Building blocks for IoT testing: a benchmark of IoT apps and a functional testing framework

Conference Paper · February 2023

DOI: 10.1145/3528227.3528568

3 authors, including:

Rareș Cristea
University of Bucharest
7 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Ciprian Paduraru
University of Bucharest
50 PUBLICATIONS **157** CITATIONS
SEE PROFILE

Building blocks for IoT testing - a benchmark of IoT apps and a functional testing framework

Rareș Cristea
rares.cristea@unibuc.ro
University of Bucharest
București, Romania

Mihail Feraru
mihail.feraru@s.unibuc.ro
University of Bucharest
București, Romania

Ciprian Paduraru
ciprian.paduraru@fmi.unibuc.ro
University of Bucharest
București, Romania

ABSTRACT

IoT security is a topic that offers numerous opportunities for improvement and development. In this paper, we first present a set of open-source mock IoT applications along with the necessary infrastructure specifically designed to emulate a real IoT system. With our app set, users can add their own applications, automation rules, and communication flows with little technical effort, and test different scenarios to reproduce bugs that are not specific to the use of a single device. Second, we describe a functional testing framework for the IoT that is inspired by behavior-driven development (BDD), a testing methodology that serves as a proof-of-concept for how the application set can be used in different test scenarios. The application set and the functional testing framework are independent of each other. Our goal is to help IoT developers and testers find new testing techniques and benchmarking them in a reproducible, comparable, and less biased environment. We believe that they form the basis for a better understanding of how to test systems composed of heterogeneous devices to find issues and vulnerabilities that arise mainly from their interaction and data persistence management.

ACM Reference Format:
Rareș Cristea, Mihail Feraru, and Ciprian Paduraru. 2022. Building blocks for IoT testing - a benchmark of IoT apps and a functional testing framework. In *4th International Workshop on Software Engineering Research and Practice for the IoT (SERP4IoT'22), May 19, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528227.3528568

1 INTRODUCTION

The Internet of Things (IoT) is an ever-growing field today. Examples such as smart cities connecting transportation networks, intelligent car systems, and smart home systems are more present in our lives than ever. Sensors, actuators, end-user applications, gateways, and servers are interconnected through various communication protocols and methods without us even realising it. It is common for an IoT system to be composed of devices from different vendors, resulting in a unique combination of technologies that interact in unpredictable ways. However, as one would expect, the rapid development and deployment cycles lead to major challenges, including security and privacy issues. Problems can occur at multiple levels, from single isolated applications to networked application streams exposed to various risks, such as Distributed Denial-of-Service (DDoS) [1] , identity management and user security issues [20], etc. After reviewing the literature and finding that there is a lack of common ground for evaluating and researching testing approaches, we add the following contributions in this paper from the authors' knowledge:

- The first open-source set of IoT-specific software applications that serves as a benchmark for the community to test different testing methods and compare them efficiently by providing a reproducible environment, a less biased comparison of results, and a simpler process for evaluating the effectiveness of a particular tool or technique. In addition to this suite of applications, we also provide tools to automate the development process and communication infrastructure so that users can focus on the methods and algorithms rather than the technical work. We hope that facilitating and standardizing these processes will increase interest in and accelerate the advancement of testing methods in the IoT space.
- The first open source framework that serves as a basic method for functional testing in the IoT domain. We consider that the proposed system can be easily extended by the user, with the possibility of adding their own set of applications and customized communication flows between them. We also provide the tools needed to automate issue detection. In our framework, the concept of *issue* can be understood as a template. Examples include exploit detection in the security domain, performance issues, or classic source code bugs. The user of the framework can be either a software developer or a quality assurance professional (tester).

As a side story, the applications in our application set were initially developed as part of an undergraduate course in Software Engineering at the University of Bucharest. The goal of the course is to teach students the general practices of software engineering in the field of IoT. The students have to define a free-form software development project, evaluating the topics presented in the lectures of the course: Application Analysis, Architecture, Security, Agile Practices, etc. In the 2020-21 edition of the course, students were required to create a software project that met the criteria of vision for our application set. As the literature suggests, providing students with a real-world use case for their work projects motivates them and leads to a better software development process and final product [15]. Since the applications are developed by multiple independent teams, they can better represent the IoT system scenario: Devices from different vendors are integrated into complex systems that interact in unpredictable ways and present new testing challenges. In the following Section, we review the existing literature on testing

SERP4IoT'22, May 19, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9332-4/22/05…$15.00
https://doi.org/10.1145/3528227.3528568

heterogeneous IoT systems, current challenges, proposed solutions, and arguments that support the topic of our paper. A technical description of our initial set of applications, test infrastructure, and user guidance on extending the existing set with new or different applications is presented in Section 3. Then, Section 4 describes the testing framework we propose for functional testing in the IoT domain. Evaluation of our approaches is presented in Section 5, along with a discussion of their strengths and weaknesses. Finally, in Section 6 Conclusions and future work are presented.

