observed. The offset range of 59-438 is part of the MPEG-2 TS payloads carried by the AVTPDUs.

Fig. 5(c) visualizes the payloads during a replay attack. The injected frames can be distinguished using the vertical line on the right. Conspicuous grid-like patterns can be observed on the aforementioned fields; the patterns correspond to the alternation of normal and injected stream AVTPDUs. Such a pattern means that the fields representing the sequence or the context in the payload are fragmented. However, the replay attack may seem like a normal flow when the adversary dominates the network traffic (see near the 40th packet). This makes it difficult to detect replayed packets.

There is no change in the Ethernet header or the VLAN header. Malformed stream AVTPDUs injected by a weak adversary should look conspicuous and awkward in the benign stream in automotive Ethernet-based networks. For example, we can easily detect a DoS attack based on a modified VLAN header. An IDS can quickly detect such injections with substantially less effort; an example of this is rule-based intrusion detection that observes the modification of static payloads.

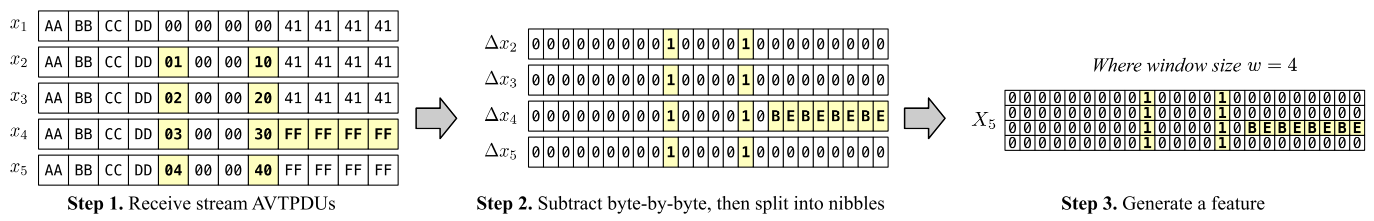
4.4. Feature generator

Next, we design the feature generator so that the proposed CNN model can distinguish and classify the changes in the payload after a packet injection. The feature generator receives a stream AVTPDU from a BroadR-Reach network and generates a feature to be used as an input of the CNN. Based on our observations, we decided to focus on the first 58 bytes of the i-th arriving stream AVTPDU. Moreover, the feature generator uses a window size $w \in \mathbb{N} \mid w \geq 4$; it aggregates the recently arrived traffic using w and then produces a two-dimensional vector from it.

Let x_i be a one-dimensional vector containing j-bytes of i-th stream AVTPDU, which is given as $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,(j-1)}, x_{i,58})$ where $x_{i,j} \in \mathbb{N} \mid 0 \leq x_{i,j} < 256$. Thus, vector x_i includes all fields and only the first 8 bytes of `Payload` field. We set $j = 58$ because we consider the first 58 bytes of each stream AVTPDU payload. Considering two stream AVTPDUs x_{i-1} and x_i, let Δx_i be the state change of the i-th stream AVTPDU, which is derived as follows:

$$\begin{aligned}\Delta x_i \equiv& (x_i - x_{i-1}) \mod 2^8 \\ \equiv& (x_{i,1} - x_{(i-1),1}, \ldots, x_{i,58} - x_{(i-1),58}) \mod 2^8 \\ \equiv& (\Delta x_{i,1}, \Delta x_{i,2}, \Delta x_{i,3}, \ldots, \Delta x_{i,(j-1)}, \Delta x_{i,58}) \\ =& (u_{i,1}, v_{i,1}, u_{i,2}, v_{i,2} \ldots, u_{i,58}, v_{i,58})\end{aligned} \tag{1}$$

where $\Delta x_{i,j} = u_{i,j}2^4 + v_{i,j}$ and $u_{i,j}, v_{i,j} \in \mathbb{N} \mid 0 \leq u_{i,j}, v_{i,j} < 16$. Finally, a $w \times 2j$ sized two-dimensional vector X_i is defined as



x_1 AA BB CC DD 00 00 00 00 41 41 41 41
x_2 AA BB CC DD 01 00 00 10 41 41 41 41
x_3 AA BB CC DD 02 00 00 20 41 41 41 41
x_4 AA BB CC DD 03 00 00 30 FF FF FF FF
x_5 AA BB CC DD 04 00 00 40 FF FF FF FF
Step 1. Receive stream AVTPDUs
Δx_2 000000000100001000000000
Δx_3 000000000100001000000000
Δx_4 00000000010000010BEBEBEBE
Δx_5 000000000100001000000000
Step 2. Subtract byte-by-byte, then split into nibbles
Where window size $w = 4$
X_5
Step 3. Generate a feature

Fig. 6. Proposed feature generation process.

Table 2
Structure of the convolutional neural network and the output dimensions (where $w = 44$).

