Model-Based Testing of Internet of Things Protocols

van Dommelen, X.M.; van der Bijl, M.; Pimentel, A.

DOI
10.1007/978-3-031-15008-1_12

Publication date
2022

Document Version
Final published version

Published in
Formal Methods for Industrial Critical Systems

Citation for published version (APA):
van Dommelen, X. M., van der Bijl, M., & Pimentel, A. (2022). Model-Based Testing of Internet of Things Protocols. In J. F. Groote, & M. Huisman (Eds.), *Formal Methods for Industrial Critical Systems: 27th International Conference, FMICS 2022, Warsaw, Poland, September 14–15, 2022 : proceedings* (pp. 172-189). (Lecture Notes in Computer Science; Vol. 13487). Springer. https://doi.org/10.1007/978-3-031-15008-1_12

Model-Based Testing of Internet of Things Protocols

Xavier Manuel van Dommelen[1,2](✉), Machiel van der Bijl[2],
and Andy Pimentel[1]

[1] University of Amsterdam, Amsterdam, The Netherlands
xavier_vd@outlook.com
[2] Axini, Amsterdam, The Netherlands
https://www.axini.com

Abstract. Internet of Things (IoT) is a popular term to describe systems/devices that connect and interact with each other through a network, e.g., the Internet. These devices communicate with each other via a communication protocol, such as Zigbee or Bluetooth Low Energy (BLE), the subject of this paper. Communication protocols are notoriously hard to implement correctly and a large set of test-cases is needed to check for conformance to the standard. Many of us have encountered communication problems in practice, such as random mobile phone disconnects, difficulty obtaining a Bluetooth connection, etc. In this paper, we research the application of industry strength Model-Based Testing (MBT) within the IoT domain. This technique contributes to higher quality specifications and more efficient and more thorough conformance testing. We show how we can model part of the BLE protocol specification using the Axini Modeling Platform (AMP). Based on the model, AMP is then able to automatically test the conformance of a BLE device. With this approach, we found specification flaws in the official BLE specifications as well as conformance errors on a certified BLE system.

Keywords: Model-Based Testing · Internet of Things · Communication Protocol · Bluetooth Low Energy · Embedded Systems

1 Introduction

The term *Internet of Things (IoT)* has become well known. IoT generally refers to everyday objects that have obtained the ability to connect and interact with each other through a network [33]. Over the years, the number of these IoT devices has grown tremendously, reaching an approximate amount of 9.9 billion devices in 2021 [17]. Along with this growth, new IoT devices are being developed that often implement the same widely accepted communication protocols [2]. Examples are Bluetooth Low Energy [6] and Zigbee [11]. It is important that these protocols are implemented correctly. When the implementations deviate from the specification, the functionality to interact with other systems using the same communication protocol could be affected.

J. F. Groote and M. Huisman (Eds.): FMICS 2022, LNCS 13487, pp. 172–189, 2022.
https://doi.org/10.1007/978-3-031-15008-1_12

Currently, manufacturers face several challenges that prevent them from extensively testing the communication protocols in their IoT devices [7]. For this reason, research has started looking into different testing approaches to overcome these challenges. One of these approaches is *Model-Based Testing (MBT)* [8].

Research on MBT for IoT protocol testing looks mainly into proof of concepts and investigates individual challenges [1,16,29]. As a result, it is difficult to evaluate to what extent MBT is capable of resolving the problems in this domain. Such an evaluation is needed to compare testing approaches to determine which one is the most optimal, in particular in an industrial setting. For this reason, our research tries to investigate which challenges industrial strength model-based testing is able to resolve and what other influences this approach brings.

1.1 Related Work

IoT can be seen as cyber physical systems. Model-based testing is an interesting technique that has shown its merits in modeling and testing cyber physical systems [30,31].

Our work focuses on testing protocol conformance through MBT on IoT systems, but there is related work that researches other aspects. The work of Yoneyama et al. [34] uses MBT to test the robustness of the COAP protocol by modeling network faults. Additionally, the work of Aziz et al. [3] demonstrates that by formally modeling the MQ Telemetry Transport protocol, an IoT protocol, and analyzing the result, they can evaluate the correctness of the protocol. These papers differ from our work by concentrating on testing the protocol itself instead of testing the conformance of the implemented protocol. Malik et al. [22] use MBT as a tool to demonstrate that we can automatically test IoT protocols on systems remotely. In their work, they briefly describe why they make use of MBT but their main topic is the framework for remotely testing IoT systems and their protocols. The case study of Tappler et al. [29] shows how models for a model-based testing approach can be automatically created through active automated learning. Furthermore, this work demonstrates that by using their automatically generated models they are capable of finding implementation mistakes that go against the MQTT communication protocol specifications. Ahmad et al. [1] investigate the possibility to use a model-based testing approach to test IoT systems in their entirety. In addition to just testing the system, they discuss a framework that enables sharing models between developers as a service. While the focus of these papers is to obtain a proof of concept with a specific goal in mind, our work differs by highlighting the implications of using MBT in the IoT domain in an industrial setting.