2 RELATED WORK

In [19], the authors noted that new methods of security assessment in IoT could benefit from the existence of a set of applications on which to test new tools. The goal is to provide a benchmark that can be used to detect known and new bugs and compare the performance of the detection with other similar tools. A similar tool is FuzzBench [16], which is "on its way to becoming a standard benchmarking platform for fuzzers." Fuzzbench uses the OSS-Fuzz applications as targets against which it compares the fuzzing tool under test. Due to the heterogeneity of the IoT landscape, to the authors' knowledge, there is no repository of IoT applications (open source or not).

IoT systems involve interactions between multiple communication protocols, devices, and software from different vendors. We accept that such systems are highly heterogeneous, a feature that increases testing complexity. In their meta-analysis, [10] concludes that the majority of industry testers of heterogeneous systems prefer manual exploratory testing, which is highly dependent on the tester's skills and can hardly be evaluated objectively.

The lack of methodology and automated tools for testing heterogeneous IoT systems is reinforced by [7]. Their analysis of available tools and platforms for IoT testing shows that most of these tools are tied to a specific vendor and support only a limited number of devices or protocols. In addition, almost all of the tools use a simulated environment rather than an emulated or real-world environment, making testing easier to set up and less expensive, but reducing the accuracy of the process. Other tools are only available on remote test runners, which impacts data protection and the ability to test certain custom conditions.

The need for a common set of applications that serve as benchmarks and playgrounds for testing techniques is emerging in other areas as well. For example, [9] highlights that almost every paper uses a different set of web applications in its analysis. This affects the reliability of comparisons between approaches because the starting point is different. [9] concludes that "having a set of subject systems that every-one can use for rigorous controlled experimentation is needed" containing not just applications, but a complete architecture of software, test runners, and faulty data. FIRMCORN [11] has focused on streamlining the testing process by using a mix of emulating the firmware and using the real device. However, the devices used are only routers and webcams, and the testing methodology used only one device at a time, rather than a more realistic scenario - as part of a larger IoT system.

An approach to functional testing similar to our own has been explored by [4] in collaboration with an industry partner. Their framework, PatrIoT, aims to provide a foundation for building integration test environments for IoT systems that allow real hardware to be mixed with simulated devices. To benchmark their approach, they have developed a set of pilot applications that are not readily reproducible, highlighting the need for a common set of benchmarks for researchers.

Private companies are also interested in exploring the challenges of heterogeneous, interconnected devices. A study published by TrendMicro, [12], assesses the security risks of two smart homes they call "complex IoT environments." They show how networking multiple IoT devices and orchestrating them through an automation server can create a plethora of security risks, vulnerabilities or malfunctions.

3 APPLICATION SET, INFRASTRUCTURE FOR DEPLOYMENT AND TESTING

In our work, we first propose a set of applications that can be used to test and benchmark different testing methods in the IoT field community. In this Section, we briefly introduce these applications, the backend component implemented to facilitate deployment and testing, and an overview for users on how to set up their own test and application suite. We consider that the implemented infrastructure can be reused for the user's own suite with minimal effort.

3.1 Description of the current application set

The application set at the moment consists of five interconnected software applications that mimic the behaviour of smart devices, which we have connected through various examples of automation workflows. Our motivation for the smart home subset of systems comes from observing the rise of proprietary products such as Google Home Assistant, Amazon Alexa, or Facebook Portal, as well as open source solutions such as openHAB or Home Assistant. Also from our previous study, [19], we concluded that smart home IoT use cases could be a main target for a first set of applications [1].

Below is a brief description of each application:

(1) *SmartFlower* - A *smart pot* application designed to help the user manage the resources of a plant.
(2) *WindWow* - A smart home application designed to facilitate the automatic and distant use of a window in an indoor space.
(3) *Smarteeth* - An application running a *smart toothbrush* that helps users track information about the health of their mouth and use appropriate toothbrushing programs.
(4) *SmartKettle* - An *smart kettle* Application that speeds up and automates the preparation of drinks.
(5) *SmartTV* - An *smart TV* application where users can have individual profiles and, based on their behavior, a recommendation system is used to recommend movies or TV shows.

