Implementation of the CAN-FD Protocol in the Fuzzing Tool beSTORM

Ryosuke Nishimura*, Ryo Kurachi†, Kazumasa Ito‡, Takashi Miyasaka‡, Masaki Yamamoto†, Miwako Mishima*

* Graduate School of Engineering, Gifu University, Japan
u3126028@edu.gifu-u.ac.jp
miwako@gifu-u.ac.jp

† Graduate School of Information Science, Naogya University, Japan
kurachi@nces.is.nagoya-u.ac.jp
myamamoto@nagoya-u.jp

‡ Aiuto, Ltd., Japan
k.ito@t-aiuto.jp
t.miyasaka@t-aiuto.jp

***Abstract*—With the growth of ECUs that are mounted in automobiles, the transmission capacity of Controller Area Network (CAN), which is currently used by most on-vehicle networks, is becoming insufficient, and therefore CAN With Flexible Data Rate (CAN-FD), presented by Bosch GmbH, is viewed as a next-generation standard. Recently, the number of attacks on ECUs connected to CAN has been increasing, and from the viewpoint of CAN security there has been much discussion of safety. However, with regard to CAN-FD, which is an expansion of CAN, although a discussion of safety is needed, that has hardly happened. This paper reports on an implementation that uses the generic fuzzing tool beSTORM to investigate the vulnerability of the CAN-FD protocol. We also aim at practical application by measuring the transmission time for beSTORM test data and by estimating the time required for CAN-FD fuzzing tests.**

I. Introduction

Today there are dozens of electronic control units (ECUs) mounted for every automobile in existence [1], and they implement a variety of functions by exchanging the data stored in the ECUs by using on-vehicle control network protocols such as those of the Controller Area Network (CAN) [2], Local Interconnect Network (LIN) [3], and FlexRay [4]. In particular, CAN has grown to the point that one or more CAN busses are used in every vehicle sold today, and it can be called a de facto standard.

Recently, there have been many reports of attacks where unplanned operation occurred as the result of spoofing messages inserted onto the CAN bus by connecting an improper device [5], [6]. Most of the attacks were implemented by analyzing the contents of messages to be transmitted on the CAN bus and then inserting forged CAN messages on the CAN bus with improper frequency or improper timing. In response to this, there have been attempts [7] in the automobile industry to attach a message authentication code (MAC) to the CAN payload in order to guarantee message completeness. However, the CAN message payload is limited to a maximum of 8 bytes, so it is difficult to attach a sufficient MAC to the payload. Also, in the CAN specification the maximum transmission rate is 1 Mbps, which is becoming an insufficient transmission capacity, and for these reasons CAN With Flexible Data Rate (CAN FD) [8] has been presented as a protocol that expands the transmission capacity of CAN.

CAN-FD is a protocol that was presented in 2012 by Bosch GmbH, and is now being drawn up as an international standard by ISO11898 [8]. Two characteristics of the CAN-FD protocol are that it continually traces the mechanisms for arbitrating frame systems and transmission rights that are compatible with CAN, and that the payload length has been expanded to 64 bytes. It is also a protocol that can realize higher transmission efficiency than that of CAN because some of the fields included in the expanded payload have their transmission rates raised to a maximum of 8 Mbps.

A. Background of the Current Research

There has already been research related to methods of evaluating the vulnerability of ECUs, and Bayer et al. [9] present an evaluation method for the vulnerability of on-vehicle control systems, with evaluation levels. The evaluation levels presented in [9] show the evaluation level of the ECU based on various evaluation methods such as fuzzing tests and penetration tests.

Matsumoto et al. [10] propose a fuzzing test method that uses CAN, and they experimented on an actual machine from the three viewpoints of the transmission of unused IDs, transmission with high frequency, and transmission of erroneously generated messages. They indicated in particular that testing for erroneously generated messages is important for CAN.

Bayer et al. [11] designed a fuzzing test tool for CAN and showed that it is possible to execute a fuzzing test on “ UDS on CAN ” in realistic time.

However, as far as we know, although existing research mentions fuzzing tests of CAN, there is no mention of fuzzing tests of CAN-FD. Therefore, with this paper we have designed and evaluated a fuzzing test tool that supports CAN-FD. The test tool that we have designed is implemented by integrating the existing fuzzing tool beSTORM with a CAN-FD interface that we developed. This paper evaluates the execution time for a fuzzing test in order to show the usability of the fuzzing tool that has been designed.