Finally, the work of Inçki et al. [16] presents a model-based testing implementation in which they could perform interoperability tests to evaluate the IoT communication protocol COAP. However, they do not present any experiments that make use of their presented approach. Consequently, we are not able to evaluate the benefits or disadvantages of using MBT in contrast to our work. Furthermore, they do not give an in-depth explanation and reflection on the implications of using model-based testing.

1.2 Contributions

Our research focuses on the application of MBT with a commercial tool on a non-trivial part of the industrial BLE protocol. We describe which implications MBT could have on the IoT protocol testing domain based on practical experience. Furthermore, we discuss how the specifications of a widely used IoT communication protocol, Bluetooth Low Energy, can be translated into a formal model. Based on this experience, we discuss which obstacles are likely to be encountered and how they can be overcome when translating an IoT protocol. In this process, we highlight several flaws in the official Bluetooth Low Energy specifications version 4.2 [6], showing that MBT is a method to improve specifications. Finally, by applying our proof of concept to test a certified BLE system, we show that certain assumptions about MBT also hold in practice. And we find implementation errors in the process.

2 Preliminaries

2.1 Internet of Things

IoT refers to everyday objects that have obtained the ability to connect and interact with each other through a network [33]. According to Elnashar [10], the challenges related to IoT fall into two categories: challenges relating to unlicensed networks that aim for short-range communication and challenges relating to cellular licensed networks.

This document focuses on the short-range communication category, because this category contains significantly more manufacturers [10,15,26]. This means that to ensure interoperability between IoT systems more parties require a sufficient testing environment. Additionally, IoT systems from this category generally use the same communication protocols [2,32]. As a result, a generic testing environment becomes more important since this would benefit all the different manufacturers.

One of the popular communication protocols in the IoT domain is the Bluetooth Low Energy (BLE) protocol [2]. This protocol is known for its low power consumption, low setup time, and supporting star network topology with unlimited number of nodes. BLE systems can receive a certificate indicating that their system conforms to the BLE specifications when they pass a list of unit tests defined by the organisation behind BLE, Bluetooth SIG[1].

2.2 Model-Based Testing

Software testing verifies that a software system implements its requirements. Such a verification can be done in four steps [18,20,30]: specification interpretation, test creation, test execution, and test result evaluation. Model-Based Testing (MBT) is a method that can automate all of these steps except for the

[1] https://www.bluetooth.com/.

specification interpretation step by using a formal model defining the requirements/specifications [8,31]. The model describes the behavior of the System Under Test (SUT) in terms of how the inputs and outputs of the SUT relate, and uses this formal definition to generate and execute test cases to evaluate the correctness of the SUT. A testing environment using MBT generally requires three key technologies [8]: Modeling Language, Test Generation, and a Supporting Infrastructure. Figure 1 gives an overview of the components which we discuss below.

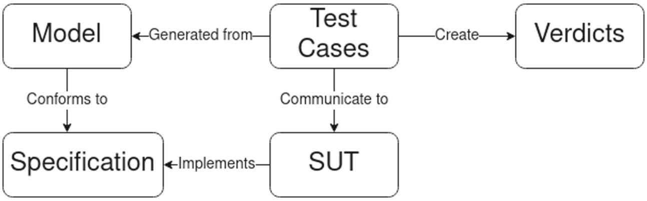


Fig. 1. Model-Based Testing pipeline [18]

Modeling Formalism. There are several modeling formalisms that can be used in MBT [31], for example Finite State Machine (FSM), Labeled Transition System (LTS) [30], Unified Modeling Language (UML) [4], and Symbolic Transition System (STS) [12]. FSMs and LTSs are often used for MBT [31]. To describe a SUT using an LTS, a set of states and transitions are defined. The transitions are used to reflect the correct behavior between the different states in which the SUT could be. Finally, a Symbolic Transition System (STS) is an extension to an LTS that introduces the concept of data to the models. This addition of data is relevant since it allows us to prevent a state-space explosion when dealing with data structures [12].

Test Case Generation. Based on a formal model an algorithm can generate test cases automatically. Using this approach, a large number of test cases can be generated. Due to time-constraints it is not always possible to execute all of these test cases, therefore we need *test criteria* [24,27] to limit the number of generated tests. Test cases consist of two ingredients: *stimuli* which represents inputs to the SUT and a set of allowed *responses* which represent possible outputs from the SUT. Once a test case is generated, stimuli will be passed on to the SUT and observed outputs are presented to the testing environment. The MBT tool checks if the observed responses are defined in the model. If this is the case, the test case will pass otherwise it will fail. For the assignment of verdicts a correctness notion between the model and the SUT, a so called conformance relation, is important. The conformance relation that we use is the *input-output conformance*, *IOCO* theory [30] which also uses STSs [13].

In order to automatically execute test cases we need some supporting infrastructure. The connection to the SUT is often implementation specific, in our case BLE. The connection to the MBT tooling is often standardized.

2.3 Axini Modeling Platform (AMP)

For our research we use the Axini Modeling Platform. Axini is a product company that specializes in modeling and model-based testing. AMP is an industry strength MBT tool that is used in Finance, Rail and High-tech. It is based on the IOCO theory and research from Tretmans [30].

