electronics

MDPI

Article

A Formal Modeling and Verification Scheme with an RNN-Based Attacker for CAN Communication System Authenticity

Yihua Wang [1,2], Qing Zhou [1,*], Yu Zhang [3], Xian Zhang [3] and Jiahao Du [1]

[1] Key Laboratory of Electronics and Information Technology for Space System, National Space Science Center, Chinese Academy of Sciences, Beijing 100190, China; wangyihua19@mails.ucas.ac.cn (Y.W.); dujiahao@nssc.ac.cn (J.D.)
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China
[3] Beijing Jinghang Research Institute of Computing and Communication, Beijing 100074, China; zhangyu03092022@163.com (Y.Z.); 13691572171@163.com (X.Z.)
* Correspondence: zhouqing@nssc.ac.cn

Abstract: To enhance the attack resistance of the Controller Area Network (CAN) system and optimize the communication software design, a comprehensive model that combines a variable attacker with the CAN bus (VACB) is proposed to evaluate the bus communication risk. The VACB model consists of a variable attacker and the CAN bus model. A variable attacker is a visualized generation of the attack traffic based on a recurrent neural network (RNN), which is used to evaluate the anti-attack performance of the CAN bus; the CAN bus model combines the data link layer and the application layer to analyze the anomalies in CAN bus data transmission after the attack message. The simulation results indicate that the transmission accuracy and successful response rate decreased by 1.8% and 4.3% under the constructed variable attacker. The CAN bus's authenticity was promoted after the developers adopted this model to analyze and optimize the software design. The transmission accuracy and the successful response rate were improved by 2.5% and 5.1%, respectively. Moreover, the model can quantify the risk of potential attacks on the CAN bus, prompting developers to avoid it in early development to reduce the loss caused by system crashes. The comprehensive model can provide theoretical guidance for the timing design of embedded software.

Keywords: controller area network bus; anti-attack; attack traffic generation; recurrent neural network; model checking; UPPAAL

Citation: Wang, Y.; Zhou, Q.; Zhang, Y.; Zhang, X.; Du, J. A Formal Modeling and Verification Scheme with an RNN-Based Attacker for CAN Communication System Authenticity. *Electronics* **2022**, *11*, 1773. https://doi.org/10.3390/electronics11111773

Academic Editor: Cheng-Chi Lee

Received: 3 April 2022
Accepted: 27 May 2022
Published: 2 June 2022

1. Introduction

The Controller Area Network (CAN) bus is a high-speed serial fieldbus with extraordinary performance. It plays an indispensable role in aerospace, where the safety of people and space vehicles is always prioritized. Therefore, significant risks, such as path deadlock, timing conflicts, software failure, and attacker intrusion, must be rigorously simulated and verified to reduce tragic losses [1]. To identify software system design flaws as early as possible and improve the safety, reliability, and efficiency of the system, it is preferable to conduct system modeling in the software requirements analysis phase. It is worth noting that, in aerospace and rail transportation [2], the software in embedded systems is inseparable from the external hardware equipment and the operating environment and is expected to be robust. The formal method [3] has been widely introduced in communication protocols and bus verification. It is able to provide rigorous, efficient, and interpretable verification for state transitions and timing constraints with mathematical descriptions.

In recent years, several excellent models on the CAN bus formal verification have been proposed. Krakora proposed the timed automata (TA) based on the CAN bus model

on the data link layer of the protocol [4]. A CAN bus model in a CPS system was constructed [5], and the fault-tolerant clock synchronization mechanism of the CAN bus was verified in [6]. Furthermore, the CAN application layer modeled in [7] mainly verified the message scheduling algorithm and node recovery mechanism. The authors in [8] modeled and verified the joint communication of the robot controlled by the CAN bus. In the aspect of protocol security, a formal model based on the MAuth-CAN protocol [9] was constructed and indicates that the MAuth-CAN protocol is resilient to flooding attacks. However, these studies did not further analyze the transmission performance of the CAN bus, and the potential instability was ignored when assessing the plausibility of the software requirements.

Fortunately, studies on communication bus intrusion attacks have constantly attracted the interest of researchers up to the present. In terms of intrusion attack detection, attack traffic detection systems are implemented by training feature vector parameters to distinguish messages through machine learning methods or neural networks [10–13]. However, such studies are limited to detecting the legitimacy of messages. In addition, the attack datasets used are simulated by manual interleaving, reversing, or dropping. In terms of attack traffic generation, a tool for DoS attack injection named Carshark was proposed in [14], and it mainly uses CAN bus interface reprogramming to generate message data. Another tool of CAN bus attack traffic generation (ATG) [15] was developed with customized configurations of attack types. Similarly, Hayward constructed a CAN bus attack simulator based on CANoe operating attack parameters [16]. These studies have made many attempts and promotions on packets of data reorganization and configuration. It is noteworthy that attack traffic generation combined with deep learning methods is a valuable direction. Ring used neural networks to learn the features of data and proposed an unsupervised method, IP2Vec [17], to learn the vector representation of IP addresses. Subsequently, they use GAN [18] to generate flow-based network data. However, their method can only generate the attack traffic previously seen in the target network. Yanjie proposes a framework for attack traffic generation based on migration learning [19]. The framework extracts attack invariants from existing attack datasets and generates new anomalous traffic for the target industrial control system. However, their study is still on the IP protocol, which is still quite different from CAN bus data. Qiumei proposed Attack-GAN [20] based on SeqGAN to generate domain-constrained adversarial network traffic, capable of estimating partially generated packets and complete packets. However, the purpose of this study focuses on circumventing intrusion detection systems using the CTU-13 dataset. In general, methods for modeling the attacker and formally verifying the anti-attack ability of the CAN bus are still deficient.

To address the limitations of existing models for the verification of CAN bus robustness, a comprehensive model that combines the variable attacker with the CAN bus (VACB) is proposed. The design steps of this comprehensive model are as follows: (1) An attacker through formal modeling based on an RNN (FMA-RNN) is constructed. The FMA-RNN model can generate malicious network packets against the CAN bus. (2) A formal model of the CAN bus system is established based on the CAN bus protocol and application layer requirements. (3) The attack traffic generated by the FMA-RNN model is used to verify the performance of the CAN bus system model. Using UPPAAL SMC, the statistical probability of bounded simulation is given for the transmission and response behavior of the CAN bus based on the VACB comprehensive model. Based on the above model design and validation, the VACB model can quantify the risk of the CAN bus attacks effectively and mitigate the impact of the FMA-RNN on bus communication authenticity. The VACB model is able to assist developers in optimizing the timing design of communication software and improve the robustness of the CAN bus system.

2. Design Principle

The anti-attack performance of the CAN bus system is measured with diverse application designs to verify its temporal security under the risk of incursion. Typically, there are three types of temporal requirements for the communication of aerospace embedded software. The first type consists of bus transmission period requirements: the inter-frame interval period of composite frames does not exceed 0.5 ms; the time code broadcast period is 1 ± 0.2 s; and the response period of polling a slave node is 0.5 ms. The second type consists of reply period requirements: the response duration does not exceed 2 ms; otherwise, it is regarded as a timeout. The third type consists of the CAN bus fault recovery requirements: if the count of the transmit error register or receive error register is greater than 90, the bus controller is reinitialized, and the software initialization time is less than 50 s.

The above timing constraints design can meet the conventional requirements for transmission, but the software is continually exposed to various risks of attack in actual operation. Aerospace software is strongly influenced by space environment effects, which can cause single event effects, locking, or even burn-ups when transmitting data, resulting in abnormal communication. In addition, the dysfunction of the load may break down and send anomalous messages to the CAN bus continuously. Moreover, malicious attacks are almost inevitable in most applications of the CAN bus.

In this paper, we implement the attacker model for the representative attacks (Fuzz, DoS, etc.) based on the background of the above application-specific requirements. In addition, since CAN messages contain long data fields, CAN bus devices usually set mask words to filter out messages that are not relevant to them. Therefore, a simulation for the ID field of CAN messages can improve the arithmetic effect of model checking and ensure that the data is a valid attack. This can be done by simulating multifarious data errors and temporal errors based on the FMA-RNN, and such errors are reckoned as the variable attacker to verify the performance of the CAN bus.

2.1. Recurrent Neural Network

The FMA-RNN is built on a recurrent neural network [21], which is theoretically Turing-complete [22] and can process arbitrary input sequences. The improvement of the LSTM generation effect is not much compared with RNN, and the transformation model is too complex and operationally overloaded. Therefore, we select RNN for attack traffic generation. The RNN establishes connections of weights between neurons of different layers. At the initial moment, there is no output from the previous hidden layer; the output of each subsequent moment is related to the input of the current moment and the output of the previous moment. That is, each moment passes part of its information to the next moment. Therefore, the RNN is able to learn not only the features of the samples but also the serial correlation between the samples.

Fuzzy testing [23], which aims to find security vulnerabilities, can be conducted by sending unintended message data to the CAN bus. We use the RNN to predict the new message sequence for the next time series, which means the attack traffic can be regarded as a fuzzy test case. According to the normal and attack messages obtained on the CAN bus, the RNN is applied to implement an encoder and a decoder. After training, the network can generate messages conforming to the CAN protocol format but with data variants that are introduced into the model as attack traffic.

The network structure design is described in Section 3.1.2. The computational principles of the basic RNN unit are as follows:

$$h_t = f(Ux_t + b + Wh_{t-1}) \tag{1}$$

$$o_t = \text{softmax}(Vh_t + c) \tag{2}$$

where x_t is the input at moment t, h_t stores memory information at moment t, and o_t is the output at moment t. U, W, V, b, and c are different weight matrices. $f()$ denotes the activation function.