B. The organization of this paper

The following is the structure of the remainder of this paper. First, section II gives an overview of the CAN and CAN-FD protocols that will be the object of the fuzzing. Then, section IV explains the characteristics of the existing fuzzing tool beSTORM and gives an overview of the fuzzing tool that we have developed. Section III explains the fuzzing test method using the developed fuzzing tool, section V measures and discusses the processing time required for the fuzzing test, and finally section VI summarizes the paper.

II. CAN and CAN-FD

To address the fact that the CAN transmission capacity cannot handle message sizes that are expected to grow in the future, various protocols such as FlexRay, CAN+ [12], and Scalable CAN [13] have been presented. Among these, CAN-FD is seen as the next-generation standard to replace CAN. This section gives overviews of the CAN and CAN-FD protocols, and explains their differences.

A. Controller Area Network(CAN)

CAN, which currently is used by many on-vehicle networks, is a bus-type serial communication protocol developed by Bosch GmbH. It has been made a standard by ISO11898 ISO:CAN, and is used not just by automobiles but also for example by airplanes and manufacturing robots.

The CAN communication path comprises two wires, and has a level that must always be either the dominant level or the recessive level based on the potential difference. Even if noise is mixed in there is almost no change in the potential difference between the wires, so it has excellent anti-noise characteristics.

In CAN, all of the nodes connected to the bus can transmit messages, and broadcast communication is performed. If two or more nodes simultaneously start transmission, arbitration is performed from the first bit of the field that is called the ID of the CAN message, and the node that had continuously transmitted the dominant level for the longest time obtains transmission rights, and nodes that have not obtained transmission rights transition to reception operation from the next bit.

Next we explain the data frame (Figure 1) used for test data when performing a fuzzing test.

A data frame is a frame that is used for transmitting and receiving messages. There is a standard format and an expanded format in which the ID field is expanded. This section explains the standard format.

The data frame is mainly composed of (1) the ID field, (2) a control field, (3) a data field, and (4) a CRC sequence. Each of their roles is as follows:

- **(1) ID Field:** Contains the CAN-ID. As already mentioned, this is used for arbitration when there is a frame collision.

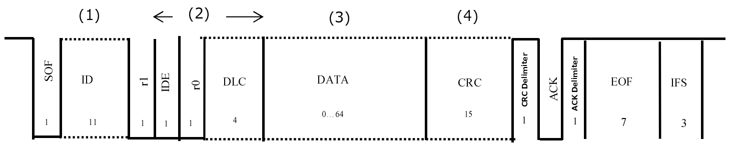

Fig. 1. Traditional CAN Data Frame.

- **(2) Control Field:** Contains reserved bits r1 and r0, each of which is fixed dominant, and a DLC (data length code) that specifies the length of the data field.
- **(3) Data Field:** Stores the data being transmitted. According to the DLC (data length code) in the control field, it can specify a message having 0 to 8 bytes, in units of 1 byte.
- **(4) CRC Sequence:** This stores a 15-bit CRC (cyclic redundancy check) symbol for the bit sequence from SOF through the data field.

B. Controller Area Network(CAN) with Flexible Data-Rate

1) Characteristics of CAN-FD: CAN-FD is a serial communication protocol based on CAN, and it has the following two characteristics.

1) In addition to the 0 to 8 bytes of CAN, it can expand the payload length to 12, 16, 20, 24, 32, 48, or 64 bytes.
2) In on-vehicle CAN the communication rate is limited to 500 kbps but, from the second half of the control field through the CRC sequence, modulation is performed by switching the division ratio, so that it can be accelerated to 8 Mbps.

2) Differences from CAN: In CAN-FD, the following three bits are added to the CAN control field (6 bits), expanding the control field to 9 bits (Figure 2).