AMP uses a modeling language called the Axini Modeling Language (AML). This language is inspired by ProMeLa, the language of the Spin model checker [21]. The semantics of the language is expressed in STS. The reason we choose AMP is: the modeling language is suited to model cyber physical systems, AMP is a proven industry grade platform (10+ years) that can handle big industrial systems and models with big state spaces. Examples are safety-critical rail systems, pension and insurance systems and cyber-physical systems.

3 MBT in the Context of IoT

3.1 IoT Testing Challenges

Looking at existing literature, we see that one of the overarching challenges for the industry to make fully conformant BLE devices, is that it costs too many resources to obtain and maintain an extensive test-suite [7,19,23,29]. First, the protocols from this domain change regularly [19,28]. As a result, testing environments need to be updated frequently and thus require significant maintenance [35]. Another obstacle is the large number of test cases necessary to test for conformance. IoT protocols, such as BLE, contain a wide range of different potential configurations. Optimally, a tester would test all combinations to test for conformance. However, with conventional manual methods, this becomes too expensive [7,19]. Finally, the quickly changing protocols also require backwards compatibility. Manufacturers are required to test against systems implementing older supported protocol versions.

3.2 Positioning MBT in IoT

MBT holds several benefits over traditional testing techniques. One benefit is that the resulting testing environment can quickly respond to changing specifications [24,31]. Changes made within the model are easier to maintain than manually changing individual low-level test cases when requirements change. Because frequently changing specifications are a problem, MBT would give a benefit over traditional testing methods that do not use an abstract representation within this domain.

Another benefit is that MBT results in arguably better tests compared to the manually created tests [5,31]. Pretschner [25] presents this with a different angle.

He mentions that the resulting tests cases are not necessarily of higher quality but that the higher quantity is the cause for a better testing environment. This higher number of test cases results in a higher coverage. In the IoT context, because it is difficult to obtain high coverage, this is a desired trait. MBT makes this possible through its high level of automation.

For MBT to reach this high level of automation, a model is required before testing can begin. The creation of such models is a non-trivial process, resulting in an additional potentially time-consuming step [5,9,24]. Consequently, it potentially takes longer before testing can begin compared to other methods that do not require this step. The modeling step also brings benefits. Because a modeler needs to critically think about the specifications for the creation of the model, this increases the chance of finding specification flaws [24,31]. This is specifically relevant in the IoT domain, where different manufacturers all need to follow the same specifications. Additionally, because manufacturers need to follow the same specifications, one model should suffice to supply every manufacturer with an extensive testing environment.

Based on the literature, we believe that MBT can form a solution to overcome the problems in the IoT testing domain if the previously discussed assumptions hold.

4 The AMP MBT Environment to Test BLE IoT Systems

To evaluate the assumptions from the previous section we require an MBT environment that can test the conformance of BLE devices. In this section, we discuss our design decisions, experience, and findings when implementing such an environment on the AMP platform.

4.1 SUT

For our experiment, we decided to model and test systems that implement the BLE specifications version 4.2 [6] from the official Bluetooth organization: Bluetooth SIG[2]. This version was chosen because a system running this version was easily accessible for experiments. Based on our experience we believe that the resulting process would be similar to other protocol versions.

One can access a Bluetooth Controller's capabilities through the Host Controller Interface (HCI) [6]. This interface functions as an API to perform specific actions on the different lower-level software layers on a Bluetooth system. We use this to test the conformance of the BLE protocol on a system.

The specifications of BLE describe the protocol using different *layers*. Each of these layers has its requirements and provides specific functionality. For the scope of our research, we decided to model the Link Layer. This layer describes the steps that two systems implementing the BLE protocol should take to obtain and sustain a connection. If a manufacturer makes a mistake in the implementation of this layer, it can directly influence the interoperability. Because interoperability is an important factor for IoT systems, we decided to model this specific layer.

[2] https://www.bluetooth.com/.

4.2 Model Creation

We will use a representation of the Link Layer's behavior, see Fig. 2, to highlight which parts we implemented within our model. The states within this figure that are accessible within our model are marked green.

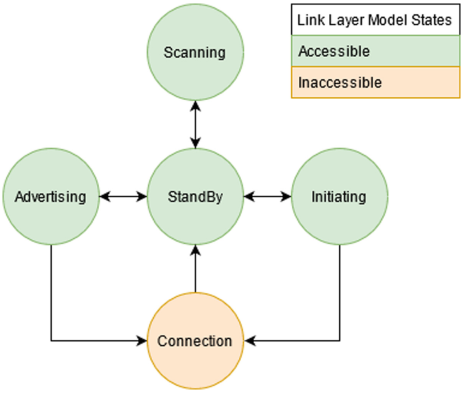


Fig. 2. State diagram of the Link Layer state machine on the Low Energy Controller according to the Bluetooth Core Specification version 4.2 [6] (Color figure online)

Due to time constraints, we decided not to model the *Connection* state, marked orange. Being able to also test this behavior would extend our work such that we could also directly evaluate interoperability between systems. We leave this to future work. Given the experience with the scale of models in AMP we do not expect any problems with such an extension. Finally, a full version of our obtained model can be requested by contacting Axini.