2.2. Formal Method

Formal methods are adopted for the verification of the VACB model. The techniques of formal verification include model checking and theorem proving [3]. Model checking is widely adopted in the verification of various protocols. We used UPPAAL [24] to verify whether the protocol meets the safety properties. The modeling language of UPPAAL is time automaton, which belongs to the finite-state machine and is represented as a tuple: (L, l_0, C, A, E, I) [25]. The verification formula consists of a path formula that quantifies the branching of the model path and a state formula that describes the individual states. Assuming $\varnothing$ is a state formula, the syntax of the validation formula is as follows:

$$\varphi ::= \varnothing \mid A\Box\varnothing \mid E\Box\varnothing \mid A\Diamond\varnothing \mid E\Diamond\varnothing \mid \varnothing_1 \rightsquigarrow \varnothing_2 \tag{3}$$

where A and E are branching quantifiers, representing "all paths" and "exist paths"; $\Box$ and $\Diamond$ are unary modal operators, representing "all states" and "exist states". The above formulae express three different properties: reachability, safety, and liveness [24].

This paper uses UPPAAL SMC [26] based on the statistical probabilistic for model checking, which substantially reduces the pernicious effects of path state explosion. The verification syntax for UPPAAL SMC is as follows:

$$\varphi ::= ap \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid O\varphi \mid \varphi_1 \bigcup_{\leq d}^{x} \varphi_2 \tag{4}$$

The following equation is used in SMC validators to describe the following properties:

$$\text{Simulate N } [\leq \text{bound}]\{ \text{E1}, \ldots \text{Ek} \} \tag{5}$$

$$\Pr[\text{bound}](\psi) \text{ or } \Pr[\text{bound}_1](\psi_1) \geq \Pr[\text{bound}_2](\psi_2) \tag{6}$$

$$\text{E}[\text{bound}; \text{N}](\min : \text{expr}) \text{ or E}[\text{bound}; \text{N}](\max : \text{expr}) \tag{7}$$

Equation (5) requests that the checker provides the number of N simulations and runs them over bound time units. Probability estimation, hypothesis testing, and probability comparison can be achieved by Equation (6). Equation (7) calculates the expected min or max values of an expression that evaluates to a clock or an integer value.

UPPAAL SMC is a probabilistic checker based on a bounded model, and the results have a certain probabilistic randomness. This characteristic will cause the experimental results to be approximate rather than exact, but it does not prevent us from using SMC to analyze the safety performance of the CAN bus within a specific range quantitatively. SMC can describe the trend of system performance powerfully and qualitatively. Therefore, this research can provide a theoretical reference value of timing constraints for the design of the CAN bus communication software.

3. The Proposed VACB Model

3.1. Modeling of the FMA-RNN

This section shows the whole process of building the FMA-RNN. We implemented visual white-box modeling of the RNN, which can ensure the correctness and accuracy of the generation model. First, the RNN is trained to generate the sequence, which includes acquiring and processing the CAN bus packet datasets, designing a neural network framework, and optimizing the training results. Second, the model transformation algorithm is introduced, and the automaton of FMA-RNN is drawn in UPPAAL based on the algorithm.

3.1.1. Data Set Preprocessing

To facilitate the measurement of the CAN bus attack resistance, data similar to the "real messages" are generated as fake messages. Due to the temporal correlation of packets, the RNN is trained to learn the features of the front four message IDs and predict the fifth sequence by grouping every five message IDs. The dataset used for training was obtained from the open dataset provided by Seo [13]. It can be divided into two categories: the first category is normal and the second category is a mixture of data containing both normal and attack messages. Specifically, the latter includes DoS attacks, fuzzy attacks, and RPM/GEAR attacks. Figure 1 shows the data format where the tag T indicates an attack message and the tag R indicates a normal message.

Timestamp	**CAN ID**	DLC	DATA[0~7]	**Flag**
Recorded Obtain Time	Identifier of CAN Message in HEX	Number Count	Up to 8 bytes (Length Specified by DLC)	T: Injected message R: Normal message

Figure 1. Message format of the Controller Area Network (CAN) bus packets.

The CAN ID in the message dataset is preprocessed using one-hot coding, which is mapped into a 3×16 two-dimensional matrix. The input of the RNN encoder is a 12×16 matrix of four sequential IDs. The network learns the message format and temporal characteristics to predict the probability distribution of the following ID. Finally, the output is compared with the label to evaluate the effectiveness of the RNN model. Figure 2 shows the ID prediction process and one-hot coding schematic diagram.

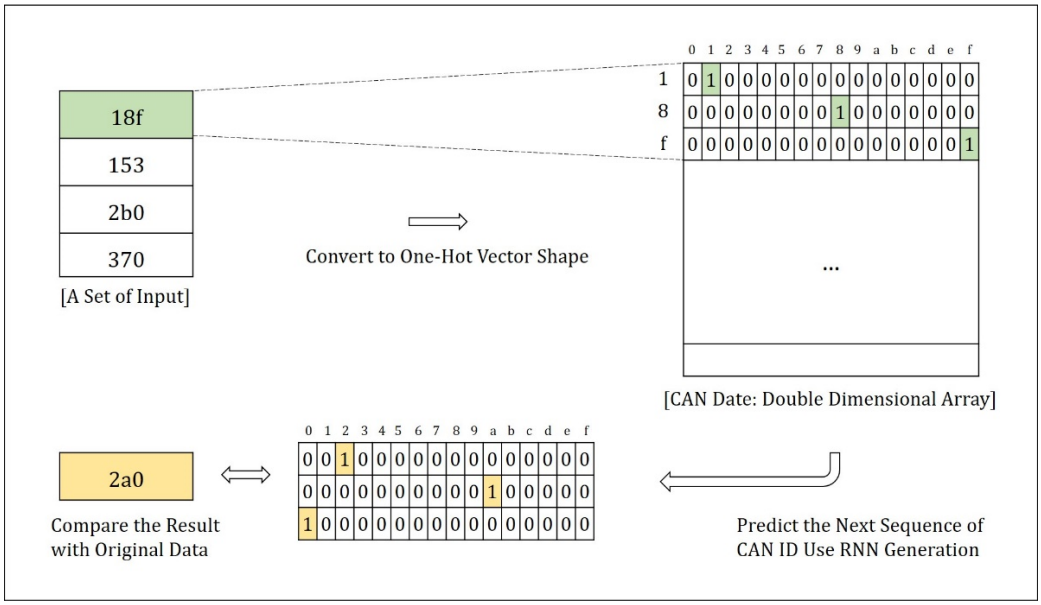

Figure 2. CAN ID prediction and one-hot coding schematic.

3.1.2. Train Strategy of the RNN

The prediction effect and the ease of model change are synthetically considered in the RNN design, where the encoder has a "many-to-one" structure and the decoder has a "many-to-many" structure. Figure 3 shows the time series expansion model of the RNN. The encoder and decoder are designed as follows:

1. Encoding: The encoder consists of 12 timesteps that process the 12 CAN ID sequences of the input in turn. The output of each timestep depends on both the current input and the previously hidden layer output; K1 is the weight matrix from the input layer

to the hidden layer, and R1 is the weight matrix from the hidden layer to the output layer. The activation function is tanh, and the output is a 1×8 coding matrix.

2. Decoding: According to the decoding layer design, the encoded features are copied into a sequence with a timing of 3 as the input of the decoding layer. The decoder contains 3 timesteps, and the activation function for each layer is tanh. Meanwhile, the output y_i at each step is used as both the output of the current timestep and the input to the hidden layer at the next timestep.

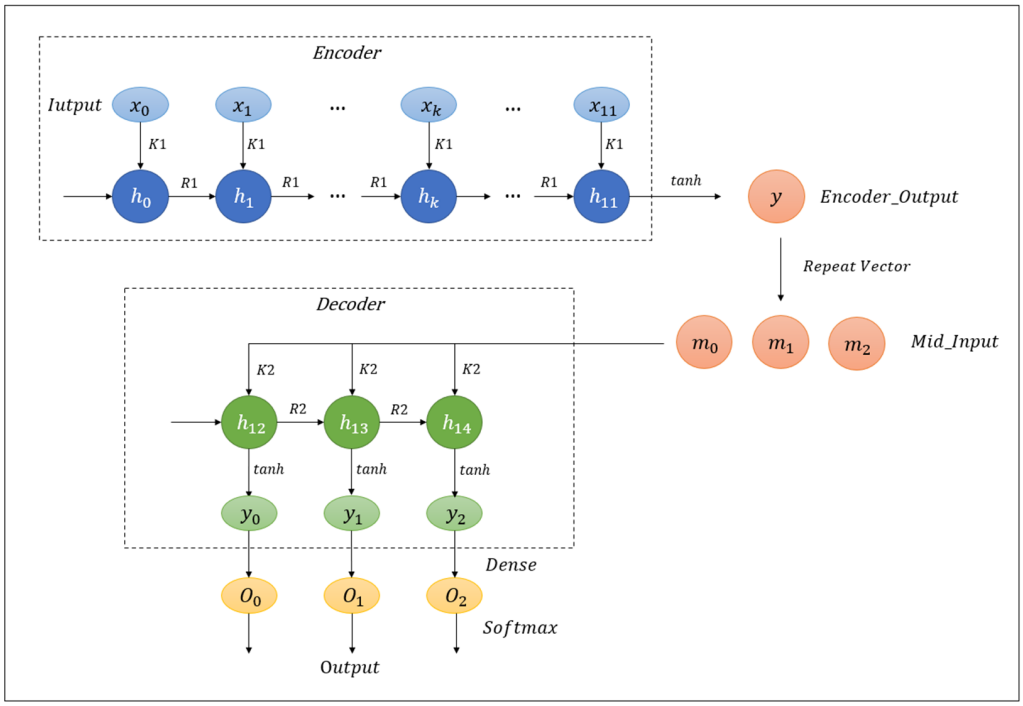

Figure 3. Time series expansion model of the recurrent neural network (RNN).