- **Extended Data Length (EDL):** Tells whether the frame format is CAN-FD or CAN.
- **Bit Rate Switch (BRS):** Tells whether modulation is to be performed.
- **Error State Indicator (ESI):** Tells the error status of

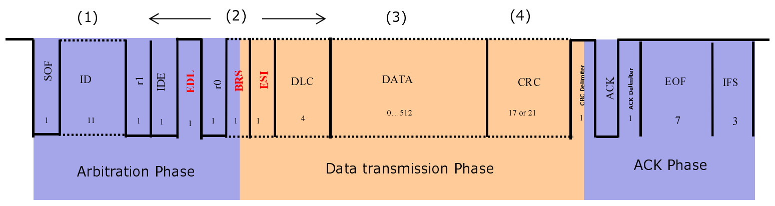

Fig. 2. CAN-FD Data Frame in non-ISO version.

In the CAN frame, if RTR is recessive and the data field is removed then it is a remote frame. However, CAN-FD does not have a remote frame, and the bit that corresponds to RTR in the CAN frame becomes a reserved bit r1 that is fixed dominant in CAN-FD.

3) CRC Sequence: In the CAN frame the CRC is 15 bits, but in CAN-FD, in order to expand the payload, the CRC sequence has been expanded to match it. If the payload is at most 16 bytes then a 17-bit CRC sequence is attached, or if the payload is more than 16 bytes then the CRC sequence is 21 bits.

III. CAN and CAN-FD Protocols for beSTORM

A. beSTORM

beSTORM [14] is a fuzzing tool that runs on Windows, developed by Beyond Security, Inc., and it is marketed in Japan by AIUTO Co., Ltd. beSTORM has been introduced by many organizations and companies in the world.

beSTORM supports internet protocols. It also supports CAN, which is not an internet protocol, and to perform a fuzzing test it is necessary to separately prepare an interface. Therefore, for this research we developed an interface to transmit test data from beSTORM to the CAN/CAN-FD bus.

Figure 3 shows the screen when a fuzzing test is being executed by beSTORM. In Figure 3, (1) can show the start time and elapsed time of the test, and the number of executed test cases. (2) is a chart that shows the number of test cases executed during each second and their average. When completed, the number of test cases executed and the time required for test completion are shown, and the test results are reported.

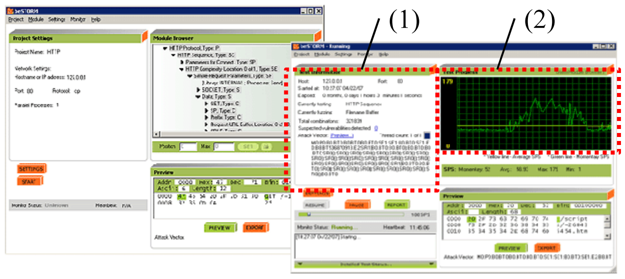

Fig. 3. Overview of beSTORM.

The following are the characteristics of beSTORM:

- All fuzzing data that includes the possibility of being a vulnerability for the protocol being tested is generated, and the tests are performed. A generation method for efficient combination can be selected, to realize high-speed testing.
- It is possible to define unique protocols (called modules) with XML, so there is excellent expandability.
- It is possible to call one ’s own DLL (dynamic link library).
- If an attack succeeds, a detailed report is recorded. This report makes it easy to reproduce the attack.

B. Development of the CAN-FD Interface

With beSTORM it is necessary to define a protocol module in order to perform fuzzing. It is assumed that the protocol module will be described in the XML format. The necessary fuzzing data is defined within the module, and it is possible to transmit the fuzzing data created by beSTORM to the target being evaluated by calling functions within a DLL. Therefore, in this research, we defined a protocol module for the CAN/CAN-FD protocol, and within it we created a configuration for performing CAN/CAN-FD communication through an external interface (hereafter, the CAN-FD interface) by calling functions within a DLL.

The following was the method for designing the CAN-FD interface:

1) In order to minimize the development scale, we used serial communication for the interface between beSTORM running on a PC and the CAN-FD interface.
2) There exist CAN-FD communication ports on the market, but we used a CAN-FD controller IP mounted on an FPGA board in order to introduce intentional forgeries in the CAN-FD frames into the fuzzing.
3) The object of the development was CAN-FD, but we designed it to also be able to support the CAN protocol.

Based on these assumptions, we used the following devices. Note that from here on this paper may only mention CAN-FD, but the CAN protocol is also supported.