Role	Layer name	Data shape	Activation	Hyperparameters
Input	Input (X_i)	$44 \times 116 \times 1$	–	–
Feature extraction	Conv2D_1	$44 \times 116 \times 32$	ReLU	filter_size=$5 \times 5 \times 32$, stride=1×1, “same” padding, L2 reg.
	BatchNormalization_1	$44 \times 116 \times 32$	–	momentum=0.99, epsilon=0.001
	MaxPool_1	$22 \times 58 \times 32$	–	pool_size=2×2
	Conv2D_2	$22 \times 58 \times 64$	ReLU	filter_size=$5 \times 5 \times 64$, stride=1×1, “same” padding, L2 reg.
	BatchNormalization_2	$22 \times 58 \times 64$	–	momentum=0.99, epsilon=0.001
	MaxPool_2	$11 \times 29 \times 64$	–	pool_size=2×2
Binary classification	Flatten	20416	–	–
	Dropout_1	20416	–	dropout_ratio=0.3
	Dense_1	64	ReLU	–
	Dropout_2	64	–	dropout_ratio=0.3
	Dense_2/Output (Y_i')	1	Sigmoid	–

$$X_i = \begin{pmatrix} \Delta x_{i-w+1} \\ \Delta x_{i-w+2} \\ \vdots \\ \Delta x_i \end{pmatrix} = \begin{pmatrix} u_{(i-w+1),1} & v_{(i-w+1),1} & \cdots & v_{(i-w+1),58} \\ u_{(i-w+2),1} & v_{(i-w+2),1} & \cdots & v_{(i-w+2),58} \\ \vdots & \vdots & \ddots & \vdots \\ u_{i,1} & v_{i,1} & \cdots & v_{i,58} \end{pmatrix} \tag{2}$$

This expression gives the window size corresponding to a recent stream change.

Fig. 6 illustrates the process of feature generation. In summary, the feature generator returns X_i on the arrival of the i-th stream AVTPDU once the window is full (i.e., $i > w$). The feature generator feeds each X_i to the proposed CNN model. $Y_i \in \{0, 1\}$ is employed to identify whether the i-th stream AVTPDU is benign or injected; this data is then used for training our CNN model.

4.5. CNN-based intrusion detection model

Our CNN model comprises an input layer, two hidden sub-layers, and two dense layers. Table 2 details the proposed CNN architecture and hyperparameters. Window size w must be a multiple of 4 owing to size reduction in two max-pooling layers. To express the dimension of each layer, we use window size $w = 44$, which is the optimal value for our dataset (see Section 5.4). Thus, the input layer accepts $44 \times 116 \times 1$-sized data.

Each hidden sub-layer contains a convolution layer, a batch normalization layer, and a max-pooling layer. The *batch normalization* layers prevent overfitting and accelerate the training phase [11]. In the two *Conv2D* layers, we use L2 regularization as the kernel regularizer, which limits the values of the weights. We deploy two *dropout* layers between fully connected layers, which also prevents overfitting [12]. The rectified linear unit (ReLU) activation function gives non-linearity to the neural network and is widely used in CNN architectures to learn complex features in input data.

The last dense layer returns Y_i' through a *sigmoid* activation function, where $Y_i' \in \mathbb{R} \mid 0 \le Y_i' \le 1$. We consider output Y_i' as the probability of X_i being injected. In the training phase, the binary cross-entropy loss function compares Y_i' to Y_i. Our CNN model is trained using the Adam optimizer [13], with a learning rate of 0.001.

5. Experiment

5.1. Environment

In the experiments, we use the Python library Keras [14] to implement our 2D-CNN model. We train and evaluate our model using Google Colaboratory on an NVIDIA Tesla P100 GPU.

We performed four packet captures of attack-free stream AVTPDUs from our BroadR-Reach network in the PCAP format for 70 minutes in total. The first two packet captures were performed indoors (in our laboratory) for 25 min, and the last two packet captures were performed for 45 min in a vehicle driving on the road. We noticed that the packet transmission rates were different in both environments. A possible reason is that the camera connected to the AVB talker was capturing a more dynamic scene during driving. For instance, the average interval between packets is 3,157 µs indoors and 1,735 µs for a moving vehicle.

Each packet capture corresponds to a single AVB session, including broadcast messages for node discovery, establishment of an AVB session, video transmission over stream AVTPDUs, and disconnection of the AVB session.

5.2. AVTP intrusion dataset

To demonstrate a replay attack, we continuously inject a *video frame* (the 36 stream AVTPDUs discussed in Section 4.2) and recapture the four packets. Note that we waited for 120 seconds before the attack to obtain some data corresponding to the attack-free scenario, which allowed our 2D-CNN model to learn characteristics corresponding to the benign status as well. Then, we turned the four captured packets into AVTP intrusion datasets using the feature generator. All packets except for stream AVTPDUs were excluded during the feature generation.