The replication of encoder_output is taken as the mid input (m_0, m_1, m_2) of the decoder. The output matrix of the decoder is subjected to dense and softmax operations, denoting o_i in Figure 3. The cross-entropy loss function is used as the objective function and is defined as:

$$\text{Loss} = \sum_{t=1}^{N} \text{Loss}_t = -\sum_{t=1}^{N} [v_t \ln(p_t) + (v_t - 1) \ln(1 - p_t)] \quad (8)$$

where v_t is the true label value of the input at moment t, p_t is the model's predicted value, and N represents all N moments.

The final output is a probability distribution of the predicted values of the message ID sequence, and the generated message data can be obtained through probability transformation.

We adopted accuracy and precision as two evaluation indexes to measure the performance of the RNN. The accuracy is defined as the ratio of correct prediction results in total samples; the precision is defined as the ratio of actually positive samples in all predicted positive samples. As shown in Table 1, the results in different datasets show that the average accuracy of the RNN generator is 71.36%, and the average precision is 77.44%, which is close to the real data that can be used as attack traffic.

3.1.3. Model Transformation

This section introduces the modeling of the FMA-RNN. It starts with the model transformation algorithm, which can ensure the correctness and accuracy of the generation model. Furthermore, it can visualize the computation process inside the network and avoid the effects of illegal, tiny perturbations in neural networks. The neural network model is structurally similar to the timed automata model. Specifically, the layer of the

neural network can be mapped to the states in TA; the data in the data flow can be defined, computed, and passed on; the weight calculation and activation functions in the neural network can be mapped to the update assignments of TA; and the performance of the neural network can be evaluated by the query formula or guard conditions in UPPAAL. In combining deep learning with formal verification, a method for modeling DNNs as TA is proposed in [27] to verify the neural networks' correctness and gives a conceptual paradigm. This paper first implements the transformation in practical applications based on real-life projects. The FMA-RNN can run on UPPAAL and generate the attack traffic for the follow-up CAN bus system.

Table 1. Accuracy and precision of the RNN generator.

		Attack Free	Dos	Fuzzy	Gear	RPM
Accuracy		77.09%	74.68%	63.79%	71.87%	69.38%
Precision	class 0	80.01%	75.75%	64.68%	74.78%	78.16%
	class 1	77.79%	81.04%	74.20%	81.05%	67.47%
	class 2	87.97%	88.73%	80.86%	91.36%	83.21%
	class 3	68.26%	63.13%	65.03%	60.95%	63.73%
	class 4	72.78%	67.77%	64.64%	67.29%	71.52%
	class 5	69.63%	66.52%	69.77%	66.03%	66.19%
	class 6	83.12%	83.37%	81.26%	86.72%	56.30%
	class 7	86.29%	83.36%	77.00%	75.34%	74.71%
	class 8	92.79%	89.93%	80.99%	89.77%	66.33%
	class 9	85.85%	84.17%	80.87%	83.01%	64.03%
	class a	97.28%	91.19%	90.86%	94.23%	89.45%
	class b	88.48%	84.20%	79.75%	77.70%	77.92%
	class c	74.01%	74.88%	90.87%	63.79%	47.97%
	class d	88.09%	89.41%	82.92%	56.68%	76.46%
	class e	94.39%	94.35%	75.43%	82.60%	52.14%
	class f	83.14%	97.24%	86.95%	73.01%	44.15%
	average	83.12%	82.20%	77.89%	76.52%	67.48%

The pseudo-code for transforming is given in Algorithm 1. The algorithm takes the parameters of the RNN network, including the number of time rounds, the weight matrix, the bias matrix, the activation function, and the number of structural layers as input. Furthermore, x_t denotes the data flow for each computation, and h_i denotes the current hidden layer output, which is also the input for the next time round of computation. Moreover, Lines 3 to 9 of the algorithm describe the process in each time round of the RNN. After the activation layer, a guard function with a return type of Bool is used to verify whether the RNN satisfies the function's required quantitative relationships.

Next, the timed automaton of FMA-RNN is shown in Figure 4. The model FMA-RNN can calculate a random input and then generate a message sequence in CAN format. Next, the parameters of the timed automaton model are described in detail. The function allows the calculation of the input according to the parameters of the neural network step by step. The timestep of the encoder RNN is time1. The parameters of function inputEncoder() are the input matrix, the encoding weight matrix weightK_Encoder, and the current time round time1. The function hiddenEncoder() calculates the hidden layer output by multiplying the input with the weight matrix weightRK_Encoder as the output_Encoder. Iteratively, the data is computed in a time sequence and activated through the function tanhEncoder() as an output. Moreover, the calculation of the decoder RNN is similar to that of the encoder RNN. The output of the decoder is multiplied by the weight matrix weight_Dense, and the prediction result is calculated by softmax. The function propertyGuard() is used to check the prediction result. If OK, the message sequence is successfully generated as expected and then returned to the initial position.

Algorithm 1 Convert the RNN Model to Timed Automata

Input: Time round of RNN: t; the weight of RNN: K_i, R_i, V_i; the bias of RNN: b, c; list of layers for $L = \langle l_1, l_2, \ldots, l_n \rangle$;
Activation function of RNN $F = \langle f_1, f_2, \ldots, f_n \rangle$.
Output: Abstract automaton of RNN

```
node_1 ← L[0]
for index i of layers do
    if L[i] is encoder or decoder layer then
        for time round of RNN t do
            T_(i+1)i ← y_1 = K_i ⊗ x_t + b
            T_(i+2)(i+1) ← y_2 = R_i ⊗ h_(t−1)
            T_(i+3)(i+2) ← h_t = f_i(y_1 + y_2)
        end for
    end if
    if L[i] is activation layer then
        T_(i+1)i ← O_i = f_i(V_i ⊗ h_i + c)
    end if
end for
T_(n+1)1 ← Guard()
return FMA-RNN
```

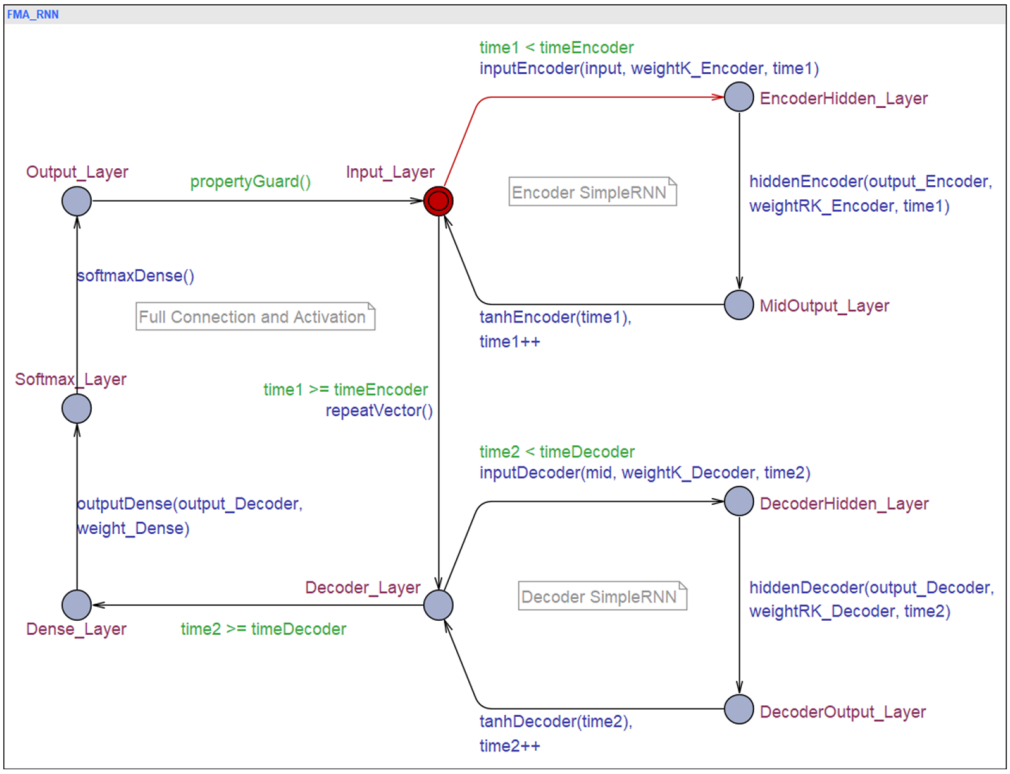

Figure 4. Transformed formal model: FMA-RNN in UPPAAL.

3.2. Modeling of the CAN Bus System

This section introduces the formal construction of the CAN bus. The model abstracts the critical behaviors of the master controller and slave nodes for data sending and receiving, fault handling, priority arbitration, and application layer constraints. Furthermore, the VACB model serves as a foundation for the checking of system properties. Prior to the description of the CAN bus model, the symbols and variables for formal descriptions are defined in Table 2.

Table 2. Symbol and variable definitions.