- **DE0-NANO:** The FPGA board manufactured by Altera Corporation. We mounted TOPPSER/ATK2 [15], which is a real-time OS for an on-vehicle control system.
- **USB serial cable to connect beSTORM (PC) and DE0-NANO:** We developed the following two items in order to make beSTORM support the CAN-FD protocol.
- **CAN/CAN-FD Protocol Module:** The following are the contents of the CAN/CAN-FD protocol module. More details about ”(b) Transmitted data definition” are found in section 3.3.
  - **(a):** Procedure for opening the COM port of the PC.
  - **(b):** Transmitted data definition.
  - **(c):** Procedure for closing the COM part of the PC.
- **The Interface Between beSTORM and the CAN-FD Bus:** In order to transmit the data generated by beSTORM to the CAN-FD bus we created a DLL for communicating between beSTORM and DE0-NANO. By calling this DLL in the CAN/CAN-FD protocol module that is “ Development Item 1 ” in Figure 4 it is possible to transmit the test data generated by beSTORM to DE0 NANO via the USB serial cable. In DE0-NANO the received data is converted into CAN-FD packets and transmitted on the CAN-FD bus (“ Development Items 2 and 3 ” in Figure 4).

C. CAN/CAN-FD Protocol Module

Figure 5 is a portion of the CAN-FD protocol module, showing an example of the description of a data definition. The defined content is shown as follows.

1) Call the function BSISend inside bestorm-de0-serial.dll in order to transmit to DE0-NANO (lines 1-3).

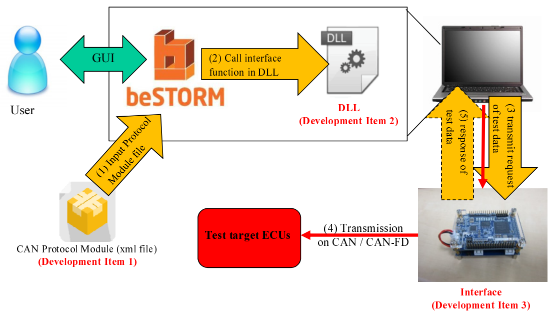
User
GUI
beSTORM
(2) Call interface function in DLL
DLL
DLL (Development Item 2)
(3 transmit request of test data
(5) response of test data
(1) Input Protocol Module file
CAN Protocol Module (xml file) (Development Item 1)
Test target ECUs
(4) Transmission on CAN / CAN-FD
Interface (Development Item 3)

Fig. 4. Overview of our developed test tool.

```
1: <SP Name="CAN Send"
2:    Library="bestorm-de0-serial.dll"
3:    Procedure="BSISend">
4:    <S Name="Identifier">
5:        <C Name="ID1" Value="01" />
6:        <C Name="ID2" Value="02" />
7:    </S>
8:    <S Name="Config">
9:       <EV Name="CAN-FD ID Format"
10:        Value="06"
11:        Description="CAN-FD ID Format" />
12:   </S>
13:   <S Name="Dlc">
14:       <EV Name="Dlc" Value="08"
15:         Description="CAN Dlc" />
16:   </S>
17:   <S Name="Data">
18:        <B Name="Data1" Value="00" />
19:        <B Name="Data2" Value="00" />
:              :
25:        <B Name="Data8" Value="00" />
26:   </S>
27: </SP>
```

Fig. 5. Example of a xml module description.

2) Indicate the CAN-ID constants for the fuzzing data to generate (in this case, fuzzing data for the CAN-IDs “ 001 ” and “ 002 ” will be generated) (lines 4-7).
3) In this implementation the first bit is for a standard or expanded ID, the second bit is the EDL, and the third bit is the BRS. The specified value is “ 06 ” and indicates the setting for a standard ID with CAN-FD and BRS (lines 8-12).
4) In this case, the length of the payload of the generated fuzzing data is a constant 8 bytes (lines 13-16).
5) Defines the payload. The substituted “ 00 ” is the default value when fuzzing is not being performed (lines 17 27).

The ”B” elements used in the definition of the payload in Figure 5 are the elements used when beSTORM generates test data. In the example of Figure 5, for the two frames in which the ID is “ 001 ” and “ 002 ”, beSTORM automatically generates test data where the payload is 8 bytes.

The data generated by beSTORM is transmitted from the COM port of the PC to DE0-NANO by the called function inside the DLL using the format shown in Figure 6. In this case, as shown in Figure 6, the content defined in Figure 5 is serialized in byte order, with a frame header and an operation code (OP code) at the start of the data, and a checksum attached to the end of the data, and transmitted to the COM port. This data is received in the DE0-NANO from the UART port, converted to a CAN/CAN-FD frame, and transmitted to the ECU that is the evaluation target. In Figure 6, the numbers within the fields show the number of bytes.