We denote the AVTP intrusion datasets as $\mathcal{D}_{\text{indoors}}$ and $\mathcal{D}_{\text{driving}}$. We selected $\mathcal{D}_{\text{indoors}}$ as the training/validation set. $\mathcal{D}_{\text{indoors}}$ was used to choose proper values for w and for the hyperparameters of our 2D-CNN model. We used $\mathcal{D}_{\text{driving}}$ as the test set. Note that the packet collection environments of both datasets are independent. Thus, there can be no potential over-fitting caused by unrecognized payloads. $\mathcal{D}_{\text{indoors}}$ has 446,372 benign X_is and 196,894 injected X_is. $\mathcal{D}_{\text{driving}}$ has 1,494,257 benign X_is and 376,236 injected X_is.

5.3. Evaluation metrics

We use Accuracy, Precision, and Recall as performance metrics. The *True Positive (TP)* and *True Negative (TN)* are a number of AVTPDUs correctly classified as benign and injected, respectively. On the other hand, the *False Positive (FP)* and *False Negative (FN)* are a number of AVTPDUs incorrectly classified as benign and injected, respectively. Then, the performance metrics are defined as:

We use accuracy, precision, and recall as performance metrics. *True positive (TP)* and *true negatives (TN)* are the numbers of AVTPDUs correctly classified as benign and injected, respectively. *False positives (FP)* and *false negatives (FN)* are the numbers of AVTPDUs incorrectly classified as benign and injected, respectively. Thus, the performance metrics are defined as follows:

$$\begin{aligned}\text{Accuracy} &= (\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN})\\ \text{Precision} &= \text{TP}/(\text{TP}+\text{FP})\\ \text{Recall} &= \text{TP}/(\text{TP}+\text{FN})\end{aligned} \tag{3}$$

Moreover, the F1-score is the harmonic mean of precision and recall, which is sometimes more useful than accuracy, especially when the class distribution is imbalanced. The F1-score is calculated as follows:

$$\text{F1-score} = 2 \times \text{Precision} \times \text{Recall}/(\text{Precision}+\text{Recall}) \tag{4}$$

We also used the receiver operating characteristic (ROC) curve and the area under the ROC curve (AUC) to demonstrate the accuracy of our model with regard to the classification of X_is. The ROC curve is a common method to measure the performance of binary classifiers. The ROC curve shows changes in the true-positive rate (TPR) and false-positive rate (FPR) according to the threshold change for Y_i'. The TPR and FPR are calculated as follows:

$$\begin{aligned}\text{TPR} &= \text{Recall} = \text{TP}/(\text{TP}+\text{FN})\\ \text{FPR} &= \text{FP}/(\text{TN}+\text{FP})\end{aligned} \tag{5}$$

5.4. Choosing an optimal window size

To find an optimal w, we created 20 $\mathcal{D}_{\text{indoors}}$ sets for $w = (4, 8, 12, \ldots, 76, 80)$ using the feature generator. Then, we built and trained 20 2D-CNN models with different input shapes. We randomly assigned 80% of X_is to the training set and 20% of X_is to the test set for each $\mathcal{D}_{\text{indoors}}$.

Table 3 presents the performance of the model with respect to w. The best result in each column is highlighted. When a smaller window size is employed, the training requires less time. However, the classification performance is poor for small window sizes. Conversely, one might conclude that a large w will be advantageous because a wide window allows the 2D-CNN model to capture more information from the input data. However, the largest w in our experiment does not bring the best result. Instead, the best performance was obtained for $w = 44$ for the test set and $w = 60$ for the training set. Therefore, we chose 44 as the optimal window size because the model is more robust to the test set.

Table 3
Model performance on our dataset depending on window size w. The best outcome in each column is highlighted. The result shows that the optimal value of w is 44 or 60 (instead of the highest value).

w	Training time (μs/sample)	Train loss	Train accuracy	Test accuracy	Test F1-score
4	**81**	0.4454	0.7578	0.7603	0.7603
8	96	0.2996	0.8249	0.8303	0.8303
12	104	0.1981	0.9101	0.9126	0.9126
16	115	0.1225	0.9595	0.9633	0.9633
20	134	0.0915	0.9771	0.9781	0.9781
24	146	0.0666	0.9856	0.9867	0.9867
28	153	0.0606	0.9878	0.9928	0.9928
32	166	0.0575	0.9889	0.9903	0.9903
36	180	0.0523	0.9894	0.9897	0.9897
40	198	0.0741	0.9851	0.9929	0.9929
44	206	0.0520	0.9891	**0.9954**	**0.9954**
48	216	0.0574	0.9871	0.9931	0.9931
52	228	0.0513	0.9893	0.9939	0.9939
56	238	0.0551	0.9884	0.9868	0.9868
60	267	**0.0493**	**0.9903**	0.9916	0.9916
64	275	0.0554	0.9869	0.9109	0.9109
68	269	0.0682	0.9853	0.9929	0.9929
72	276	0.0622	0.9863	0.9873	0.9873
76	296	0.0687	0.9842	0.9911	0.9911
80	307	0.0566	0.9873	0.9877	0.9877

Table 4
Classification results of five-fold cross-validation using dataset $\mathcal{D}_{\text{indoors}}$.