Model Overview. For the creation of our model, we used the state machine from Fig. 2 as our starting point. We decided to use the same states within our model and search through the specifications to look for the corresponding HCI commands for the basic transitions.

Using the HCI command descriptions as a foundation, we concluded that the following HCI commands would be most applicable to reflect the state transitions:

- HCI_LE_Set_Advertise_Enable. Handles the transitions between the *StandBy* and *Advertising* state.
- HCI_LE_Set_Scan_Enable. Handles the transitions between the *StandBy* and *Scan* state.
- HCI_LE_Create_Connection. Handles the transition from the *StandBy* to the *Initiating* State.
- HCI_LE_Create_Connection_Cancel. Handles the transition from the *Initiating* state to the *StandBy* state.

To model the different configurations, we selected the configuration options for the *Scanning* and *Advertising* state. The model represents this using transitions that go towards the same state after successfully changing the state configurations. The HCI commands that resemble these transitions are:

- HCI_LE_Set_Advertising_Parameters
- HCI_LE_Set_Scan_Parameters

4.3 AML Model Example

Given the scope of this paper it goes too far to introduce the entire AML modeling language. Instead we treat a part of the model and we show a part of the visualization of the model. The visualization is shown in Fig. 3. The model uses similar states as the state machine from the BLE specification in Fig. 2: Scanning, Advertising, Standby, Initiating.

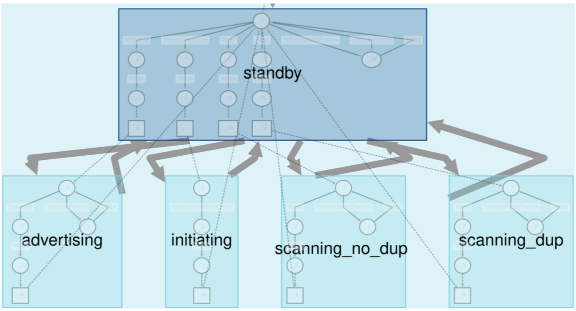


Fig. 3. AMP model visualization State diagram of the Link Layer

To give the reader some idea of what AML looks like, we discuss a simplified model in which we can successfully enable advertising following the BLE protocol. This model is shown in Listing 1.1.

Listing 1.1. AML model example

```
process('main') {
    stimulus 'hci_le_set_advertise_enable',
        'advertising_enable' => :integer
    response 'status', 'code' => :integer

    state 'standbyState'
        receive 'hci_le_set_advertise_enable',
            constraint: 'advertising_enable==1'
        send 'status', constraint: 'code==1'
        goto: 'advertiseState'
    ...
}
```

In this model we define a process named ‘main’. This process has one interface with one stimulus (input) ’hci_le_set_advertise_enable’ and one response (output) ’status’; both have an integer parameter. The process shows a *state* with two actions: after the SUT receives a *hci_le_set_advertise_enable* stimulus with the *advertising_enable* parameter set to 1 it should give a *status* response back with a value of 1. The test case will continue from the *advertiseState* and pick a new action to test. For these tests, the stimulus parameters are solved with a constraint solver. We use constraints to define more complex input domains to model the other commands and different scenarios.

Model Configurations. In addition to the model that reflects BLE specifications, we added several model configuration options. A tester can use these configurations to manage to what extent the model is used during the generation of test cases. A list of supported configuration options is shown in Table 1.

Table 1. Model configuration options

ID	Configuration	Data Type	Motivation
1	error_paths	Boolean	Scenario Simulation
2	error_self_loop_paths	Boolean	Assumption due to underspecification
3	error_future_param_paths	Boolean	Assumption due to underspecification
4	error_validation_strength	Integer	Assumption due to underspecification
5	scan_between_duplicates	Boolean	Assumption due to underspecification
6	force_link_layer_transitions	Boolean	Assumption due to underspecification

- Configuration 1 allows one to trigger transitions that would result in an error code.
- Configuration 2 allows one to trigger transitions that could change the state but instead would result in the same state.
- Configuration 3 allows to trigger transitions that would result in an error because parameter values would be used that are reserved for future usage.
- Configuration 4 accepts five different strength values:
  - With strength 0 all error codes are allowed when an error code is expected.
  - With strength 1, only the error codes that are specifically mentioned in the specifications need to correspond to any of the expected errors if multiple errors could be thrown. Otherwise all error codes are accepted.
  - With strength 2 we have the same situation as with strength 1 however we apply our assumption on which error has a precedence when multiple errors could be thrown thus only allowing only one error code.
  - When strength 3 we only accept one or more of the expected errors. There is no precedence check.
  - Finally, with strength 4 we only accept the error codes with the highest precedence according to our assumptions.

– Configuration 5 allows one to trigger transitions that would move between the two possible scanning states in which *Filter_Duplicates* is enabled or disabled.
– Configuration 6 allows one to trigger transitions that would check if transitions that would not be possible according to the link layer specification result in the correct error code.