Header	Operation Code	CAN ID	Config	DLC	Payload	Checksum
2bytes	1byte	4bytes	1byte	1byte	0 ~ 64 bytes	1byte

Fig. 6. Format of the Frames Sent to the UART.

IV. CAN-FD Fuzzing Test

In order to verify the usability of the fuzzing tool developed in this research, we performed a measurement experiment of the execution time of a fuzzing test. This section explains the method of the measurement experiment performed in this research.

A. *Experimental Environment*

In all experiments, we used a windows7 64bits machine running on a 2.8GHz Pentium(R) Core i7 CPU processor with 8GB main memory.

In this experiment we connected the PCAN-USB FD to the CAN-FD bus for debugging. The PCAN-USB FD is a USB adaptor that can send and receive CAN messages on the CAN or CAN-FD bus. Packets received by the PCAN USB FD can be monitored with PCAN-View.

B. *Experimental Method*

The measurement experiment was performed by the following procedure. In this paper we call the CAN-FD message generated by beSTORM the fuzzing data.

- **Step1:** The ID, DLC, and payload of the data to be sent are defined in the module that defines CAN-FD, in order to specify the fuzzing data.
- **Step2:** The module of Step 1 is read by beSTORM and the test is executed.
- **Step3:** The packet transmitted on the bus is received by using PCAN-USB FD. The received message is monitored with PCAN-View.
- **Step4:** From the execution time displayed by beSTORM and the time of reception of PCAN-View, the time to receive the test data and the time required for the fuzzing test are calculated.

The interface developed in this research and PCAN-USB FD were connected as a test target ECU in Figure 7.

V. Measurement of the Processing Time and Analysis of the Results

Because the fuzzing test is time consuming method, we must calculate the time required for the fuzzing test using beSTORM. In this experiment we used two methods to perform the measurement experiment. Also, in this experiment we only used standard IDs for the CAN-IDs.

- ”The first experiment method” measured the time required to transmit one item of fuzzing data.

TABLE I
PROCESSING TIME FOR ONE ITEM OF FUZZING DATA

Payload Size (bytes)	8	12	16	20	24	32	48	64
Average Transmission Time for One Item of Fuzzing Data (ms)	152.207	151.976	149.925	150.602	152.207	152.207	154.083	153.846
Transmission Time Obtained from Formula (1) (μ s)	103.5	119.5	135.5	154.0	170.0	202.0	266.0	330.0
Transmission Time Obtained from Formula (2) (μ s)	118.0	138.0	158.0	180.5	200.5	240.5	320.5	400.5
Maximum Transmission Time Overhead (ms)	152.104	151.856	149.789	150.448	152.037	152.005	153.817	153.516

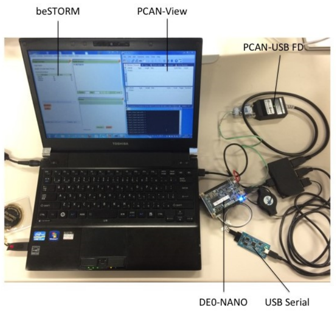

Fig. 7. Overview of our evaluation environment.

- ”The second experiment metod” used tens of thousands of fuzzing tests and measured the time required for the total test.

A. Measurement of the Time Required to Transmit One Item of Fuzzing Data

We transmitted 2,000 items of fuzzing data from beSTORM and took the average to find the time required to transmit one item of fuzzing data (Table I).

The processing required to transmit one item of fuzzing data can be divided into the following three steps:

- **(1):** Generation of the fuzzing data by beSTORM.
- **(2):** Processing to transmit to the UART of DE0-NANO. This is executed by the DLL that is called from the CAN FD module.
- **(3):** Processing to convert the data received from the UART into a CAN-FD packet and to transmit the packet on the bus.