[1]The repository containing the applications and the Hub can be accesed at this link https://github.com/unibuc-cs/IoT-application-set.

3.2 Communication and deployment infrastructure

Although currently the five applications are focused on smart home systems, the technologies developed and the overall deployment infrastructure are independent of the type of applications. Along this application set, we have built the necessary infrastructure to allow users and the community to customize and scale the existing set, as well as deployment scripts to easily evaluate different testing methods and algorithms.

The infrastructure built is independent of the programming languages in which the applications are developed. The current corpus includes applications written in C/C++, as we have found this to be the most commonly used programming language for open source IoT projects [6]. Further planned work will extend the set to include other applications written in other languages, such as Python. Each of the current applications in our set models a smart home device. At the communication level, they all needed to implement an HTTP-based layer that allows sending and receiving messages between applications and from external users. HTTP was chosen as the communication protocol for some of the following reasons:

(1) It is widely used and a very popular choice for a variety of applications due to its simplicity and synchronous request-response pattern.
(2) Many existing open source tools have already been developed for it and provide good support and compatibility.
(3) The OpenAPI platform [2] provides a rigorous specification standard for device communication over HTTP.

We chose to use the OpenAPI specification to describe the communication interface for all of our applications, creating a common interface between them. Our experiments have shown that encapsulating exposed functionalities under a common and strict specification simplifies the testing process, even if they are not designed to be cohesive.

At the core of a number of IoT systems, such as in the smart home domain, is the communication network-style infrastructure on top of orchestration software deployed in a physical location, usually known as a *Hub Application*. In addition to developing the use cases in our application set, we also developed such a hub application that is responsible for defining and executing automation tasks between the software and the devices. We chose to write it in Python as this allows for rapid prototyping. Using OpenAPI Generator [3] we automatically generated the client code necessary to interact with the smart applications according to the HTTP specifications.

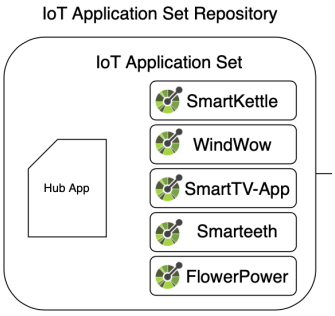

Figure 1: The figure describes the components of our application set and test infrastructure, i.e., the hub app and our initial set of IoT applications, each with an OpenAPI specification.

One drawback of the HTTP protocol, while widely used in IoT, is that it is not the most appropriate protocol in some use cases. A concrete example is applications that rely on intensive processing of asynchronous events and are deployed on devices with limited computational resources and capabilities [21], [17]. To address this issue, we plan to extend the testing framework to include MQTT communication, which is described in its own terms as "The Standard for IoT Messaging" [4]. We will leverage a similar initiative to OpenAPI, namely AsyncAPI [5], which aims to provide the same level of standardisation for event-driven systems typically built around publisher-subscriber protocols like MQTT.

3.3 User Guide to Testing, Adding New Applications, and Setting Up the Testing Infrastructure

When adding new applications to the corpus, the user can define new automation tasks in the hub app. A good practise for the automation tasks is to cover as many features of the new application as possible. Then our test framework can identify and cover these tasks and run in-depth tests on them. The current application repository is shown in Figure 1. During a test process, each application is deployed in a *Docker* container on the same network so that they can discover each other. Deployment configuration is automatically managed using the $docker - compose$ tool. The only requirements for adding a new application to the repository are:

(1) The app must contain an OpenAPI specification file.
(2) The app must be compatible with deployment inside a Docker container or at least added to a Docker network. [3]

The test framework can be set up by using the mentioned docker-compose configuration from the repository. For the interested user, we describe what happens behind the scenes in the backend. For example, the following steps are automatically executed by our system:

(1) Each *smartdevice* is deployed in a Docker container.

[2]OpenAPI specification can be consulted here: https://spec.openapis.org/oas/latest.html.
[3]OpenAPI Generator can be consulted here: https://github.com/OpenAPITools/openapi-generator.
[4]MQTT specification can be consulted here: https://mqtt.org/mqtt-specification/.
[5]AsyncAPI specification can be consulted here: https://www.asyncapi.com/docs/specifications/v2.0.0.

(2) The managing communication API between the applications is automatically built using their provided OpenAPI specification. In this step, each application becomes a *client* in the test infrastructure.
(3) The created clients are added to the Docker container of the hub application.
(4) The hub application starts executing automation rules and test cases based on the received events.

The resulting system architecture after the setup process is shown in Figure 2.