UPPAAL	Description
msg_Num	Number of messages
msg[i][j]	Double dimensional array of CAN network packets
rep[i][j]	Reply message array
typedef cid_t	Custom types: txNode with id I; values are between 0 and N-1
typedef mid_t	Message with id e; values are between 0 and msg_Num-1
signal[i]	Initial signal flag bit on the CAN bus
canState	Predicate to indicate whether CAN bus is occupied or not
ArbState	Predicate to indicate whether arbitration is going on or not
arb[i][e]	Arbitration message e released by node i
t_UppLim	Upper limit of the reply time from the slave node
t_Done_Time	Maximum message sending time
txErr_Cnt	Transmission error count on the CAN bus

According to the application software requirements, a five-node model with one master and four slave nodes of the CAN bus is constructed. The master has the highest priority, and the slave nodes have to obey the two rules: do not communicate with other slave nodes, and do not send messages to other nodes on the bus without inquiry. The transmit error counter (TEC) is maintained when there is a node crash or flooding attack on the CAN bus, and once beyond, the bus is reset and reinitialized.

Due to the time-efficient communication of the CAN bus, the bus's resistance to attack can be verified by whether the message is successfully sent and whether the node can respond within the required time under attack. The attack traffic can be simulated by the FMA-RNN, and the interaction diagram between the CAN bus components is shown in Figure 5. The state transitions of the master controller during transmission are shown in Figure 5a. Messages can only be transmitted when the bus is idle, and they must be checked before sending. The messages can be identified into four categories: messages that do not need a reply, messages that need a reply, reset messages, and error messages. These situations correspond to synchronization send!, synchronization reply!, synchronization reset!, and executing the error count incremented, respectively. Figure 5b shows the state transitions in the application layer. The model abstracts the node actions and time constraints during the data transmission. Meanwhile, both sending timeouts and packet loss cases are taken into account. The transitions of slave node reply activity are shown in Figure 5c. In our design, the slave node template is instantiated into four processes. If more than one slave node requests to send messages simultaneously, arbitration must be completed before the transmission. Moreover, the arbitration transitions when receiving an arb[i][e]? signal are shown in Figure 5d. The transmission time consists of the sending time and the arbitration time. The size of the state space is reduced by setting the bitwise arbitration operation to be executed atomically, thus speeding up the verification. The total arbitration time is measured by adding invariant constraints for key nodes and setting the delay rate of the location.

3.2.1. Master Controller

As shown in Figure 6, the timed automaton named "Main_Controller" represents the master controller on the CAN bus. The model summarizes the different operations when the master controller node transfers the message on the CAN bus. The probabilistic branch of the UPPAAL SMC is adopted to implement random combinations of different messages sent in various orders. When the value of canState and ArbState is 0 simultaneously, it means that the bus is available and the message can be sent. Meanwhile, there are four branch paths after the location MsgIdentify. The function guard_MSGType() identifies the message ID, returns the interrelated identifier, and executes different procedures according to the identifier.

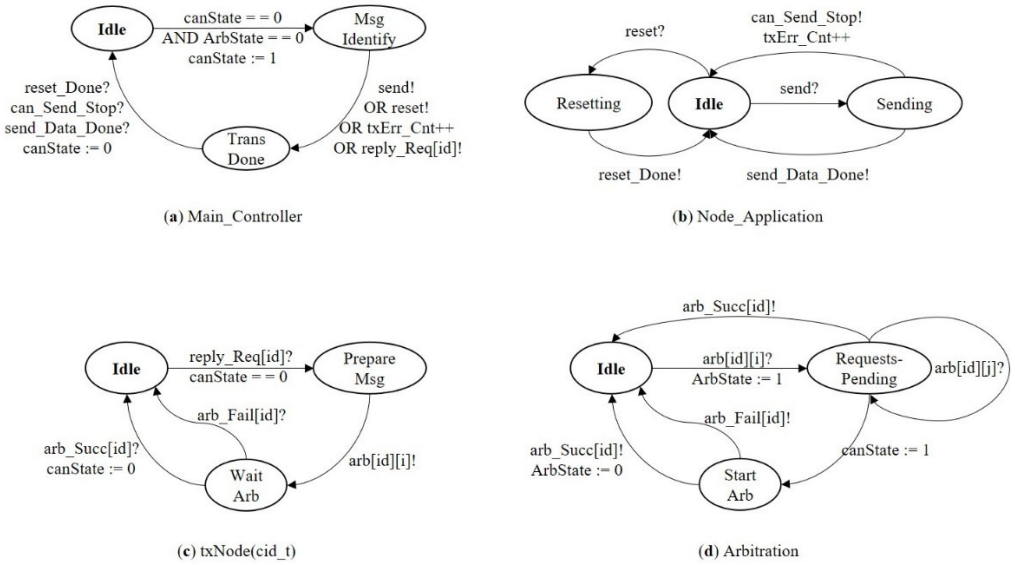

Figure 5. The interactive model of the CAN bus. (**a**) The state transitions of the master node. (**b**) The behavior patterns and state transitions of the nodes. (**c**) The state transitions of the slave node. (**d**) Arbitration state transitions.

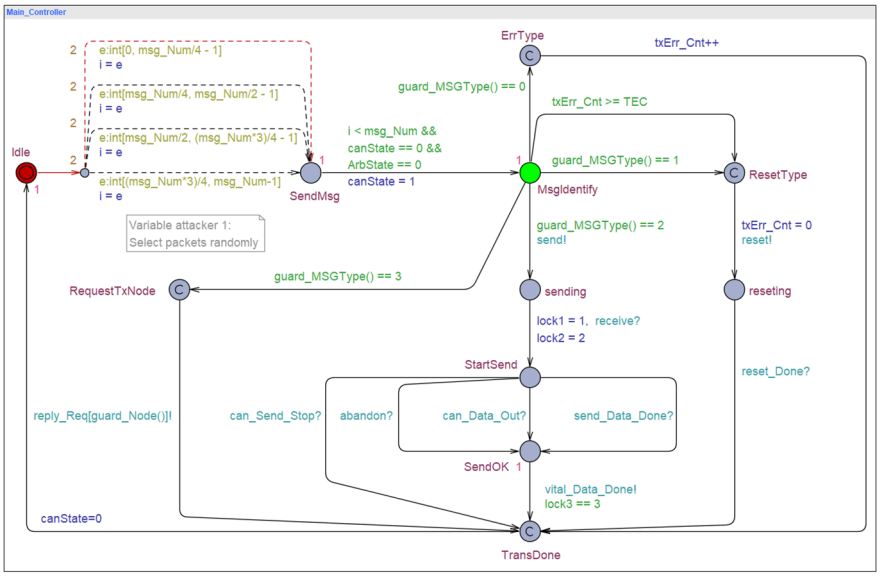

Figure 6. Timed automata (TA) of the master on the CAN bus: Main_Controller.

Next, the model takes the contents of the subsequent bits of the message and waits for the answerback. The TEC is incremented if the bus detects an error message. The bus will send a reset synchronization to initialize when txErr_Cnt increases to the given threshold. While the reset message is received, the communication with the template Node_Applicition is done by the channel synchronization reset! on the transition to ResetType. When the MSGType is 2, it starts sending and synchronizes with send!. Up until the application layer transfer is done, the template receives the signal synchronizes to move to the location SendOK to end this transition. If the MSGType is 3, the model continues to use the function guard_Node() to identify the message ID and returns the number of which node needs to be answered. Finally, a signal reply_Req [id]! is sent to the corresponding node to notify it to prepare the response frame.

3.2.2. The Application Layer of the CAN Bus

We simulated the transfer of behavior in the case of both data transmission and bus reset on the application layer of embedded communication software development. Figure 7 shows the TA of the CAN bus on the application layer. During the transmission, the frame interval of a multi-frame packet is represented by the clock t_DataSend_Interval, and the frame interval does not exceed t_Done_Time time units. Moreover, the location SendState has the invariant t_DataSend_Interval <= interval_UppLim.

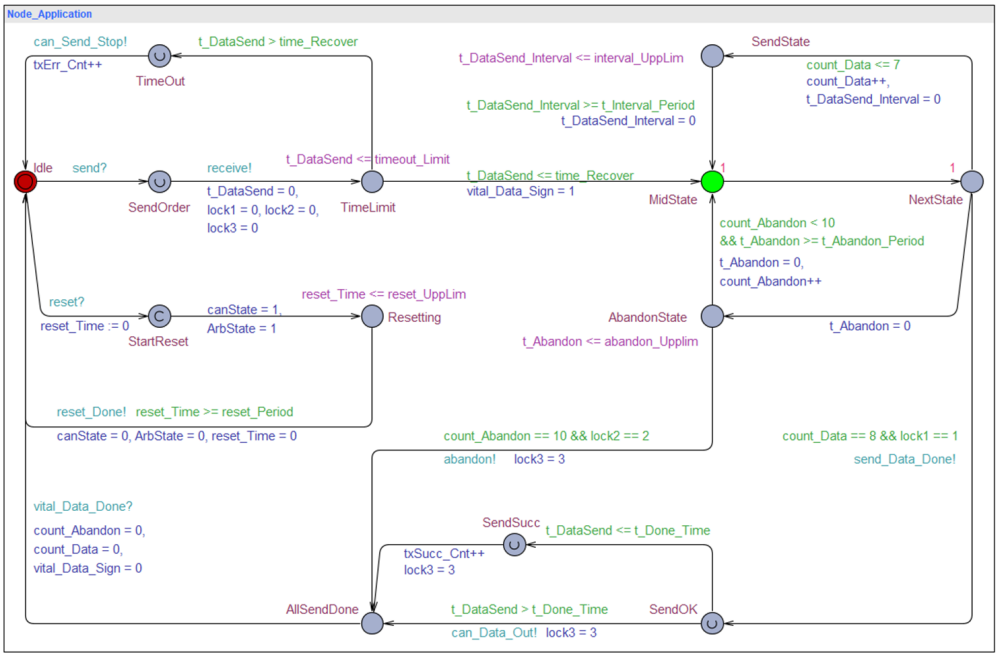

Figure 7. TA of the CAN bus application layer: Node_Application.