The CAN-FD transmission time depends on the number of insertions of stuffing bits within the frames, and evaluations generally use a minimum transmission time (BCTT) or a maximum transmission time (WCTT). The minimum and maximum transmission times defined in reference [16], found by the following formulas, are shown.

$$BCTT(p) = 29t_{arb} + (27 + 5\lceil\frac{p-16}{64}\rceil + 8p)t_{data} \quad (1)$$

$$WCTT(p) = 32t_{arb} + (28 + 5\lceil\frac{p-16}{64}\rceil + 10p)t_{data} \quad (2)$$

Here, t_{arb} is the transmission time for the ID field, t_{data} is the transmission time for the data field, and p is the number of bytes for the data size. In this experiment the ID field and the data field were sent with the bit rates of 500 kbps and 2 Mbps, respectively, so $t_{arb} = 2(us)$ and $t_{data} = 0.5(us)$. At this time we expected the CAN-FD transmission time to require the maximum time, so we performed the evaluation using the value calculated by Formula (2).

1) Analysis of the Processing Time: The processing during times that are overhead corresponds to the time from when DE0-NANO receives the data that was sent by beSTORM, until it is converted into a CAN-FD packet. From Table I, within the fuzzing data transmission time, we find the fraction of the time that is overhead. If the payload size is 64 bytes then the transmission time is 153.846 ms, so if the time requirement for one fuzzing data transmission is T_{fuzz} then the fraction of the overhead relative to the total transmission time is:

$$\frac{T_{fuzz} - WCTT(64)}{T_{fuzz}} = \frac{153.846 - 0.4005}{153.846} = 0.99 \quad (3)$$

Therefore, we see that 99% of the time required for fuzzing data transmission is overhead separate from the CAN FD network transmission time. The overhead time breaks down into the following three items:

1) Creation of fuzzing data by beSTORM and the process of transmitting it to the UART
2) Transmission over the USB serial cable from beSTORM to DE0-NANO.
3) The process of converting from the UART of DE0-NANO into a CAN packet.

The processing flow for one generated fuzzing data item up until transmission to the PCAN-USB FD.

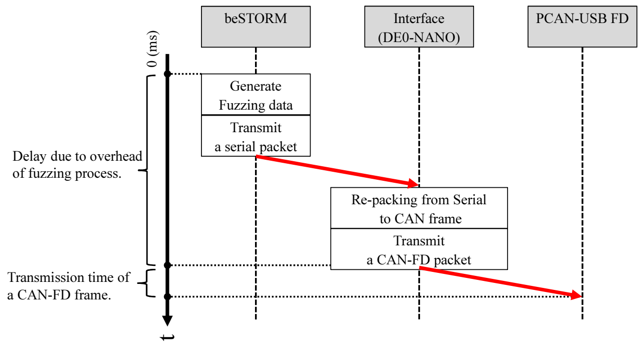

Fig. 8. Process flow of our developed fuzzing tool.

In this experiment, the time required to generate fuzzing data and the time until DE0-NANO receives the data could not be measured, so we have not been able to determine which process is the bottleneck. If the time required for (1) above is the bottleneck then using a higher-speed PC would be an improvement. Or, if the time required for (2) or (3) is the bottleneck, then using a different high-speed interface such as Ethernet to transmit the fuzzing data could be an improvement. Table I also shows that the transmission time for one item has almost no relationship to the payload size. As mentioned above, 99% of the transmission time is spent in processing prior to CAN-FD communication. The relationship between transmission time and payload load shows that the overhead time does not depend on the data length. Also, CAN-FD performs modulation so that as the data length gets longer the amount transmitted per unit time increases, and that characteristic can also be given as a reason why the transmission time does not increase even when the payload gets longer.

B. Measurement of the Time Required for a Complete Fuzzing Test

1) Measurement of the Processing Time: In order to find the effect of the payload length on the time required for a test we executed approximately 65,000 test cases for each of the three payload lengths of 8, 32, and 64 bytes (Table II).

TABLE II
TRANSMISSION TIME PER 1 FUZZING DATA ITEM.