Fold	Accuracy	Precision	Recall	F1-score	AUC
1	0.9958	0.9916	0.9947	0.9932	0.9998
2	0.9967	0.9937	0.9958	0.9947	0.9999
3	0.9945	0.9895	0.9927	0.9911	0.9997
4	0.9946	0.9888	0.9937	0.9912	0.9998
5	0.9958	0.9931	0.9932	0.9932	0.9998
Total	0.9955	0.9913	0.9940	0.9927	0.9997

5.5. Experiment results

We performed a five-fold cross-validation on $\mathcal{D}_{\text{indoors}}$ to ensure that our 2D-CNN model is sufficient to detect intrusions and that the hyperparameters are properly determined. In each cross-validation, 80% and 20% of the samples are randomly selected as the training and validation sets, respectively. Each input is used once for validation. In each cross-validation, the training set and the validation set have the same proportion of benign/injected labels. After the cross-validation, we obtain five trained 2D-CNN models.

We tuned the batch size to 64 and epoch to 30 for training our model. To evaluate the final performance of our method, we performed five experiments using each model on $\mathcal{D}_{\text{driving}}$.

Table 4 shows the classification results obtained for the five-fold cross-validation. As our dataset is slightly imbalanced, we should focus on the F1-score (although the accuracy is better). The experimental results show that the model exhibited outstanding performance, with F1-scores from 0.9911 to 0.9947. Thus, the model can classify almost all stream AVTPDUs correctly. The high recall implies that the proposed model is very sensitive with regard to the detection of almost all injected stream AVTPDUs, even if the attacker sends AVTP traffic previously generated in the target automotive Ethernet-based network. We listed the validation results obtained for each cross-validation and created the “total” row in Table 4. In conclusion, we achieved a satisfactory performance of 0.99 or higher for all evaluation indicators for $\mathcal{D}_{\text{indoors}}$.

Notably, there is no significant difference in performance metrics among the five cross-validations. Hence, the training process is stable with the chosen hyperparameters. Moreover, our 2D-CNN model converges after the training phase regardless of input data. In other words, the experimental results obtained using $\mathcal{D}_{\text{indoors}}$

Table 5
Test results using dataset $\mathcal{D}_{\text{driving}}$.

Model	Accuracy	Precision	Recall	F1-score	AUC
1	0.9953	0.9821	0.9949	0.9885	0.9996
2	0.9930	0.9677	0.9990	0.9831	0.9992
3	0.9877	0.9439	0.9984	0.9704	0.9979
4	0.9923	0.9650	0.9984	0.9814	0.9996
5	0.9913	0.9600	0.9988	0.9790	0.9989

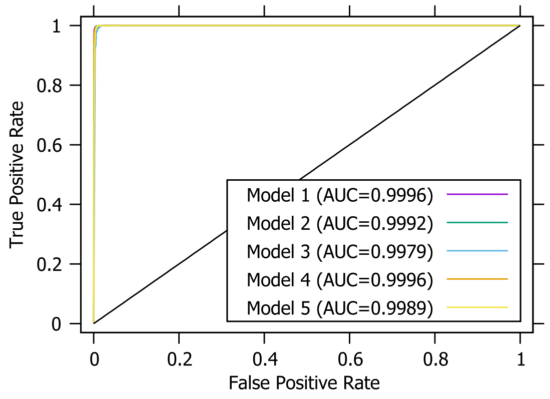


Fig. 7. ROC curve and AUC scores using test set $\mathcal{D}_{\text{driving}}$.

demonstrate that our feature generator, 2D-CNN model, and hyperparameters can effectively detect AVTP replay attacks.

Table 5 shows the test results obtained using $\mathcal{D}_{\text{driving}}$. We performed five tests using the five models that were trained during the cross-validation. We found that the overall performance was slightly reduced compared to that observed for the cross-validation test. This is mostly due to a decrease in precision, meaning that there are considerable *FP* cases in $\mathcal{D}_{\text{driving}}$. Interestingly, we confirmed that the recall had increased compared with that obtained for the validation result. Thus, we can conclude that our intrusion detection model (1) well distinguishes replayed stream AVTPDUs that are generally difficult to identify; however, (2) it may entail false-alarm fatigue during the attack-free period.