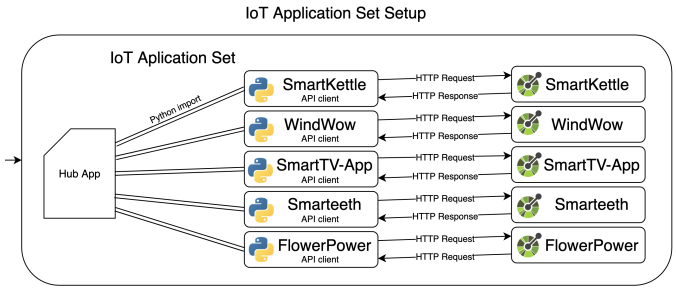

Figure 2: Application set architecture after the system is set up. API clients are generated, the hub application imports the clients and uses them to send HTTP requests to the applications.

3.4 Automation rules and bugs injection methodology

As we mentioned earlier, IoT systems consist of heterogeneous devices that interact with each other in complex patterns. To illustrate and analyse how the system can be tested and what challenges need to be overcome in different scenarios, we created a set of functional test cases on this infrastructure to evaluate the correctness of the communicating applications. We also added artificial bugs of several classes into the applications to provide a baseline for evaluating test methods and algorithms.

In our case study, we mimicked a smart home system where the communication flows between the participating applications are described by the commonly known concept of *automation rules*. The automation rules are stored in the hub application and can be triggered by one of the conditions listed below:

(1) A direct action manually requested by the user.
(2) An event generated and registered by a device.
(3) A scheduled event set by the user or application logic.

Based on our current application set, we have defined the following rules as a concrete example of possible automation processes:

(1) If the temperature of $WindWow$ is T degrees, then set the RPM of $SmartKettle$ to T.
(2) If the temperature of $FlowerPower$ is above 30 degrees, reduce the luminosity of $WindWow$ to half.
(3) If the luminosity of $WindWow$ is less than L, turn on the solar lamp of $FlowerPower$.
(4) The brightness of the $SmartTV$ is set relative to the luminosity provided by $WindWow$.
(5) The lower the temperature of the $WindWow$, the higher the temperature of the $SmartKettle$.

4 TESTING FRAMEWORK AND BASELINE METHODS

Unit testing the individual parts of a software system and ensuring that each part works correctly does not guarantee that the entire system will work correctly. By integrating multiple devices into complex logical flows and considering the persistence mechanisms behind the IoT software, it is expected that subtle interoperability and integration bugs will occur. Integration testing under the assumption of data persistence introduces new challenges, which we can divide into two perspectives:

- Testing whole multi-part systems is slower and additional technical work must be done to efficiently reproduce the reported issues.
- The issues are related to invalid states and business logic constraint violations rather than individual machine crashes, making them harder to detect. For example, in an industrial robotics pipeline, all the individual devices might be working correctly, but the orchestration logic might be causing malfunctions that damage the system, such as when two robot arms collide. We have seen an even more difficult situation with smart home systems, for example, where the end user can set their own automation rules. The end user then becomes the *programmer* of the system, but is generally unaware of the technical aspects of the system.

In [5], a survey was conducted with industry IoT solution providers and concluded that interoperability and integration are important concerns for developers. Given the above arguments and the lack of open source software addressing this issue, we believe that providing a functional method for testing integration processes is essential to advancing the current state of the art in IoT.

The implementation of our test framework suite is independent of the applications under test. In our case, we used our application suite, which consists of the applications defined in Section 3. Users of the framework can (and should) replace our applications with their own.

4.1 Functional Testing Framework

Functional testing was defined by [13] as an approach in which the design of a program is viewed as an integrated collection of functions. Although this testing method is probably one of the best known for testing an application, the unique nature of an IoT system is that the collection of functions is not part of the same software program and the success of a test case generally depends on functions from different interconnected devices. Functional testing may be the preferred testing approach for conformance testing. Our work aims to continue and improve [14] by extending conformance testing not only to a single IoT device, but to the entire IoT system. We also want to involve as many stakeholders involved in the development as possible [18]. Therefore, we chose to define test scenarios using Behave, a Python library for Behavior-Driven Development (BDD) that uses tests written in natural language [2].