Payload Size (bytes)	8	32	64
Number of Fuzzing Data Items Generated	65787	65599	65567
Fuzzing Execution Time (hr:min:sec)	02:46:59	02:42:56	02:51:28
Average Number of Items Transmitted Per Minute	394	402	382

2) Analysis of Processing Time: As mentioned in section V-A1, it is known that the transmission time per one fuzzing data item does not depend on the payload length. Table II shows that approximately 400 items of test data can be transmitted per minute regardless of the payload length. In an actual test it is expected that data of multiple payload lengths will be combined and sent in large quantities. If a test using 1,000,000 fuzzing data items were performed, in this environment a test time of approximately 42 hours would be required. For example, if one prepared a test environment with 10 machines, divided the test data into 10 parts, and executed tests in parallel, the testing would complete in approximately 4 hours. It is believed that this estimate is realistic for an actual testing environment. However, since 99% of the transmission time is overhead time in our test environment, even higher speeds can be expected. For example, if the overhead time were reduced by only 30% then it would be possible to complete the test in 30 hours, to improve the usability.

VI. CONCLUSION

The paper has reported an implementation of fuzzing for the CAN-FD protocol using beSTORM and has examined a measurement experiment that was performed on the execution time of a fuzzing test. With regard to the fuzzing execution time, we learned that about 400 tests can be performed in one minute, but as mentioned in section V-B2 it is possible to make the tests run faster by eliminating the bottleneck in the test data transmission time. An investigation of which process is the bottleneck is a challenge for the future. Also, in an actual test it would be necessary to evaluate whether the response is as expected for each fuzzing data item that was transmitted for testing. For that it is necessary to determine the expected response value based on the CAN-FD network design information and to build an environment for evaluating the actual response value into beSTORM. These two points remain as challenges for the future.

ACKNOWLEDGMENT

This work was supported by Ministry of Internal Affairs and Communications (MIC) in Japan, Strategic Information and Communications R&D Promotion Programme (SCOPE) Grant Numbers 152106005. This work was supported by JSPS KAKENHI Grant Number 16K16025.

REFERENCES

[1] Leohold, J.. Communication Requirements for Automotive Systems, 5th IEEE Workshop on Factory Communication Systems, 2004.
[2] International Organization for Standardization, Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling, ISO11898-1, 2003.
[3] International Organization for Standardization, Road vehicles - Local Interconnect Network (LIN) - Part 1: General information and use case definition, ISO/DIS 17987-1.
[4] International Organization for Standardization, Road vehicles - Communication on FlexRay - Part 1: General information and use case definition, ISO 10681-1, 2010.
[5] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, xperimental Security Analysis of a Modern Automobile, IEEE Symposium on Security and Privacy, 2010.
[6] C. Valasek, C. Miller, "Adventures in Automotive Networks and Control Unit", http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf, 2014.
[7] Specification of Module Secure Onboard Communication (SecOC) AUTOSAR Release 4.2.2, http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and-security/standard/AUTOSAR_SWS_SecureOnboardCommunication.pdf, 2014.
[8] International Organization for Standardization, " Road Vehicles: Controller Area Network (CAN), Part 1: Data Link Layer and Physical Signaling " ISO11898-1, 2015.
[9] S. Bayer, T.Enderle, D.K. Oka, M. Wolf " Security Crash Test - Practical Security Evaluations of Automotive Onboard IT Components ", In Automotive - Safety & Security 2015, Stuttgart, Germany, April 2122, 2015.
[10] T. Matsumoto, Y. Kobayashi, Y. Tsuchiya, N. Yoshida, N. Morita, S. Kayashima, " Methods of Fuzzing On-Vehicle ECUs Through CAN ", Institute of Electronics, Information, and Communication Engineers, SCIS 2015, 2015 (in Japanese).
[11] S. Bayer, A. Ptok, " Don 't Fuss about Fuzzing; Fuzzing In-Vehicular Networks ", In Embedded Security in Cars Conference (escar EU 2015), Cologne, Nov. 1112, 2015.
[12] T. Ziermann, S. Wildermann, and J. Teich. CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates. In DATE, pages 1088-1093. IEEE, 2009.
[13] R. Kurachi, H. Takada, M. Nishimura, S. Horihata," A New High-Speed Bus Topology LAN Protocol Compatible with CAN ", SAE Technical Paper 2011-01-1043, 2011, doi:10.4271/2011-01-1043.
[14] AIUTO Co., Ltd., beSTORM, http://www.bestorm.jp, 2015.
[15] TOPPERS Project, TOPPERS/ATK2, https://www.toppers.jp/atk2.html, 2015.
[16] U.D. Bordoloi, S. Samii, " The Frame Packing Problem for CAN-FD ", IEEE Real-Time Systems Symposium (RTSS) 2014, pp.284-293, 2014.