Fig. 7 shows that the AUCs are high although there was some performance reduction when the test set was employed. The ideal ROC curves and the high AUCs indicate that the 2D-CNN model well separates the input into benign and injected classes. It means that we can reduce the *FP* cases, thereby improving the overall performance by adjusting the threshold for evaluating Y'_i.

Table 6
Average inference time per sample for various devices.

Host	Processor	Time (μs/sample)
Google Colab	(GPU) NVIDIA Tesla P100	83
Macintosh	(CPU) Intel Core i7-7700K	787
Jetson TX2	(GPU) NVIDIA Pascal (256 CUDAs)	982
Raspberry Pi 3	(CPU) ARM Cortex-A53	35,000

To illustrate the input data and the classification results, we visualize four randomly selected X_is from our dataset in Fig. 8. Each dot expresses the extent of the state change of a nibble in the same position for adjacent stream AVTPDUs using brightness. The rightmost vertical bar represents $Y_{i-w+1}, \ldots, Y_i$ from top to bottom, in which blue and red dots mean "benign" (0) and "injected" (1), respectively.

Fig. 8(a) shows the typical state of stream AVTPDUs under the attack-free situation. It appears to have four *video frames* because there are three horizontal patterns in the MPEG2-TS header.

Fig. 8(d) shows the attacker dominating the AVTP stream. Except for the first injected packet, all injected packets appear to form a normal stream. We can observe that our 2D-CNN model successfully infers all intrusions correctly.

When a dominant injection continues for a substantially long period, e.g., when more than half of the window size corresponds to injection, the injected packets are sometimes misclassified (see the bottom of Fig. 8(c)) because the features appear similar to those in the attack-free state. This is because the replay attack was conducted using the traffic generated in the target automotive Ethernet. Nevertheless, our model correctly identifies most of the continuous packet injections.

One of the weaknesses of our model, found in Fig. 8(b), is that a single misclassification causes short-term continuous errors. In the visualization, we can observe five continuous classification errors.

5.6. Toward real-time detection

From a practical perspective, we need to consider not only the classification performance but also the inference time to determine whether the model is suitable for real-time detection in CAVs. To this end, we measured the average inference time per sample using four devices. Table 6 shows the results. For real-time detection, the inference time per sample must be less than or equal to the packet occurrence time that we noted earlier in Section 5.1, i.e., 1,735 μs. We considered the threshold to be 1,000 μs, considering packet injections by an attacker.

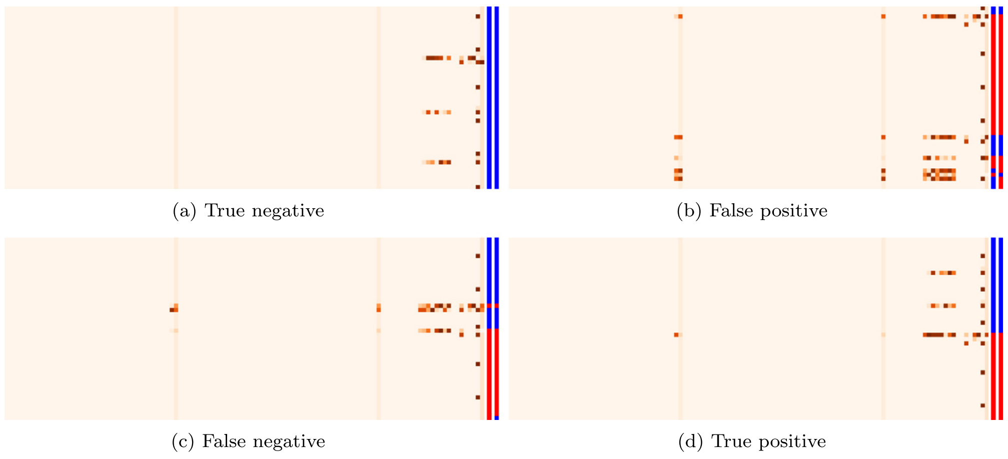


Fig. 8. Visualization of four randomly selected samples (X_is). Two vertical lines represent labels (left, $Y_{i-w+1}, \ldots, Y_i$) and predicted labels (right, $Y'_{i-w+1}, \ldots, Y'_i$). Blue and red dots mean that the traffic is (predicted as) "benign" and "injected," respectively.