BDD is a methodology that supports collaboration between business, development, and testing stakeholders using a common specification and approach to describe functionality. We found the principles of this methodology suitable for writing our test cases, as it communicates in a clear way the state of the system, the trigger conditions for applying various actions (e.g., the automation rules in our case), and the expected results after applying these actions. Overall, our experiments have resulted in facilitating the definition of test cases, interpretation, and discovery of problems in applications. Listing 1 describes several test cases for one of the above automation rules. To further understand the presented example, we briefly describe the structure of a test listing. The keyword *given* is used to set the state of the test. Triggers can be specified with the clause *When*, while expected results are marked with *Then*. All these clauses can be composed hierarchically using logical operators. In the given example, a table of values is also specified. Running the test means that the framework behind BDD takes each line of parameters and replaces them in the test specification. Thus, a test written as in Listing 1 is more like a template specification with parameters that are later defined by a requirements table, i.e., another functional test specified by each row in the table.

```
Feature: TV auto-brightness
  Scenario Outline: TV brightness is set based on window
     luminosity level
    Given TV brightness is set to <X>, window luminosity
     to <Y> and base luminosity to <Z>
      When automation rules are triggered for TV
      Then TV brightness is set to max(10 - <Y>/10 + <Z>,
     <Z>) <= 10
     Examples:
      | X  | Y  | Z  |
      | 3  | 20 | 1  |
      | 4  | 70 | 0  |
      | 1  | 0  | 10 |
```

Listing 1: An example of a template test and multiple functional test cases for the brightness automation rule 4 presented in 3.4.

The expected test suite in the IoT domain could involve either individual components or complex interaction flows and data exchange between multiple applications. In our environment, we allow users to easily specify their own test case data for scenarios related to their applications and flows under test using BDD methodology and syntax. In our current version of the framework, the following steps must be followed to create a new test:

(1) An existing automation task must be identified or a new one created in the hub application. In general, most of the interest in IoT is in testing flows that connect different endpoints of different applications. This is an important confirmation of the need for an orchestration component such as the Hub application.
(2) Add a new test using BDD syntax similar to the test shown in Listing 1. Adding a test is a two-step process. First, you need to specify the description of the test and optionally a table of individual test cases (as in our example). Then, you need to write a source code script that implements the additional operations defined in the test. Such code could, for example, define data structures for data persistence between test cases, communication flows, or new orchestration code on the existing applications.

The architectural connections between the applications, the hub application, the test case templates, and the concrete scenarios (i.e., the concrete table containing the parameters for testing the test case templates) are shown in Figures 3 and 4. An example of an application flow is shown in Figure 5. Two of the rules we designed in the hub app can interact (one triggers the other). When rule 2 is activated, it checks (by calling one of the apps) to get the temperature. When the hub receives this data, it stores it so that it can be used by another rule. According to rule 2, if the temperature provided by *FlowerPower* is above 30 degrees, then *WindWow* luminosity is reduced by half. The luminosity is also a value that is tracked by the hub. It also stores it and is immediately used by rule 3. When the luminosity is below 30, the plant lamp is turned on by *FlowerPower*.

5 EVALUATION

To provide a basis for problem discovery, we have introduced intentional problems in the smart applications and automation rules. At this point, it is important to note that the applications already contained a significant number of real issues identified by our framework. In what follows, we categorise the issues based on the level of interaction at which they occur, rather than the technical or logical cause. This decision illustrates our focus on identifying problems that arise from the integration and interoperability of different applications, rather than individual isolated software components for which there are a plethora of solutions in the literature. We consider that there are three categories of such issues:

(1) Application-level issues - malfunctions that result in invalid responses or crashing of a single, individual application.
(2) Rule-level issues - violations of the expected behaviour of a single automation rule that connects one or more applications.
(3) Persistence-level issues - violations of the expected behaviour after applying multiple rules in succession. These are usually fixed by the persistence management components of the applications.

As mentioned in the study by [22], an important class of issues are related to the triggering of unexpected actions in the flow of automated, interconnected applications. These bugs are the ones we consider at the *Rule level* and *Persistence level* we define, and are considered the most difficult to detect because they are generally a specific type of concurrency bug. The work in [22] analyzes the root cause of this category of issues and concludes that the main causes are:

(1) race conditions between rules.
(2) events missing or received in an unexpected order.
(3) duplicate or conflicting actions.

In Table 1 we have listed examples of real or artificially introduced issues in our application, categorized by the above levels. They are designed to cover all scenarios described by [22], linking application malfunctions and crashes to higher logic bugs.