The number of packets sent should be no less than the number of frames. Here, eight data frames serve as an example to simulate the data sending process. When the data has been sent, a count of the transmission time is taken and compared with the maximum duration limit of transmission t_Done_Time. If the transmission time t_DataSend is shorter than t_Done_Time, it is judged to be a correct transmission. Meanwhile, if there is a packet loss while sending data, the packet loss count is incremented. Moreover, the clock reset_Time is initialized to 0 when the initial location Idle receives reset? synchronization. The following location is a committed location StartReset, which sets the flag canState to non-idle and indicates that the bus is unavailable at the time. The bus can continue to send messages when the minimum specified reset time reset_Period is reached.

3.2.3. Subordinate Transmit Node

This section elaborates on the response operation of the slave node on the CAN bus. When a node receives a command requiring a response from the master controller, it will prepare the data and complete the reply. The transmission process is shown in Figure 8. The node controller parses the message and prepares the required reply according to the demands of the received message.

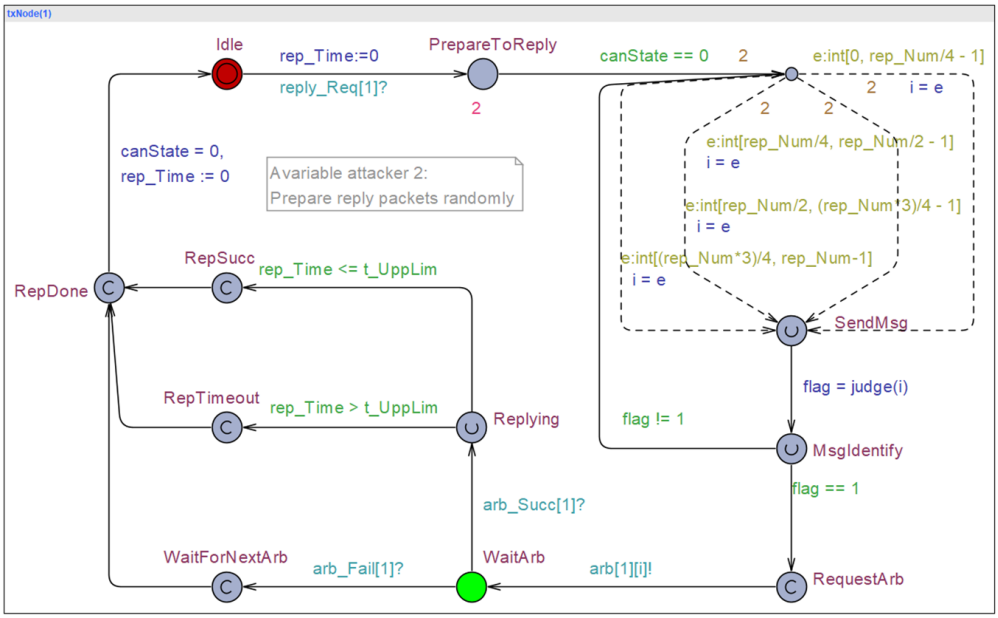

Figure 8. TA of the slave node in the CAN bus: txNode(i).

In order to simulate real transmission, we set different preparation times for different nodes in order to achieve the random timing of the transmission. When the bus is free in the arbitration phase, the node sends an arbitration request arb[i][e]! and then reaches the location WaitArb. During the arbitration, the nodes with lower priority are eliminated one after another. Thus, the model ensures no delay in data transmission for the high-priority nodes, as the beginning of the message is already being transmitted when the highest-priority message is arbitrated successfully. Finally, the reply time rep_Time is compared with the specified minimum reply time limit t_UppLim. The response is considered successful if it is completed within the specified time and reaches the location RepSucc.

3.2.4. Arbiter Model

The template Arbitration in Figure 9 shows the per-bit arbitration process that takes place before a message is transmitted. Once the signal "arb[i][e]?" arrives, the value is added to list [n], which represents node *i* requesting message *e* and waiting for arbitration. In addition, if there is no arbitration request from another node within a certain period, it is considered that there is no competing sender on the bus at this time, and the transition is taken from the location WaitOtherNode to the location NoConflicts. Here, the certain time is set to 50 time units in the timed automaton. This represents the time delay in determining whether another node is sending data to the bus at the same time, and this time delay is 0 in reality.

If other nodes request to use the bus simultaneously, it will enter the multi-message arbitration phase. Arbitration by bit is carried out at the data link layer, and an all-1 array signal [11] is set up to facilitate the calculation of the current priority. The corresponding bit of each message is multiplied by the all-1 array signal [11], and the product value signal [s] is compared with the bit of the arbitrating message. If they are equal, the message is ready for subsequent arbitration; if not, the message fails to arbitrate.

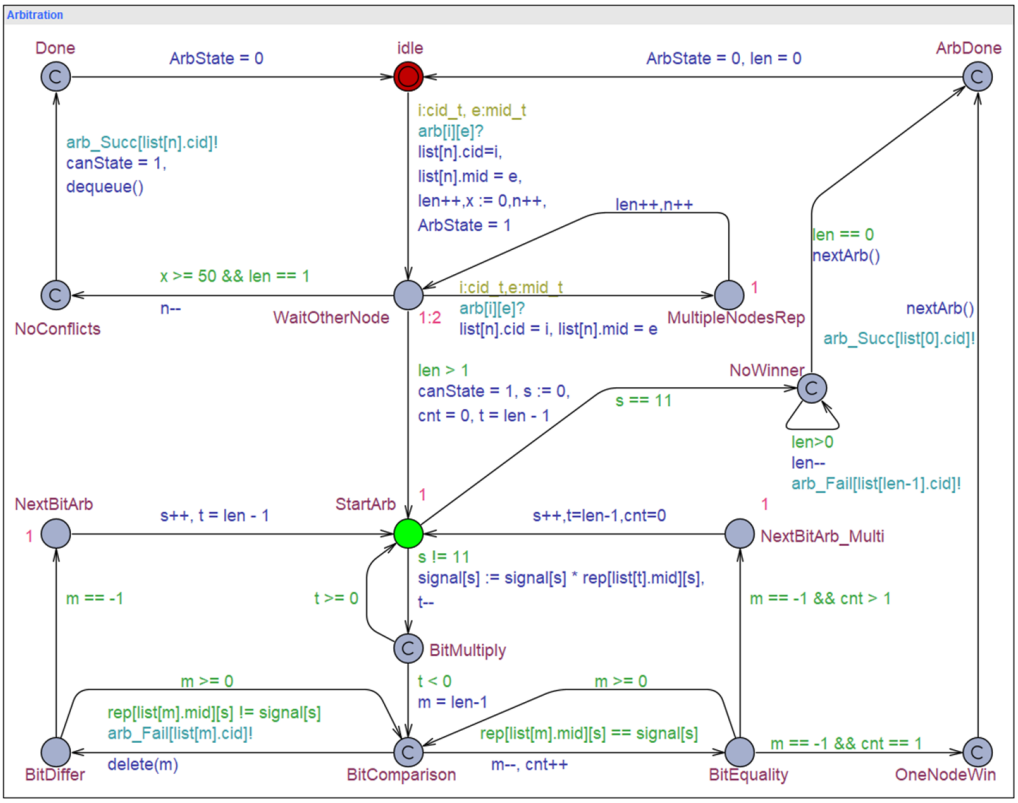

Figure 9. TA of the arbitrator in the data link layer of CAN bus: Arbitration.

4. Model Verification and Result Analysis

An obvious disadvantage of the CAN bus is the inability to accurately limit the worst-case response time for a given message [28], that is, the maximum time between message queuing and the arrival at the target processor. The purpose of this paper is to provide a reference standard for setting the timing constraints with an analysis of the satisfiability of the VACB model.

4.1. Verification of Basic Functions

In this section, the reachability properties, liveness properties, and absence of deadlock of the VACB model are checked by the UPPAAL CTL formulas. The verification formulas are described in Table 3. The verification results in UPPAAL are shown in Figure 10, where the model satisfies all the following properties in Table 3. The green in Figure 10 shows that the property can pass the validation of model checking, which indicates that the CAN bus model is well constructed and fully functional.

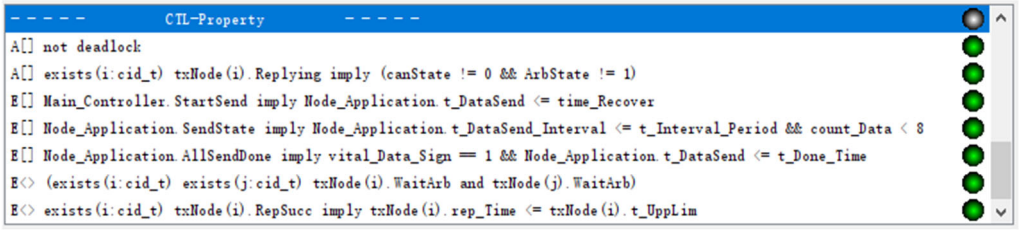

Figure 10. Verification results for the basic properties of the VACB model in UPPAAL.