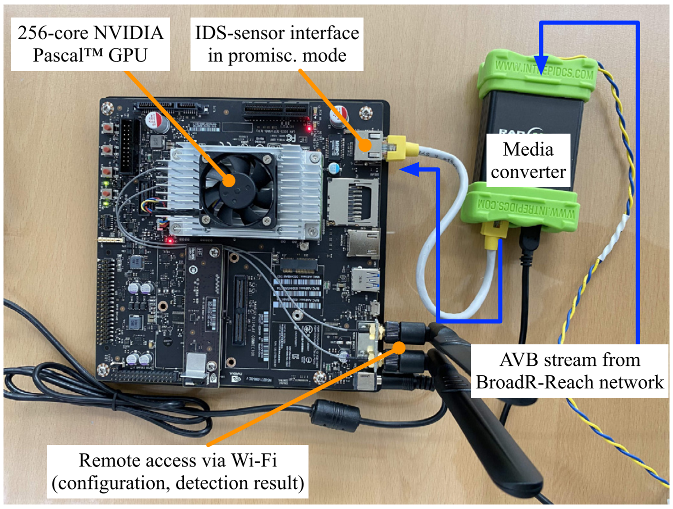

256-core NVIDIA Pascal™ GPU
IDS-sensor interface in promisc. mode
Media converter
AVB stream from BroadR-Reach network
Remote access via Wi-Fi (configuration, detection result)

Fig. 9. Implementation of the real-time IDS designed to identify injected stream AVTPDUs in a BroadR-Reach network.

In Google Colaboratory, our model takes only 83 μs to infer one sample with GPU acceleration, which is substantially shorter than the average packet interval. However, this result, obtained using a cloud artificial intelligence platform, does not reflect the situation in actual CAVs, because in-vehicle IDSs may not have such a powerful GPU. Therefore, we measured the inference time using a PC once again. Although the detection time increased substantially, it can be seen that real-time detection is possible using a CPU.

We further conducted the experiment on two embedded computers. We used a common device, Raspberry Pi 3, to show that the proposed model can function in a low-power computing environment. We successfully tested our approach on Raspberry Pi 3. However, the inference required a long time; hence, it was unsuitable for real-time detection. Next, we implemented an IDS for automotive Ethernet using an embedded GPU and a BroadR-Reach media converter. Fig. 9 shows the implemented IDS connected to our testbed via BroadR-Reach. We used an NVIDIA Jetson TX2 and an Intrepid RAD-Moon media converter. The proposed model requires approximately 982 μs to process one sample in this case. According to these results, our CNN model is suitable for real-time detection on GPU-enabled embedded computers.

In general, computation complexity and classification accuracy are in a trade-off relationship. However, it is required to reduce computation and inference time if the in-vehicle system provides less computational performance or an IDS runs on an embedded system along with multiple applications, such as perception, decision making, and on-board diagnostics. To reduce inference time, model optimization needs to be done with resizing input size, model compression, or quantization. In Section 5.4, we confirmed that the smaller window size we use, the shorter the model's training time. For instance, changing window size from 44 to 28 decreases 25% of training time and increases small misclassification error (about 0.26% of test F1-score), which is endurable. The reduction in learning time applies equally to inference time. To compress or quantize the proposed model, additional hyperparameters need to be carefully considered. This requires a detailed study and is left for future work.

6. Discussion

6.1. Dataset

During the experiment, we carefully created the automotive Ethernet intrusion datasets instead of conducting online learning. This helps us reproduce the experimental results and also lets other researchers study their own intrusion detection methods for automotive Ethernet. The datasets are recorded in the PCAP file format and, therefore, can be viewed using prevalent programming libraries and packet analyzers (such as Wireshark). As previously discussed, a replay attack was used as the intrusion technique. In this case, analysts without prior knowledge cannot distinguish the states of the packets as benign or injected. To address this, we have labeled each stream AVTPDU so that other researchers can design sophisticated experiments.

As part of our previous report, we released a CAN intrusion dataset [15]. Interestingly, we confirmed that many studies had been conducted using the published dataset. We hope that our intrusion dataset will trigger various studies on automotive Ethernet. Readers can refer to [6] to access our automotive Ethernet intrusion datasets.

6.2. Limitations

The proposed IDS must receive all stream AVTPDUs for intrusion detection continuously. As our IDS is designed to detect intrusions through continuous stream changes, intermittent traffic reception will cause numerous false positives. To solve this problem, we expect the implemented IDS to act as an AVB listener (see the AVB listener connected to the IDS in Fig. 2) because stream AVTPDUs are delivered to all participating AVB listeners.

Through the experiment, we confirmed that our model shows outstanding classification performance. However, the pretrained model exhibits a slight performance degradation when the packet capture environment is changed (from $\mathcal{D}_{\text{indoors}}$ to $\mathcal{D}_{\text{driving}}$). This is a common problem observed in deep learning models. Future studies can address this in two ways. First, a deep learning model should be designed to be more robust than the proposed model. A common method for achieving this is deploying additional layers in a model; however, this entails a trade-off relationship with the classification speed. Second, a transfer learning technique can be considered to improve classification performance in a new environment. In this way, the pretrained intrusion detection model can be further trained using new inputs (but not trained from scratch).

