Abstract—Network embedded systems including Internet of Things (IoT) systems oblige developers to conduct frequent tests for various configurations, which require a significant amount of time and money. Moreover, the tests might not be performed due to the impossibility to acquire target devices produced by other factories or manufacturers. We propose a testing environment on a computer composed of target device emulators, and a repository of system configurations and test cases. Therefore, the environment performs tests for various configurations automatically and drastically improves the test efficiency.

I. INTRODUCTION

In development of network embedded systems including Internet of Things (IoT) systems, developers should perform tests for various system configurations whenever a system device is updated. They may spend a substantial amount of time and money in these tests. Furthermore, the tests might not be performed due to unavailability of some target devices.

We propose a testing environment to solve the above problems. This environment includes emulators to execute programs of target devices thereby eliminating the need for storage of devices. It also includes a repository of system configurations and test cases, which implements automatic test preparation and execution for various system configurations.

Several studies also propose methods for testing IoT systems using emulators. [1] introduces an embedded board which is used in substitution for various boards of IoT devices. [2] suggests a software platform with emulators which operates many containers implementing IoT applications and demonstrates device-to-device interactions. [3] shows a testbed of inter-connected, emulated IoT devices, which generates logs to detect anomalies. These methods are evaluated by scalability or similarity to the actual systems and improve the test efficiency. In contrast, our proposal aims further improvement of the product development efficiency and automatic setup of target devices on a commonly used computer as well as test execution. This is because a testing environment which requires special equipment is not necessarily used in all of the product development sites.

The remainder of the paper is organized as follows. Section II describes the problems with the tests in various system configurations and explains the proposed testing environment. Section III describes the evaluation method for the environment, and Section IV draws conclusions.

II. PROBLEM AND PROPOSAL

A. Problems of Tests in Various System Configurations

In network embedded systems, many types of devices communicate with each other, which are often produced by multiple manufacturers. Hence, we feature an air conditioning system as an example (See Fig. 1). The system includes one outdoor unit, one or more indoor units, one or more controllers, and some sensors. The operation commands from controllers, and the data for units and sensors are transmitted via a network. Each device consists of many models according to customer requests. The new and the old models can communicate with each other to enable unit replacement. Additionally, the system manufacturer frequently produces the units in multiple factories, and purchases the controllers and sensors from other manufacturers.

Therefore, there are two problems with the tests for various system configurations. First, it requires a considerable amount of time and money to store the devices for constructing all the configurations to be tested. A large storehouse is required to store the devices of all models of each type. A regular check of devices is also required to reduce the risk of the test not being performed, because it may be impossible to reacquire broken devices produced by other factories or manufacturers. Second, the developers spend a large amount of time to prepare the system to be tested. They should connect the devices to each other and perform some operations manually for each configuration. For example, in the system shown in Fig.1, developers should enroll units and sensors to the controller before each test.

B. Proposed Testing Environment

Fig. 2 shows the architecture of proposed testing
environment on a computer. It automates the tests of network embedded systems for various system configurations based on the following two concepts.

1. The environment eliminates the need to store all the devices to be tested.
2. It automates both test execution and preparation.

The device programs to be tested are stored in a repository. Each program is executed on an emulator for embedded software [4] as a process. The processes communicate with each other using the wrapper, which wraps the inter-device communication protocol with the inter-process communication protocol. The speed adjuster prevents state inconsistency among the processes by making them work at the same speed as the actual devices with sufficient accuracy, based on their program counters and CPU specifications. This architecture leads to concept (1) because the target devices are managed easily on the computers.

The repository also stores configuration files [5] and test case files. The configuration file provides information about the system configurations to be tested. The type, address in the system, and total number of devices are written in a specified format. The test case files describe the operation commands for setup and test execution, and rules to decide the result. Using these files, the automatic testing tool achieves concept (2). The tool performs the following processes:

- It executes each program on the emulator and sets the address based on the configuration file.
- It sets up the system automatically by sending operation commands from the test case files to the addresses set by the tool via wrapper. For instance, in the system shown in Fig. 1, the test case file describes a command for the controller to enroll the units and the sensors. If the configuration file defines the address of a controller as 001, the tool will automatically send the command to the specified address.

Further, it sends operation commands for test execution, and decides the results of the tests from outputs and program communication logs.

### III. Evaluation

We evaluated the proposed testing environment through two experiments.

Firstly, we set up the proposed environment on a Windows 7 computer with a CPU (2.90GHz, 8 cores) and examined CPU usage of the environment while embedded programs were being executed on the emulators at the same speed as the actual devices. The program originally works on a CPU (16MHz, 1 core) of an indoor unit in the air conditioning system shown in Fig. 1.

The results shown in TABLE I confirmed that the environment executed 60 indoor units without problems related to CPU usage. The number exceeded the maximum number of devices in the air conditioning system. These data imply that the environment enables to test many IoT systems on a commonly used computer because there are many IoT devices with a low-cost CPU (e.g., sensors).

<table>
<thead>
<tr>
<th>Number of executing indoor unit programs</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU usage (%)</td>
<td>8.0</td>
<td>14.7</td>
<td>25.8</td>
<td>41.7</td>
<td>59.1</td>
<td>82.3</td>
<td>100</td>
</tr>
</tbody>
</table>

Secondly, three developers performed manually one of the 127 function tests of a controller in the air conditioning system shown in Fig. 1, followed by testing with the proposed environment.

The environment consequently automates test preparation and execution, except file selection. Work time of each developer is shown in Table II. We confirmed from the results of manual method that preparation requires more time. In contrast, the environment reduced the work time by 24.7 minutes on an average for the three developers. Therefore, we expect a reduction by 52 hours in 127 function tests. We found that the environment was especially effective for developer 2, who was not familiar with the system. Developer 2 had difficulty finding the devices because their hardware was standardized in each type and developers were forced to confirm printed information on the hardware in detail. These

<table>
<thead>
<tr>
<th>Work item</th>
<th>Developer 1</th>
<th>Developer 2</th>
<th>Developer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect and connect devices</td>
<td>7.9</td>
<td>0.1</td>
<td>24.7</td>
</tr>
<tr>
<td>Check devices</td>
<td>4.8</td>
<td>0</td>
<td>6.5</td>
</tr>
<tr>
<td>Operate devices for setup</td>
<td>7.1</td>
<td>1.1</td>
<td>7.3</td>
</tr>
<tr>
<td>Test execution</td>
<td>3.3</td>
<td>2.6</td>
<td>5.2</td>
</tr>
</tbody>
</table>
data indicate that the effectiveness of our environment does not depend on individual skills.

IV. CONCLUSION

We proposed an automatic testing environment for network embedded systems. It solves the two problems of manual testing for various system configurations.

It eliminates the need to store the devices by enabling the devices to work and communicate with each other on computers, using the emulator for the embedded software. It performs automatic setup and test execution using the configuration and test case files.

We evaluated our proposed environment in an air conditioning system test and confirmed that the work time reduced drastically. The environment improves efficiency of frequent tests for network embedded systems.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI (grant no. 16K00100) and CREST, JST.

REFERENCES