Table 3. Property verification formula and its description.

No.	Verification Formula	Description
1	A[ ] not deadlock	No deadlock in the integrated model
2	A[] exists(i:cid_t) txNode(i).Replying imply (canState!= 0 && ArbState!= 1)	CAN bus is occupied and arbitration is over when any slave node is in response
3	E[] Main_Controller.StartSend imply Node_Application.t_DataSend <= time_Recover	Main controller message sent no more than time_Recover time units
4	E[] Node_Application.SendState imply Node_Application. t_DataSend_Interval <= t_Interval_Period && count_Data < 8	The number of consecutive messages sent by the main controller meets the demand limit
5	E[] Node_Application.AllSendDone imply vital_Data_Sign == 1 && Node_Application.t_DataSend <= t_Done_Time	The node can complete the sending of all messages within the specified time
6	E<> (exists(i:cid_t) exists(j:cid_t) txNode(i).WaitArb and txNode(j).WaitArb)	Correctness of arbitration function in multi-node transfer conflict
7	E<> exists(i:cid_t) txNode(i).RepSucc imply txNode(i).rep_Time <= txNode(i).t_UppLim	Subordinate node can complete the response within the specified time

4.2. Anti-Attack Performance Analysis of the VACB Model

In this section, we analyze the anti-attack performance of the CAN bus system. Table 4 shows the simulation parameters in UPPAAL SMC. Based on the simulation environment, two indexes—transmission accuracy and the successful response rate—are measured to quantify the timing security of the VACB model.

Table 4. Simulation parameters of the VACB model.

Type	Variable	Description	Value
Timing Constraints	wait_Period	Time the other node waits to arbitration	50 μs
	time_Recover	Minimum time of the CAN bus recovery	2000 μs
	reset_Period	Reset time of the CAN bus	5000 μs
	t_Interval_Period	Interval between composite frames	50 μs
	interval_UppLim	Maximum interval time between frames	80 μs
	t_Done_Time	Message's maximum sending time	2000 μs
	t_UppLim	Upper limit of the reply time from the slave node	3000 μs
Number	Slave Node	Number of the slave node on the CAN bus	4, 8, 16
	msg_Num, rep_Num	Number of CAN messages	8, 16, 32
	txErr_Cnt	Threshold of TEC	90
Rate on Location	PrepareToReply	Exponential rate to the location PrepareToReply	(1 + id)/N*N, 1

Transmission accuracy: The developers have designed a mechanism for instruction receipt confirmation in the software application layer. The node returns a successful receiving identifier (ACK) to the master controller after receiving a bus instruction, so the upper procedure ensures the validity of packet transmission. The period of the master controller waiting for the return of ACK is set to t_Done_Time. If all the data is sent and received within this period, it is considered successfully transmitted; otherwise, the error count is accumulated.

The successful response rate: When the slave node receives the CAN instruction requiring the response, the response packet must be sent to the master controller within a

specified timing constraint. This timing constraint is set to t_UppLim. If the slave node can respond within this value, it is considered a successful response; otherwise, it is considered a response timeout, and the slave node will wait for the next arbitration.

Table 5 shows the simulation formula used to obtain the index's probabilistic value and its description. The successful response rate of Node 1 arriving at the corresponding location (denoted as RepSucc) is described by Formula (3) in Table 5. Some results in UPPAAL SMC are shown in Figure 11.

Table 5. UPPAAL SMC simulation formula.

No.	Simulation Formula	Description
1	simulate 1 [<=2000] {Node_Application.reset_Time, Node_Application.t_DataSend}	Simulation of the CAN bus application layer bus reset recovery time and transmission time
2	Pr[<=4000] (<>Node_Application.SendSucc)	Probability of reaching the successful sending location SendSucc
3	Pr[<=8000] (<> txNode(1).RepSucc)	Probability of a slave node reaching the successful response location RepSucc

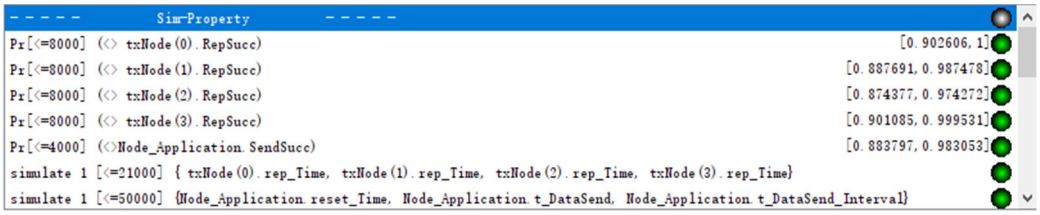

Figure 11. Simulation results in UPPAAL SMC.

There are three scenarios to test the probability: normal (using normal messages without optimization), under attack (under FMA-RNN without optimization), and optimized (under FMA-RNN after optimization). First, we run normal messages in the VACB model to test the current transmission accuracy and successful response rate of the CAN bus. Second, malicious network packets generated by the FMA-RNN are adopted to test the accuracy under the timing constraints. Finally, we propose some improvements to promote the CAN bus parameter for the degradation caused by the FMA-RNN.

- *Normal*

Under the normal circumstances and four nodes attached to the CAN bus, the average probability of successful transmission using multiple sets of normal CAN messages is [0.893009, 0.992099], and the average probability of successful response is [0.9110455, 0.985242]. Table 6 shows the detail of the probability.

- *Under Attack*

Subsequently, the transmission accuracy is tested with different t_Done_Time values under the FMA-RNN where the bus is attached to a different number of nodes (4, 8, and 16), as shown in Figure 12. The results illustrate that the optimal value of t_Done_Time is 2 ms $\pm$ 0.1 (a time unit is 1 μs).

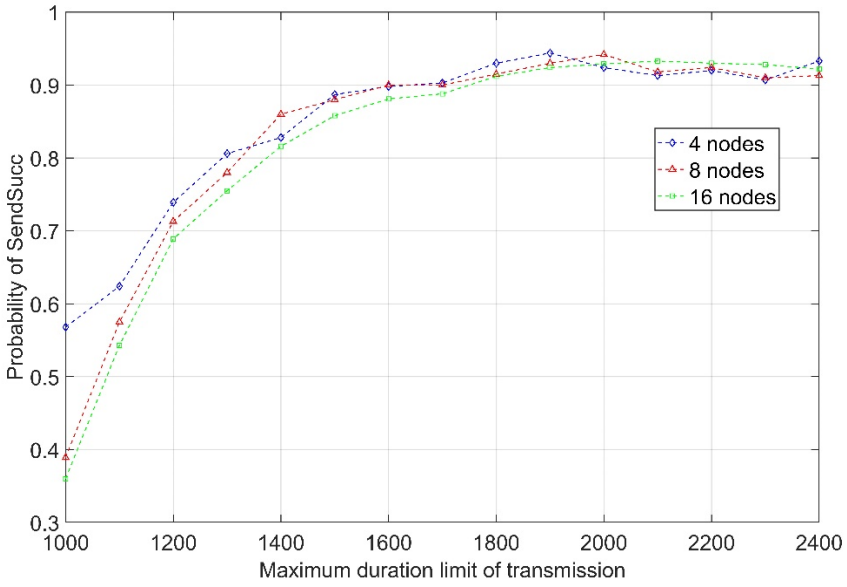

Figure 12. Transmission accuracy for various values of t_Done_Time in various numbers of nodes.

Taking four slave nodes attached to the CAN bus, the upper limit of the successful transmission probability under the FMA-RNN within the timing constraints is 97.4%, which is a decrease of 1.8% when compared to the previous normal rate. Meanwhile, as shown in Table 6, the node's successful response rate decreased by 4.3% on average when the FMA-RNN was added.

Table 6. The transmission accuracy and the successful response rate in the three cases.

		Normal CAN Packets	**Under FMA-RNN**	**Optimized (Under FMA-RNN)**
Transmission Accuracy	Node_Application. SendSucc	[0.893009, 0.992099]	[0.874377, 0.974272]	[0.901085, 0.999531]
Successful Response Rate	txNode(0).RepSucc	[0.902606, 1]	[0.87865, 0.978536]	[0.896085, 1]
	txNode(1).RepSucc	[0.897759, 0.996418]	[0.841731, 0.94138]	[0.894061, 0.993924]
	txNode(2).RepSucc	[0.874377, 0.974272]	[0.831933, 0.931764]	[0.870781, 0.970278]
	txNode(3).RepSucc	[0.870781, 0.970278]	[0.816972, 0.916916]	[0.866689, 0.966276]

- *Optimized*

The simulation result of the transmission accuracy under the FMA-RNN is shown in Figure 13a. In order to improve the transmission accuracy of the CAN bus, we made the following optimization. Each instruction was sent three times to improve the transmission accuracy. This is reflected by the fact that the relevant instructions in the dataset used for testing the probabilities are backed up in three copies. If one instruction was received and the ACK was returned, the transmission was considered successful. By increasing the number of critical instructions transmitted, the successful transmission probability was patently increased. The optimized result is presented in Figure 13b, which indicates that the upper limit of the probability reached 99.9%, which is an increase of 2.5% when compared to the prior results.