Motivation. While investigating the BLE specifications for the model, we found several topics that contained underspecifications. As a result, a developer can have different interpretations of what a *correct* BLE implementation would be. For these topics, we made assumptions about what the correct behavior of the protocol should be. However, it is also possible that a tester disagrees with our design. To compensate, we added configuration flags that allow a tester to configure the model such that test cases related to these assumptions will not get generated.

Findings. During the development of the model, we encountered several obstacles. The first obstacle is related to finding a point from which a tester can start modeling using the BLE specifications. The extensive specifications make it difficult to find a starting point. However, after finding this point, the remaining modeling process became straightforward. Additionally, the creation of the model became a time-consuming process because of the complexity of the BLE protocol. The protocol defines actions that contain many rules and can be different based on the system's state. Doing this correctly requires a tester to fully understand the specifications and reflect this flawlessly in a model resulting in a time-consuming process. These findings support the assumption that the modeling step is a time-consuming process. The authors think this could be significantly reduced if a BLE expert is available during the modeling process. Preferably the modeling takes place during the specification process.

After performing the modeling step, we see that the model is not our only result. During the process, we also discovered several flaws in the official BLE specifications. Most of these flaws are related to underspecification, but we also found a place where the specifications were contradicting. As a result, our experience confirms the assumption that we can find specification flaws during the modeling step and use this as a method to refine the specifications.

Limitations. We mentioned earlier that we use HCI commands to interact with a BLE system. These commands require two types of parameters. The first type contains parameters that together define which command should be running: OpCode Group Field, Opcode Command Field, and the expected resulting event code. We decided to separate these parameters from our model and put them as constants in our adapter. As a result, we limited the model to a static set of HCI commands that it can simulate. The second type contains parameters that define the configuration for an HCI function. According to the specifications, these parameters have a maximum memory size. We also followed this limitation in our model, but as a consequence, we are unable to test outside this memory range.

Finally, some HCI commands we simulate can generate additional event codes in the background. In our model, we only simulate the response code behavior, but for future work, we recommend also taking these event codes into account.

4.4 Adapter

As discussed in the background, the purpose of the adapter is to handle the communication between the SUT and the testing environment, AMP. Additionally, the adapter contains the translation logic from model labels into SUT actions and vice versa. This translation was straightforward to implement because our model follows the BLE specifications. However, we found that programming the communication with the HCI layer is a rather tedious task. The reason for this is that the documentation about programming on the HCI layer is scarce [14]. Consequently, the adapter step, which is supposed to be relatively small compared to modeling, became a more time-consuming process than expected. In the end, we decided to go with a Python implementation for the adapter. This adapter uses the PyBluez library[3] to communicate with the HCI of a BLE system.

5 Testing BLE Using AMP

By using AMP with the described model and adapter from Sect. 4, we can test any BLE system that implements BLE version 4.2. In this section, we describe how we test such a system, an *Intel Dual Band Wireless-AC 8265 [Bluetooth adapter]*. This SUT has received a certificate[4] from the official Bluetooth SIG organization indicating that they have correctly implemented BLE version 4.2.

To evaluate our approach and the SUT, we perform two experiments. The first experiment tests if we can find conformance errors using the platform. The second experiment looks into our found underspecifications that can potentially lead to implementation assumptions.

5.1 Assumption

A fundamental assumption we make for our experiments is that the test platform does not contain errors. In other words, we assume that the model, adapter, and testing environment (AMP) are all implemented correctly. Using this assumption, we can conclude that the found mistakes are caused by the SUT and not by potential flaws in one of these components. Our thorough analysis of the findings support this assumption.

[3] https://github.com/pybluez/pybluez.

[4] https://launchstudio.bluetooth.com/ListingDetails/3524.

5.2 Test Generation Configurations

Using our model-based testing platform, we can generate test cases to test a given SUT. The size of the test cases are configured by the tester and influence how much of the model can be traversed during one test-case. Additionally, a tester can set the number of test cases that during a test run are generated. Similar to the first configuration, this configuration influences the test coverage that can be obtained.

For our experiments, we wanted to obtain a model coverage of 100% to at least test each transition once. Through manual experiments, we found that this coverage can be achieved within a test case by setting the size to 30. Additionally, we decided to set the number of test cases that are generated during one test run to 20. We found this number to be enough for our goal to demonstrate that we can find conformance errors.

5.3 Conformance Experiment

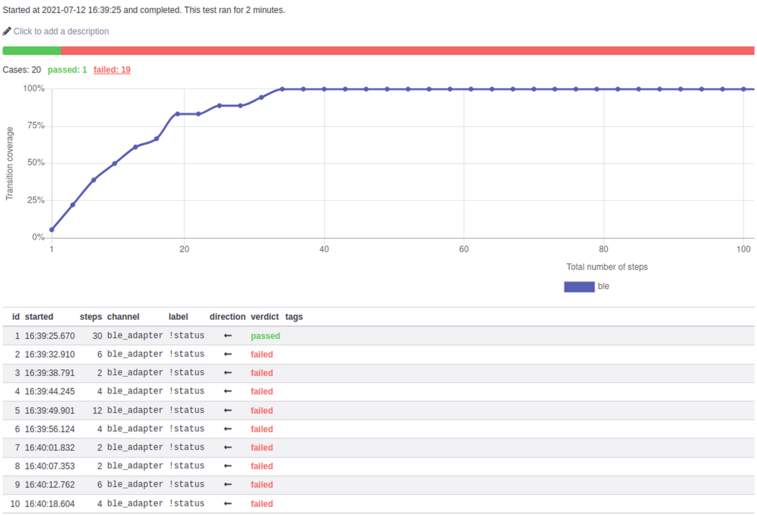
In this experiment, we test the SUT using the previously discussed test generation configurations. Additionally, in Sect. 4.3, we discussed model configurations to enable and disable some of our assumptions regarding what the correct implementation should be. Because we do not want to leave room for discussion after we would find a conformance mistake, we decided to disable all configurations regarding assumptions.