Our functional testing approach is able to formulate faulty test cases for all these scenarios (see the source code repository for

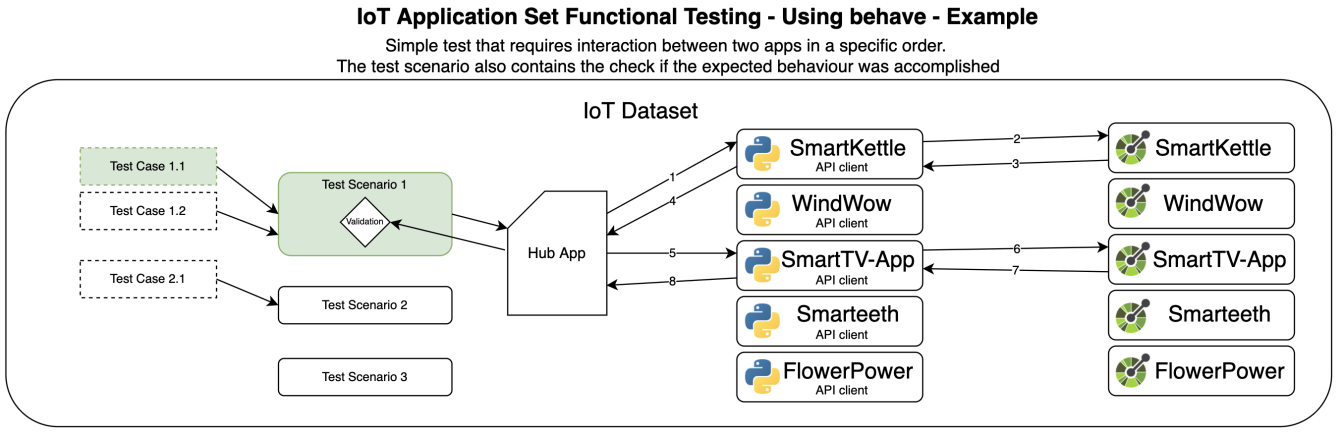

Figure 3: Application set architecture after setting up the system and defining the test cases. Upon execution, the Test Scenario, using the selected Test Case is run on the hub app. The hub app manages the requests and responses, and after the test is complete, compares the result value to the expected value of the test case.

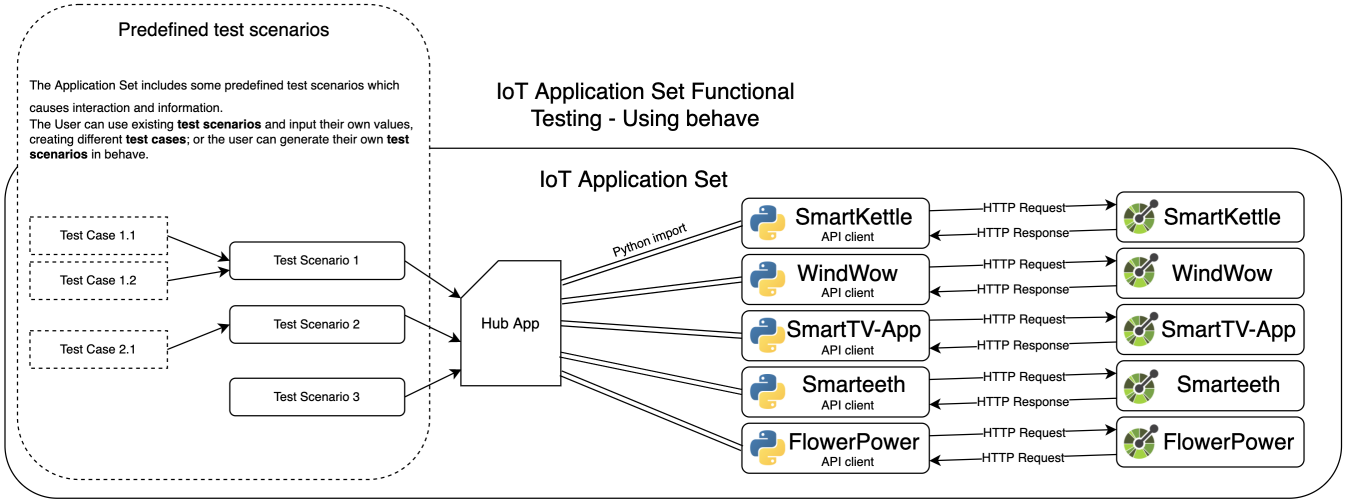

Figure 4: A comprehensive description of the Functional Testing framework. The figure describes how the framework is applied on the application set: from describing the Test Cases and Test Scenarios, to the communication with the applications.

examples of tests). It can guarantee that the system works correctly in a handful of scenarios and provide a basis for regression testing to ensure that future changes do not break the system. The application set contains 10 test scenarios that trigger various automation tasks in the hub application. Each automation requires an interaction between at least two applications, with one application generating data that is passed to another application to use.