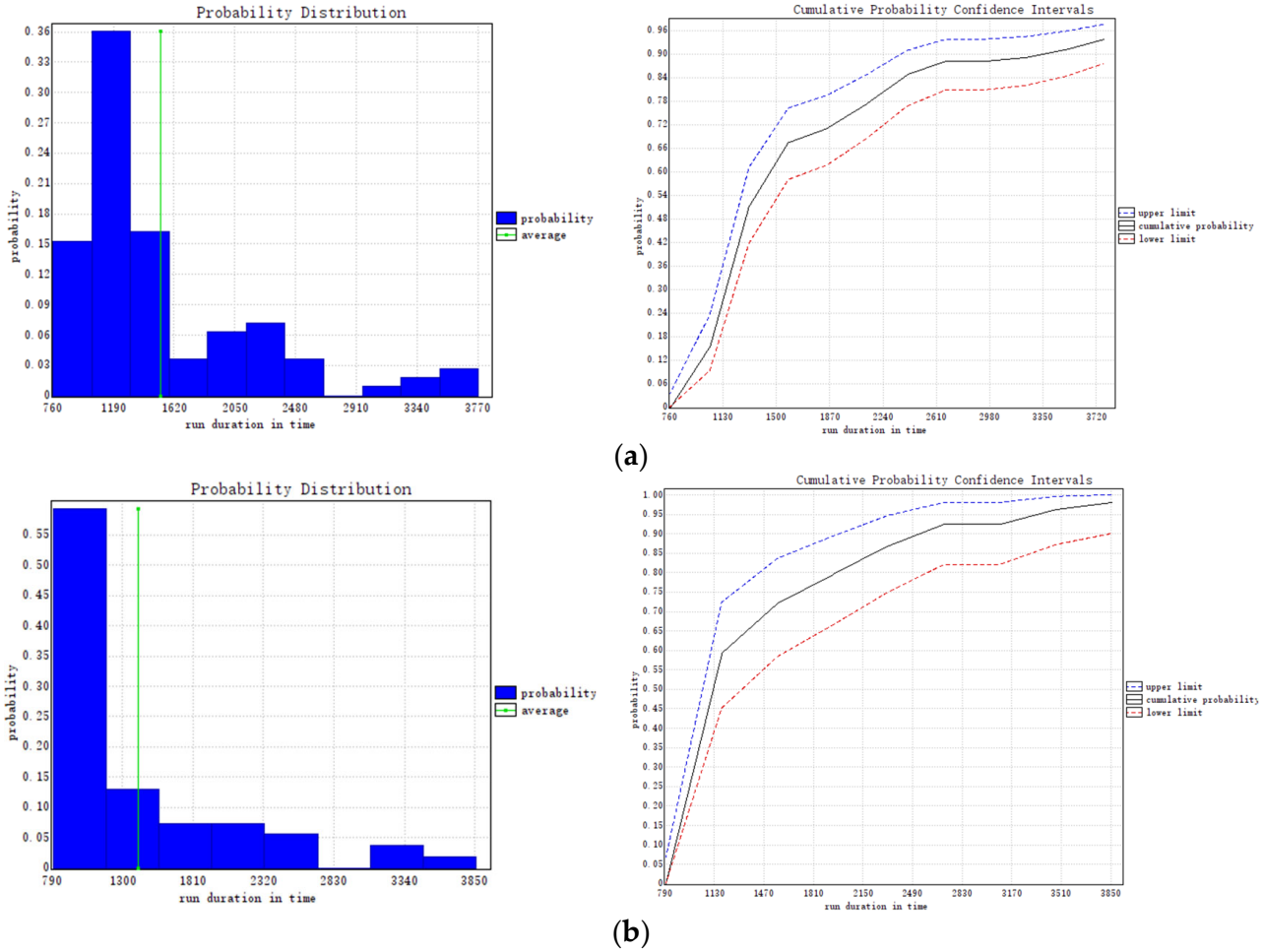

Figure 13. Probability distribution and cumulative probability confidence intervals of transmission accuracy on the CAN bus. (**a**) The probability of successful transmission under the FMA-RNN without optimization. (**b**) The probability of successful transmission under the FMA-RNN after optimization.

For the other index successful response rate, two changes were made on the application layer. First, a message filter was set before the node's receive buffer to filter out erroneous attack messages through programming. Second, the response data was prepared in advance in order to respond faster. In addition, we assigned the highest-priority interruption to the response operation to reduce interruptions. This was reflected in the TA model by increasing the rate of the location PrepareToReply from (1 + id)/N*N to 1. Meanwhile, both SendMsg and MsgIdentify were set to an urgent location to minimize time consumption.

To quantify the effect of the FMA-RNN and the optimization, taking Slave Node 1 as an example, the probability distributions of the successful response rate are compared in Figure 14. The details of the probability distribution and cumulative distribution before and after optimization under the FMA-RNN are shown in Figure 15. The successful response rate increased by 5.1%, which is close to the conventional case, indicating that optimization plays a significant role.

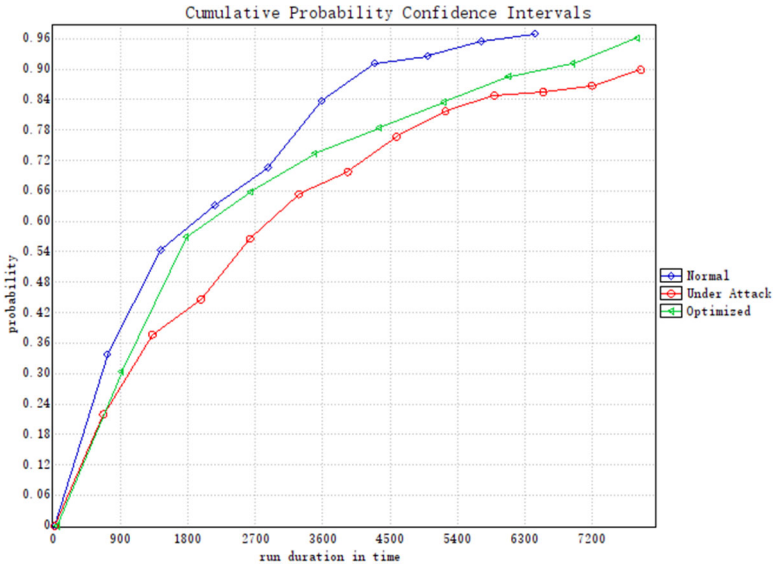

Figure 14. Cumulative probability of the successful response rate in the three scenarios.

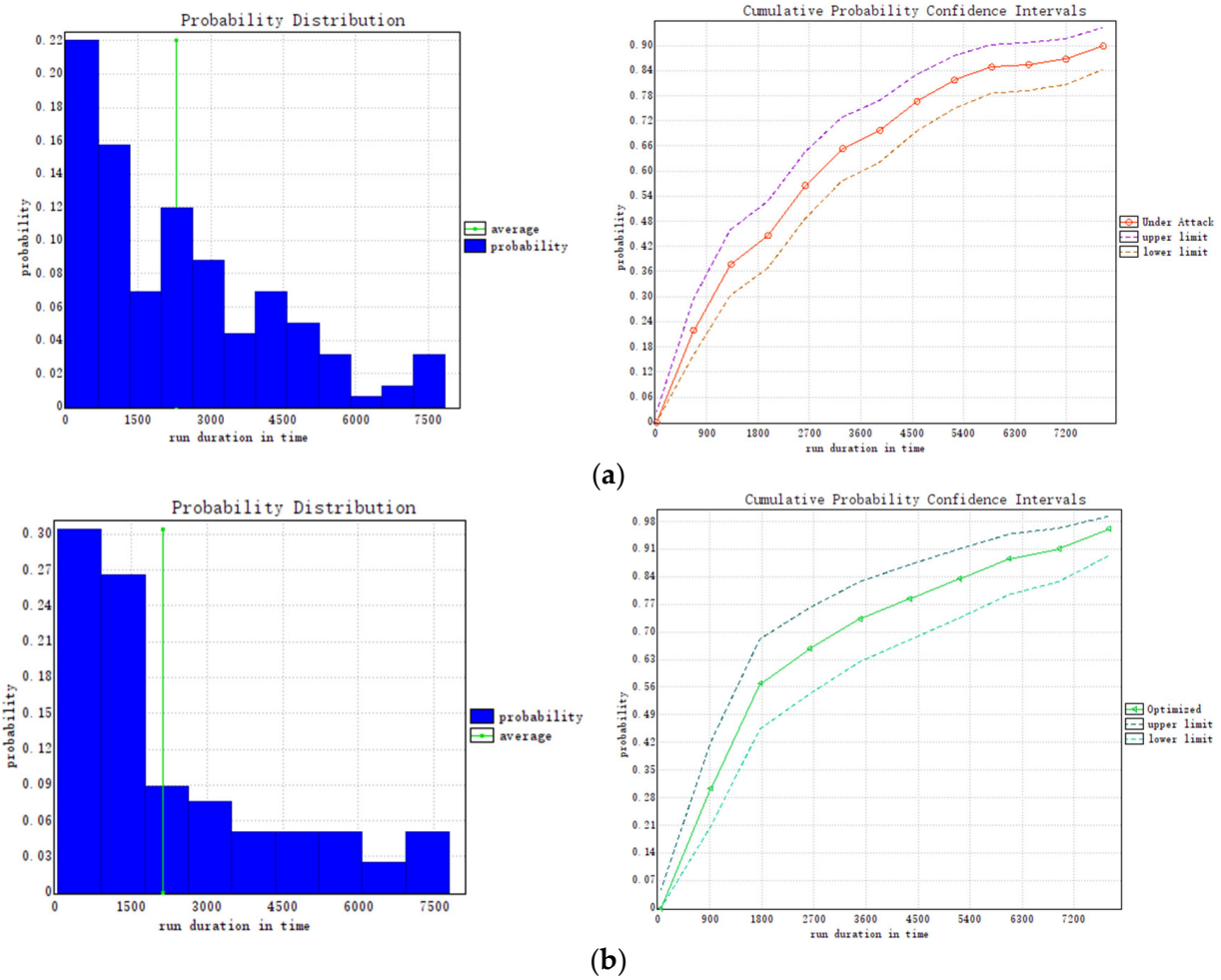

Figure 15. Comparison of results before and after optimization under the FMA-RNN. (**a**) This figure shows the successful response rate under the FMA-RNN without optimization. (**b**) This figure shows the successful response rate under the FMA-RNN after optimization.