Results. Running the testing environment with the previously described configurations, we obtain the results that are displayed in Fig. 4.

The results from Fig. 4 show us that we can obtain a Transition Coverage of 100%. Furthermore, the results show us that we can automatically find 19 test cases where the SUT does not conform to our model. If we would categorize our failed test cases based on which behavior deviates from the specifications, we obtain the categorization as shown in Table 2.

Table 2. Overview of the failed test cases and their cause using results from the *Conformance Experiment*

Test-case ID	State	Label	Expected	Output
2,4,5,7,15,17,18,20	Scan	setAdvertisingParams	0	18 (invalid parameters)
3,11,13,14	StandBy	setScanParams	0	18 (invalid parameters)
8,19	StandBy	createConnection	0	18 (invalid parameters)
9,12	StandBy	setAdvertisingParams	0	18 (invalid parameters)
10	Advertise	setScanParams	0	18 (invalid parameters)
16	StandBy	createConnection	0	13 (limited resources)



Started at 2021-07-12 16:39:25 and completed. This test ran for 2 minutes.

✎ Click to add a description

Cases: 20 passed: 1 failed: 19

id	started	steps	channel	label	direction	verdict	tags
1	16:39:25.670	30	ble_adapter	!status	←	passed	
2	16:39:32.910	6	ble_adapter	!status	←	failed	
3	16:39:38.791	2	ble_adapter	!status	←	failed	
4	16:39:44.245	4	ble_adapter	!status	←	failed	
5	16:39:49.901	12	ble_adapter	!status	←	failed	
6	16:39:56.124	4	ble_adapter	!status	←	failed	
7	16:40:01.832	2	ble_adapter	!status	←	failed	
8	16:40:07.353	2	ble_adapter	!status	←	failed	
9	16:40:12.762	6	ble_adapter	!status	←	failed	
10	16:40:18.604	4	ble_adapter	!status	←	failed	

Fig. 4. Screenshot of AMP showing a partial overview of the resulting test cases and their evaluations using the configurations as discussed for the *Conformance Experiment*

This overview shows that we can find 6 different conformance mistakes based on the specifications. Furthermore, we see that most failed cases are caused by inaccurate error responses when using valid parameters.

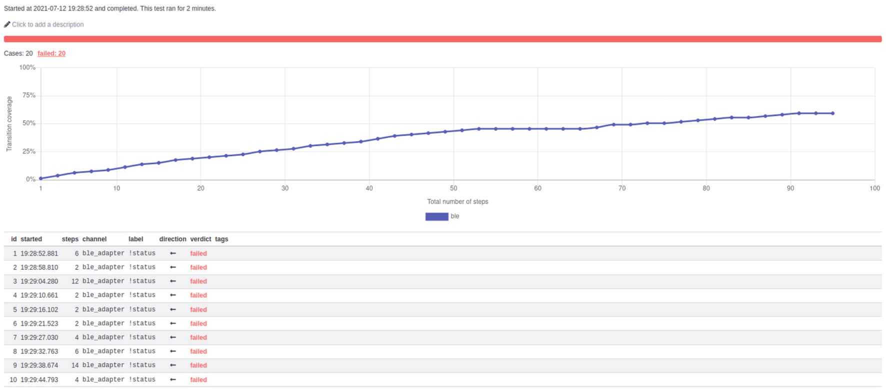
Nonetheless, some of these error categories may be caused by the same underlying problem. As a result, this overview might show more errors than the SUT contains. However, the fact remains that we can find conformance flaws in a certified BLE system by applying a state-of-the-art MBT tool.

5.4 Model Assumption Experiment

For this experiment, we want to investigate our found underspecifications. By running our testing environment, while enforcing all of our assumptions through the model configurations, we can investigate if the SUT's implementation is different from our definition of a *correct* implementation. If we find implementation differences, we can confirm that manufacturers have different interpretations of what the correct behavior is when following the BLE specifications. Such findings can support the idea that our found underspecifications are a problem.

Results. Running the testing environment using our enforced assumptions on the SUT resulted in the test-run overview shown in Fig. 5.

First, these results show that enforcing our assumption configurations results in a Transition Coverage of 59.49%. Consequently, our test run does not cover the entire model. However, within this test run, we can still find behavior on the SUT that deviates from our assumptions. Table 3 shows an overview of the related conformance errors.



Started at 2021-07-12 19:28:52 and completed. This test ran for 2 minutes.