On the other hand, the exploratory component of our approach is manual, as is typical of functional testing methods in general. Therefore, the detection of bugs is limited to the skills and time of the human tester or developer writing the functional tests. In general, concurrency bugs have a low reproducibility rate due to the order of the occurring events. Thus, we consider this a limitation from our existing framework and consider that more efforts must be spent in future work to address the the problem of detecting and reproducing concurrency issues.

6 CONCLUSION AND FUTURE WORK

In this work, we have first presented a small set of applications along with the necessary infrastructure to deploy and evaluate different methods and algorithms in the field of IoT testing. Then, we presented a framework independent of our application set that allows users to perform various functional tests for single or a set of interconnected application streams. We hope that the community

Examples of issues discovered	Application(s)	Rule(s)
Flowerpower: does not check for optional key existence in JSON object on PUT /settings	FlowerPower	-
TV brightness should be set to a maximum of 10, but the value is not validated by the app.	SmartTV	Rule 4
Rule 2 will reduce the window's luminosity if the temperature is over 30 degrees, then Rule 3 will unnecessarily turn on the lamp because the luminosity is too low.	FlowerPower, WindWow	Rule 2, Rule 3
Windwow crashes when trying to set luminosity to 25 and curtains are closed on GET /settings/{settingName}/{settingValue} (artificial bug)	WindWow	-
SmartKettle's temperature decreases for WindWow's temperatures under 0 degrees celsius instead of increasing	SmartKettle, WindWow	Rule 5
Smarteeth: "localhost" set as the hostname of the listening server thus refusing outside connections	SmartTeeth	
In FlowerPower: activateSolarLamp does not change luminosity	FlowerPower	

Table 1: Table presenting some examples of issues that have been discovered in our application set. Each issue discovered was triggered by one or multiple applications communicating. Some of the issues were discovered as part of an automatizing rule defined in our Hub Application.