The proposed intrusion detection method needs additional hardware for real-time inference. The experiment result implies that our intrusion detection method requires GPU acceleration or a high-end CPU to detect intrusions in real-time. Real-time detection was not possible on small embedded computers such as Raspberry Pi 3, although our 2D-CNN model consisted of relatively few layers. Fortunately, real-time intrusion detection can be performed using a low-power embedded GPU.

6.3. Remediation strategies

In this study, intrusion detection was performed under the assumption that an attacker has access to an automotive Ethernet-based IVN. It is important to detect intrusions to protect IVNs. However, it is also essential to make it difficult for an attack to occur and to enable a quick response to an identified attack. In brief, we suggest the following remediation strategies:

1. Efforts are needed to reduce the attack surface. A firewall should be installed on known attack surfaces to prevent arbitrary packet injections. In addition, security checks should be performed on nodes that may be subject to external attacks (such as infotainment devices).
2. Encryption and authentication protocols should be employed so that passive attackers do not easily understand the internal communication process. For example, MACSec ensures the confidentiality and integrity of communication over automotive Ethernet [16].

3. It is possible to prevent bandwidth occupancy (i.e., DoS attacks) by using the stream reservation protocol. This protocol is designed to ensure quality-of-service by reserving and increasing the bandwidth for time-sensitive A/V streams on automotive Ethernet. The reserved bandwidth can prevent communication failures due to high traffic.
4. Finally, it is necessary to develop intrusion response systems to block an attack as soon as it is identified.

7. Related work

Corbett et al. [17] presented a pioneering study for Ethernet-based vehicle networks by analyzing Ethernet protocols designed for vehicle diagnostics, communication patterns, state-of-the-art tools, and attack scenarios. They also proposed a testing framework for an automotive IDS involving parameters, metrics, and automotive-specific challenges. However, their work only provides concepts regarding framework architecture and does not provide results of realistic experiments with an implemented IDS. Consequently, their testing tool needs to be evaluated.

Corbett et al. [18] also conducted an early study that discusses security mechanisms for automotive Ethernet-based IVNs. The authors identified possible manipulation and misuse attacks on automotive Ethernet-based networks, followed by an analysis of network topology and participating devices. The protocols and operating systems designed for vehicles were also introduced.

Readers interested in the current security trends, threats, and intrusion detection methods of IVNs are encouraged to refer to state-of-the-art reviews [3,5] and the references therein. The reviews also describe the constraints, challenges, and characteristics of IDSs for IVNs. This information will be of great help in further research on vehicle intrusion detection methods. However, these reviews lack citations of Ethernet- and automotive Ethernet-specific studies.

Some research was conducted on network packet classification using deep learning techniques. Kim and Anpalagan [19] adopted a 1D-CNN model to classify encrypted Tor traffic using only the TCP/IP header. Their data processor extracts the first 54 bytes of each payload and then split bytes into nibbles. The nibbles are used as the inputs of the 1D-CNN model. The experimental results show that this model not only performs binary classification well but can also infer specific application types. Wang et al. [20] converted packets to 28×28 sized gray images to classify malware traffic using a CNN model. This CNN model has the advantages of early-stage detection, low false-alarm rate, protocol-independency, automatic extraction of features, and raw traffic data input. On a CAN-based IVN system, Song et al. [21] adopted a deep CNN model–Inception-ResNet–to classify four types of injection attacks within the CAN bus. Their frame builder is designed to represent a sequence of CAN IDs at a bit level. Taylor et al. [22] also conducted a study to detect CAN bus attacks using long short-term memory (LSTM) networks.

Changing the nature of the packets and environment degrades intrusion detection performance when a pretrained deep learning model is used. For instance, this degradation is apparent from Table 5. Moreover, new types of attacks could not be properly detected by existing models. To address the problem and identify new types of intrusion attacks, Tariq et al. [23] applied transfer learning. They trained a convolutional LSTM network using a DoS attack dataset captured from the CAN bus. Moreover, they tried one-shot learning using only a few fuzzing and replay attacks.

8. Conclusion

In this paper, we proposed an intrusion detection method that detects AVTP stream injection attacks in automotive Ethernet. To the best of our knowledge, this is the first intrusion detection method for protecting automotive Ethernet-based IVNs. Our approach includes feature generation and a CNN-based intrusion detection model. We design a feature generator that measures state changes of the AVTP stream at the nibble-level within a specific time window. The CNN model extracts features from the input and infers whether a stream AVTPDU is benign or injected. The experimental results show that our model achieved a high detection performance and remarkably high recall for real stream AVTPDUs captured from our BroadR-Reach-based testbed. We prove that our model is suitable for real-time detection in an actual CAV by measuring the time required for inference. Although we limited our experiments to stream AVTPDUs, we will also consider other AVB-related protocols and automotive diagnostics communication protocols in future work. Finally, we have released the dataset collected for this study. We look forward to further studies on automotive Ethernet security based on our AVTP stream dataset.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00312, Developing technologies to predict, detect, respond, and automatically diagnose security threats to automotive Ethernet-based vehicle).