Click to add a description

Cases: 20 failed: 20

id	started	steps	channel	label	direction	verdict	tags
1	19:28:52.881	6	ble_adapter	!status	←	failed	
2	19:28:58.810	2	ble_adapter	!status	←	failed	
3	19:29:04.280	12	ble_adapter	!status	←	failed	
4	19:29:10.661	2	ble_adapter	!status	←	failed	
5	19:29:16.102	2	ble_adapter	!status	←	failed	
6	19:29:21.523	2	ble_adapter	!status	←	failed	
7	19:29:27.030	4	ble_adapter	!status	←	failed	
8	19:29:32.763	6	ble_adapter	!status	←	failed	
9	19:29:38.674	14	ble_adapter	!status	←	failed	
10	19:29:44.793	4	ble_adapter	!status	←	failed	

Fig. 5. Screenshot of AMP showing a partial overview of the resulting test cases and their evaluations using the configurations as discussed for the *Model Assumption Experiment*

Table 3. Categorised implementation errors related to different underspecification assumptions using the configurations as discussed for the *Model Assumption Experiment*

Assumption Configuration	Number of Failed Test Case(s)
error_self_loop_paths	6
error_future_param_paths	13
error_validation_strength	0
scan_between_duplicates	1
force_link_layer_transitions	0

Based on this categorization, we can confirm that the SUT behaves differently regarding three of our specification assumptions. As a result, our approach can highlight three topics within the specification that could lead to different implementations due to underspecification.

6 Discussion

The *conformance* experiment from Subsect. 5.3 shows us that we can find conformance flaws in a certified BLE system. This suggests that MBT can test more thoroughly than the testing environment that was used for the BLE certification of the SUT. This means that MBT can assist in obtaining more extensive testing environments and thus can assist in improving conformance and finally interoperability on IoT systems. Additionally, because our experiment showed that we can test BLE systems, a similar approach can be used to also test other communication protocols within the IoT domain.

One of the potential benefits discussed in Sect. 3 is that MBT can be used to refine the specifications of a tested system. During the *assumption* experiment from Subsect. 5.4, we show that specification flaws can be discovered during the creation step of the model. This suggests that the assumption that MBT can help refine the specifications also holds for BLE. Consequently, we can assume that this will also hold for other IoT communication protocols. As a result, MBT can become a method to refine the different communication protocol specifications. Such refinements will improve the overall interoperability within the domain because different manufacturers will be able to obtain more conformant implementations.

Based on our results, we decided to get in touch with Bluetooth SIG to highlight our results. We sent an e-mail after crosschecking if these flaws also remained in the latest, 5.2, specifications. As of writing this paper, we have not received a response.

Furthermore, we discussed our approach and findings with the creator of Bluetooth, Dr. Ir. Jaap C. Haartsen. In this meeting, he highlighted the current problems in the IoT Bluetooth domain. He mentioned that interoperability with machines from other manufacturers is a challenge for IoT manufacturers. In this context, it would be interesting to extend our work to the higher software layers that apply the BLE protocol.

7 Conclusion

It is crucial for IoT systems that the communications protocols such as BLE conform to the protocol's specifications. In our research, we have shown that manufacturers struggle to obtain testing environments that can test the specification conformance of their systems. Our experiments confirmed this by demonstrating that we can find conformance flaws in a certified BLE system using our proposed MBT environment. Additionally, we showed that we can find weaknesses in the official BLE specification by using MBT. Correcting these flaws will allow different manufacturers to create implementations that are more conformant and thus will assist in ensuring interoperability. Finally, based on these findings, we believe that MBT can be a solution within the IoT protocol testing domain using existing MBT tools such as AMP.

7.1 Future Work

Our work focuses on researching the possibilities of MBT to test IoT protocols. However, our research does not perform a comparison study with other testing methods for this domain. The next step would be to compare this method to other testing methods and discuss what method would be the most optimal for this domain. Another direction that research could look into is testing the interoperability between IoT systems. This direction would be interesting because our work assumes that conformance errors will result in interoperability issues but does not test it directly. Finally, because our testing environment was able to find conformance errors on a certified BLE system, it becomes interesting to research if such errors also occur on more systems in the market.