5. Conclusions

In this paper, the VACB model is proposed to quantify the anti-attack capability of the CAN bus. The transmission accuracy and response efficiency are reduced to different degrees as the CAN bus undertakes continuous attacks generated by the FMA-RNN. The abnormal cases of the CAN bus under different scenarios are analyzed in the VACB model to reduce the potential risk of software failures. Furthermore, developers can design exception handling and optimize the embedded software based on the error warning and simulation results demonstrated by our VACB model. According to the verification results in UPPAAL SMC, the transmission accuracy and successful response rate are increased by 2.5% and 5.1%, respectively. Therefore, our model has a wide range of application prospects in analyzing the cybersecurity performance of communication buses.

Author Contributions: Conceptualization, Y.W. and Q.Z.; methodology, Y.W. and Y.Z.; validation, Y.W. and X.Z.; formal analysis, Y.W.; investigation, Y.W.; resources, Q.Z.; writing—original draft preparation, Y.W.; writing—review and editing, Y.Z., X.Z. and J.D.; visualization, Y.W. and J.D.; supervision, Q.Z.; project administration, Q.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Civil Aerospace Technology Advance Research Project, grant number: B0204.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: [https://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset].

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

Abbreviations

CAN	Controller Area Network
RNN	Recurrent Neural Network
VACB	The variable attacker with the CAN bus system
FMA-RNN	A formal modeling attacker using RNN
TA	Timed Automata
ATG	Attack Traffic Generation
GAN	Generative Adversarial Network
CPS	Cyber-Physical Systems
TEC	Transmit Error Counter
ACK	Acknowledge character
DoS	Denial of Service
ID	Identifier
DNNs	Deep Neural Network
CTL	Computation Tree Logic

Symbols

x_t	Input at moment t
h_t	Memory information at moment t
o_t	Output at moment t
K1	Weight matrix from the input layer to the hidden layer of encoder
R1	Weight matrix from the hidden layer to the output layer of encoder
y	Output matrix of encoder
m_i	Midbeam matrix between encoder and decoder
K2	Weight matrix from the input layer to the hidden layer of decoder
R2	Weight matrix from the hidden layer to the output layer of decoder
v_t	True label value of the input at moment t
p_t	Model predicted value

canState	CAN bus state
ArbState	Arbitration state
txErr_Cnt	Transmission error count on the CAN bus
MSGType	CAN network message type
t_DataSend_Interval	Clock of data frame sending
t_Interval_Period	Interval between composite frames
t_Done_Time	Message's maximum sending time
interval_UppLim	Maximum interval time between frames
t_DataSend	Clock of data transmission
time_Recover	Minimum time of the CAN bus recovery
reset_Time	Clock of the CAN bus resetting
reset_Period	Reset time of the CAN bus
rep_Time	Clock of the txNode(i) replying
t_UppLim	Upper limit of the reply time from the slave node
txNode(i)	Slave node on the CAN bus
msg[i][j]	CAN network packets
rep[i][j]	Reply Message Array
signal[i]	Signal flag bit on the CAN bus
arb[i][e]	Arbitration message e released by node i
list[n]	List of arbitration node

References

1. Thomas, J.; Davis, A.; Samuel, M.P. Integration-In-Totality: The 7th System Safety Principle Based on Systems Thinking in Aerospace Safety. *Aerospace* **2020**, *7*, 149. [CrossRef]
2. Meng, Z.; Tang, T.; Wei, G.; Yuan, L. Analysis of ATO System Operation Scenarios Based on UPPAAL and the Operational Design Domain. *Electronics* **2021**, *10*, 503. [CrossRef]
3. Clarke, E.M.; Wing, J.M. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* **1996**, *28*, 626–643. [CrossRef]
4. Krakora, J.; Hanzalek, Z. Timed Automata Approach to CAN Verification. *IFAC Proc.* **2004**, *37*, 147–152. [CrossRef]
5. Wang, R.; Guan, Y.; Li, X.; Zhang, R. Formal verification of CAN bus in cyber physical system. In Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 11–14 December 2020; pp. 249–255.
6. Rodriguez-Navas, G.; Proenza, J.; Hansson, H. An UPPAAL model for formal verification of master/slave clock synchronization over the controller area network. In Proceedings of the 6th IEEE International Workshop on Factory Communication Systems, Torino, Italy, 28–30 June 2006.
7. Pan, C.; Guo, J.; Zhu, L.; Shi, J.; Zhu, H.; Zhou, X. Modeling and Verification of CAN Bus with Application Layer Using UPPAAL. *Electron. Notes Theor. Comput. Sci.* **2014**, *309*, 31–49. [CrossRef]
8. Meng, Y.; Li, X.J.; Guan, Y.; Wang, R.; Zhang, J. Modeling and verification for robot joint bus communication system. *Ruan Jian Xue Bao/J. Softw.* **2018**, *29*, 1699–1715. (In Chinese)
9. Kim, J.H.; Jo, H.J.; Lee, I. Model Checking Resiliency and Sustainability of In-Vehicle Network for Real-Time Authenticity. *Appl. Sci.* **2021**, *11*, 1068. [CrossRef]
10. Taylor, A.; Leblanc, S.; Japkowicz, N. Anomaly detection in automobile control network data with long short-term memory networks. In Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Montreal, QC, Canada, 17–19 October 2016; pp. 130–139.
11. Tanksale, V. Intrusion detection for controller area network using support vector machines. In Proceedings of the 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), Monterey, CA, USA, 4–7 November 2019; pp. 121–126.
12. Song, H.M.; Woo, J.; Kim, H.K. In-Vehicle Network Intrusion Detection Using Deep Convolutional Neural Network. *Veh. Commun.* **2020**, *21*, 100198. [CrossRef]
13. Seo, E.; Song, H.M.; Kim, H.K. GIDS: GAN based intrusion detection system for in-vehicle network. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST), Belfast, Ireland, 28–30 August 2018; pp. 1–6.
14. Koscher, K.; Czeskis, A.; Roesner, F.; Patel, S.; Kohno, T.; Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H. Experimental security analysis of a modern automobile. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 447–462.
15. Huang, T.; Zhou, J.; Bytes, A. ATG: AN attack traffic generation tool for security testing of in-vehicle CAN bus. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–6.
16. Hayward, J.; Tomlinson, A.; Bryans, J. Adding cyberattacks to an industry-leading can simulator. In Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, 22–26 July 2019; pp. 9–16.

17. Ring, M.; Dallmann, A.; Landes, D.; Hotho, A. IP2Vec: Learning similarities between IP addresses. In Proceedings of the 2017 IEEE International Conference on Data Mining Workshops (ICDMW), New Orleans, LA, USA, 18–21 November 2017; pp. 657–666.
18. Ring, M.; Schlör, D.; Landes, D.; Hotho, A. Flow-Based Network Traffic Generation Using Generative Adversarial Networks. *Comput. Secur.* **2019**, *82*, 156–172. [CrossRef]
19. Li, Y.; Liu, T.; Jiang, D.; Meng, T. Transfer-learning-based network traffic automatic generation framework. In Proceedings of the 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 9–11 April 2021; pp. 851–854.
20. Cheng, Q.; Zhou, S.; Shen, Y.; Kong, D.; Wu, C. Packet-Level Adversarial Network Traffic Crafting Using Sequence Generative Adversarial Networks. *arXiv* **2021**, arXiv:2103.04794v1.
21. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning Representations by Back-Propagating Errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
22. Hyötyniemi, H. Turing Machines Are Recurrent Neural Networks. *Proc. STEP* **1996**, *96*, 13–24.
23. Manès, V.J.M.; Han, H.; Han, C.; Cha, S.K.; Egele, M.; Schwartz, E.J.; Woo, M. The Art, Science, and Engineering of Fuzzing: A Survey. *IEEE Trans. Softw. Eng.* **2019**, *47*, 2312–2331. [CrossRef]
24. Behrmann, G.; David, A.; Larsen, K.G. *A Tutorial on Uppaal 4.0*; Aalborg University, Department of Computer Science: Aalborg, Denmark, 2006.
25. Alur, R.; Dill, D.L. A Theory of Timed Automata. *Theor. Comput. Sci.* **1994**, *126*, 183–235. [CrossRef]
26. Kang, E.-Y.; Mu, D.; Huang, L. Probabilistic verification of timing constraints in automotive systems using UPPAAL-SMC. In Proceedings of the International Conference on Integrated Formal Methods, Maynooth, Ireland, 3–4 September 2018; pp. 236–254.
27. Lu, Y.; Sun, W.; Bai, G.; Sun, M. DeepAuto: A first step towards formal verification of deep learning systems. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, Pittsburgh, PA, USA, 1–10 July 2021; Volume 2021, pp. 172–176.
28. Tindell, K.; Burns, A.; Wellings, A.J. Calculating Controller Area Network (CAN) Message Response Times. *Control Eng. Pract.* **1995**, *3*, 1163–1169. [CrossRef]