References

[1] P. Hank, S. Muller, O. Vermesan, J. Van Den Keybus, Automotive Ethernet: in-vehicle networking and smart mobility, in: Design, Automation & Test in Europe Conference & Exhibition, DATE, 2013, IEEE Conference Publications, New Jersey, 2013, pp. 1735–1739.
[2] IEEE P1722 working group, IEEE standard for a transport protocol for time-sensitive applications in bridged local area networks, https://doi.org/10.1109/IEEESTD.2016.7782716, 2016.
[3] G. Loukas, E. Karapistoli, E. Panaousis, P. Sarigiannidis, A. Bezemskij, T. Vuong, A taxonomy and survey of cyber-physical intrusion detection approaches for vehicles, Ad Hoc Netw. 84 (2019) 124–147, https://doi.org/10.1016/j.adhoc.2018.10.002.
[4] F. Panarotto, A. Cortesi, P. Ferrara, A.K. Mandal, F. Spoto, Static analysis of Android apps interaction with automotive CAN, in: International Conference on Smart Computing and Communication, SmartCom, 2018, pp. 114–123.
[5] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, K. Li, A survey of intrusion detection for in-vehicle networks, IEEE Trans. Intell. Transp. Syst. 21 (3) (2020) 919–933, https://doi.org/10.1109/TITS.2019.2908074.
[6] S. Jeong, B. Jeon, B. Chung, H.K. Kim, Automotive Ethernet intrusion dataset, IEEE DataPort (2021), https://doi.org/10.21227/1yr3-q009.
[7] B. Metcalfe, C.M. Kozierok, C. Correa, R.B. Boatright, J. Quesnelle, M. Holden, K. Irving, Automotive Ethernet, Intrepid Control Systems, 2014.
[8] S.A.J. Alabady, Design and implementation of a network security model using static VLAN and AAA server, in: 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, IEEE, 2008, pp. 1–6.
[9] E. Vyncke, Layer 2 security, https://www.cisco.com/c/dam/global/da_dk/assets/docs/security2006/Security2006_Eric_Vyncke_2.pdf, 2006.
[10] D. Rezvani, Hacking automotive Ethernet cameras, https://argus-sec.com/hacking-automotive-ethernet-cameras/, 2018.
[11] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: International Conference on International Conference on Machine Learning, 2015, pp. 448–456, http://jmlr.org/proceedings/papers/v37/ioffe15.pdf.
[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.
[13] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv Preprint, http://arxiv.org/abs/1412.6980, 12 2014.
[14] F. Chollet, et al., Keras, https://keras.io, 2015.

[15] H. Lee, S.H. Jeong, H.K. Kim, OTIDS: a novel intrusion detection system for in-vehicle network by using remote frame, in: 2017 15th Annual Conference on Privacy, Security and Trust, PST, vol. 5, IEEE, 2017, pp. 57–5709.

[16] B. Carnevale, L. Fanucci, S. Bisase, H. Hunjan, MACsec-based security for automotive Ethernet backbones, J. Circuits Syst. Comput. 27 (5) (2018) 1–17, https://doi.org/10.1142/S0218126618500822.

[17] C. Corbett, T. Basic, T. Lukaseder, F. Kargl, A testing framework architecture concept for automotive intrusion detection systems, in: Proceedings - Series of the Gesellschaft fur Informatik (GI) P-269, in: Lecture Notes in Informatics (LNI), 2017, pp. 89–102.

[18] C. Corbett, E. Schoch, F. Kargl, P. Felix, Automotive Ethernet: security opportunity or challenge?, in: Lecture Notes in Informatics (LIN), 2016, pp. 45–54, http://www.gi.de/service/publikationen/lni/.

[19] M. Kim, A. Anpalagan, Tor traffic classification from raw packet header using convolutional neural network, in: 2018 1st IEEE International Conference on Knowledge Innovation and Invention, ICKII, no. January, IEEE, 2018, pp. 187–190.

[20] Wang Wei, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, Yiqiang Sheng, Malware traffic classification using convolutional neural network for representation learning, in: 2017 International Conference on Information Networking, ICOIN, IEEE, 2017, pp. 712–717.

[21] H.M. Song, J. Woo, H.K. Kim, In-vehicle network intrusion detection using deep convolutional neural network, Veh. Commun. 21 (2020) 100198, https://doi.org/10.1016/j.vehcom.2019.100198.

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

[23] S. Tariq, S. Lee, S.S. Woo, CANTransfer - transfer learning based intrusion detection on a controller area network using convolutional LSTM network, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2020, pp. 1048–1055.