References

1. Ahmad, A., Bouquet, F., Fourneret, E., Le Gall, F., Legeard, B.: Model-based testing as a service for IoT platforms. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 727–742. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_55
2. Al-Sarawi, S., Anbar, M., Alieyan, K., Alzubaidi, M.: Internet of things (IoT) communication protocols. In: 2017 8th International Conference on Information Technology (ICIT), pp. 685–690. IEEE (2017)
3. Aziz, B.: A formal model and analysis of an IoT protocol. Ad Hoc Netw. **36**, 49–57 (2016)
4. Bernard, E., et al.: Model-based testing from UML models. INFORMATIK 2006-Informatik für Menschen-Band 2, Beiträge der 36. Jahrestagung der Gesellschaft für Informatik eV (GI) (2006)
5. Binder, R.V., Legeard, B., Kramer, A.: Model-based testing: where does it stand? Commun. ACM **58**(2), 52–56 (2015)
6. Bluetooth SIG: Core specification 4.2 (2014). https://www.bluetooth.com/specifications/specs/core-specification-4-2/. Accessed 28 June 2021
7. Bures, M., Cerny, T., Ahmed, B.S.: Internet of things: current challenges in the quality assurance and testing methods. In: Kim, K.J., Baek, N. (eds.) ICISA 2018. LNEE, vol. 514, pp. 625–634. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-1056-0_61
8. Dalal, S.R., et al.: Model-based testing in practice. In: Proceedings of the 21st International Conference on Software Engineering, pp. 285–294 (1999)
9. Dias Neto, A.C., Subramanyan, R., Vieira, M., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. In: Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007, pp. 31–36 (2007)
10. Elnashar, A.: IoT evolution towards a super-connected world. arXiv preprint arXiv:1907.02589 (2019)
11. Ergen, S.C.: ZigBee/IEEE 802.15.4 summary. UC Berkeley, 10 September 2004
12. Frantzen, L., Tretmans, J., Willemse, T.A.C.: Test generation based on symbolic specifications. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 1–15. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31848-4_1
13. Frantzen, L., Tretmans, J., Willemse, T.A.C.: A symbolic framework for model-based testing. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES/RV-2006. LNCS, vol. 4262, pp. 40–54. Springer, Heidelberg (2006). https://doi.org/10.1007/11940197_3
14. Huang, A.S., Rudolph, L.: Bluetooth Essentials for Programmers. Cambridge University Press, Cambridge (2007)

15. Hwang, J., Aziz, A., Sung, N., Ahmad, A., Le Gall, F., Song, J.: AUTOCON-IoT: automated and scalable online conformance testing for IoT applications. IEEE Access **8**, 43111–43121 (2020)
16. Incki, K., Ari, I.: Observing interoperability of IoT systems through model-based testing. In: Fortino, G., et al. (eds.) InterIoT/SaSeIoT -2017. LNICST, vol. 242, pp. 60–66. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93797-7_8
17. Statista Inc.: Internet of things (IoT) active device connections installed base worldwide from 2015 to 2025* (2020). https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
18. Janssen, S.: Transforming source code into symbolic transition systems for practical model-based testing (2017)
19. Kim, H., et al.: IoT-TaaS: towards a prospective IoT testing framework. IEEE Access **6**, 15480–15493 (2018)
20. Koopman, P., Alimarine, A., Tretmans, J., Plasmeijer, R.: Gast: generic automated software testing. In: Peña, R., Arts, T. (eds.) IFL 2002. LNCS, vol. 2670, pp. 84–100. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44854-3_6
21. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 109–126. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24732-6_8
22. Malik, B.H., et al.: IoT testing-as-a-service: a new dimension of automation. Int. J. Adv. Comput. Sci. Appl. **10**(5) (2019)
23. Marinissen, E.J., et al.: IoT: source of test challenges. In: 2016 21th IEEE European Test Symposium (ETS), pp. 1–10. IEEE (2016)
24. Pretschner, A.: Model-based testing in practice. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 537–541. Springer, Heidelberg (2005). https://doi.org/10.1007/11526841_37
25. Pretschner, A., et al.: One evaluation of model-based testing and its automation. In: Proceedings of the 27th International Conference on Software Engineering, pp. 392–401 (2005)
26. Saleem, J., Hammoudeh, M., Raza, U., Adebisi, B., Ande, R.: IoT standardisation: challenges, perspectives and solution. In: Proceedings of the 2nd International Conference on Future Networks and Distributed Systems, pp. 1–9 (2018)
27. Schieferdecker, I.: Model-based testing. IEEE Softw. **29**(1), 14 (2012)
28. Taivalsaari, A., Mikkonen, T.: A roadmap to the programmable world: software challenges in the IoT era. IEEE Softw. **34**(1), 72–80 (2017)
29. Tappler, M., Aichernig, B.K., Bloem, R.: Model-based testing IoT communication via active automata learning. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST), pp. 276–287. IEEE (2017)
30. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78917-8_1
31. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **22**(5), 297–312 (2012)
32. Vorakulpipat, C., Rattanalerdnusorn, E., Thaenkaew, P., Hai, H.D.: Recent challenges, trends, and concerns related to IoT security: an evolutionary study. In: 2018 20th International Conference on Advanced Communication Technology (ICACT), pp. 405–410. IEEE (2018)
33. Xia, F., Yang, L.T., Wang, L., Vinel, A.: Internet of things. Int. J. Commun. Syst. **25**(9), 1101 (2012)

34. Yoneyama, J., Artho, C., Tanabe, Y., Hagiya, M.: Model-based network fault injection for IoT protocols. In: Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 201–209. SCITEPRESS-Science and Technology Publications, Lda (2019)
35. Ziegler, S., Fdida, S., Viho, C., Watteyne, T.: F-interop – online platform of interoperability and performance tests for the internet of things. In: Mitton, N., Chaouchi, H., Noel, T., Watteyne, T., Gabillon, A., Capolsini, P. (eds.) InterIoT/SaSeIoT - 2016. LNICST, vol. 190, pp. 49–55. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52727-7_7