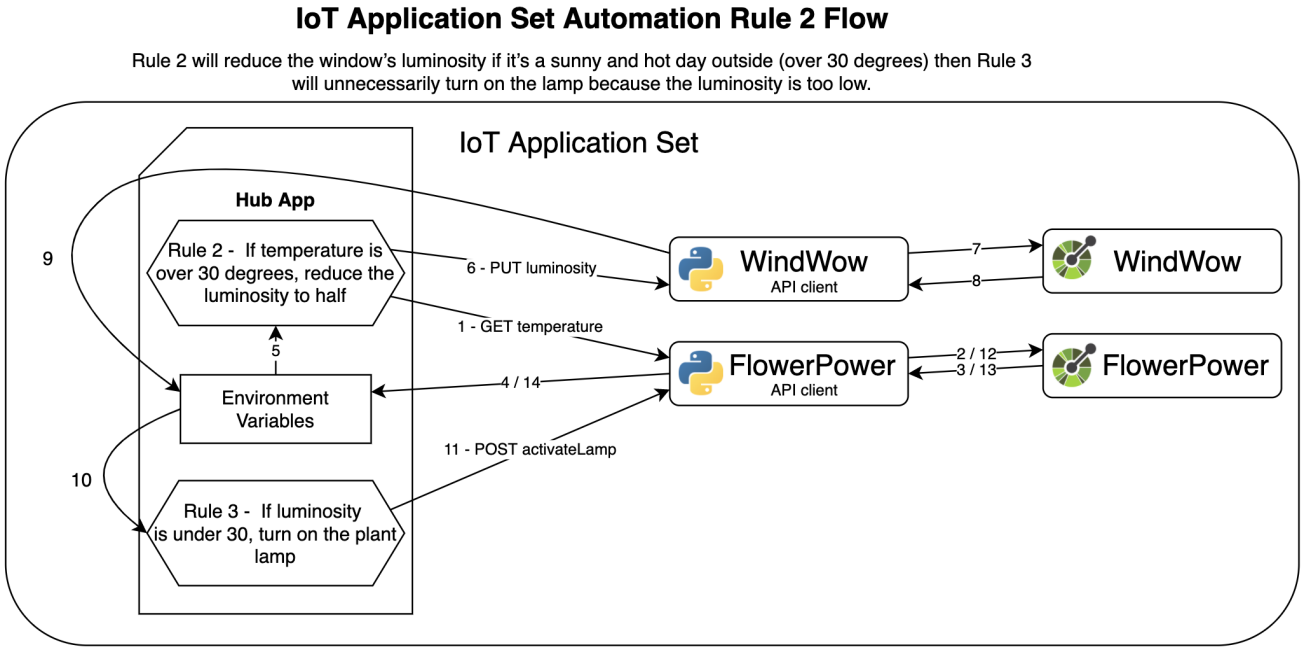

Figure 5: Flow of a multi-rule level automation (i.e., persistence management issues testing). *1,2,3,4* - The request response is sent from the Hub by rule 2, and the response is saved in the Hub's Environment Variables. *5* - Depending on the response value, Rule 2 can be continued, therefore *6,7,8,9* makes the request to WindWow, and stores luminosity in Hub's Environment Variables. If luminosity is under 30, according to Rule 3, the plant lamp is activated in steps *11,12,13,14*

developing or testing IoT applications will benefit from our foundational open-source software to experiment more quickly with new methods and increase the stability, security, and performance of applications in this domain.

The threats to the validity of our approach can be summarised in two points: First, creating a set of mock applications may not simulate the full range of common vulnerabilities in real applications. Adding some real applications to the application set could mitigate this threat. Second, a more comprehensive approach would also mean running the applications on real hardware rather than virtual machines. While this is beyond the scope of this article, it would increase the attack space and could lead to more interesting results. One of our work-in-progress plans, is to extend the application set to a variety of IoT software and communication protocols such as MQTT or ZigBee [8]. We also plan to integrate Hub software applications used by the general public, such as openHAB or Home Assistant. We could then provide a larger collection

of challenges. The current work can also be continued by experimenting with other types of testing beyond the functional testing method proposed in this paper. The practice of testing systems using fuzz testing is on the rise, and various fuzzers, both white-box and black-box based methods, can be added to expand the scope of the test suite of our current baseline methods. Fuzzing has a large exploratory aspect that contrasts with the limited ability of functional testing to examine application states in depth. Running tests on emulated or real-world devices could create new vulnerabilities for the deployed software or hardware of the system under test. Work is currently underway to extend our application to typical embedded devices on the market. Since we have not insisted on a concrete methodology for comparing research results using our application set, further work could be to develop a rigorous set of procedures for benchmarking test techniques and tools.

7 ACKNOWLEDGEMENTS

We thank the final year Informatics undergraduate students, 2020 Promotion of the Computer Science and Mathematics Faculty of the University of Bucharest that created the applications that are now part of the applications set.

This research was supported by the European Regional Development Fund, Competitiveness Operational Program 2014-2020 through project IDBC (code SMIS 2014+: 121512).

REFERENCES

[1] Yahya Al-Hadhrami and Farookh Khadeer Hussain. 2021. DDoS attacks in IoT networks: a comprehensive systematic literature review. *World Wide Web* 24, 3 (01 May 2021), 971–1001. https://doi.org/10.1007/s11280-020-00855-2

[2] Richard Jones Benno Rice and Jens Engel Revision. 2022. Behave. https://behave.readthedocs.io/en/stable/index.html. Accessed: 2022-01-10.

[3] Carl Boettiger. 2015. An Introduction to Docker for Reproducible Research. *SIGOPS Oper. Syst. Rev.* 49, 1 (jan 2015), 71–79. https://doi.org/10.1145/2723872.2723882

[4] Miroslav Bures, Bestoun S. Ahmed, Vaclav Rechtberger, Matej Klima, Michal Trnka, Miroslav Jaros, Xavier Bellekens, Dani Almog, and Pavel Herout. 2021. PatrIoT: IoT Automated Interoperability and Integration Testing Framework. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE. https://doi.org/10.1109/icst49551.2021.00059

[5] Miroslav Bures, Matej Klima, Vaclav Rechtberger, Xavier Bellekens, Christos Tachtatzis, Robert Atkinson, and Bestoun S. Ahmed. 2020. Interoperability and Integration Testing Methods for IoT Systems: A Systematic Mapping Study. In *Software Engineering and Formal Methods*. Springer International Publishing, 93–112. https://doi.org/10.1007/978-3-030-58768-0_6

[6] Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. 2020. How is Open Source Software Development Different in Popular IoT Projects? *IEEE Access* 8 (2020), 28337–28348. https://doi.org/10.1109/ACCESS.2020.2972364

[7] Joao Pedro Dias, Flavio Couto, Ana C.R. Paiva, and Hugo Sereno Ferreira. 2018. A Brief Overview of Existing Tools for Testing the Internet-of-Things. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. https://doi.org/10.1109/icstw.2018.00035

[8] Sinem Coleri Ergen. 2004. ZigBee/IEEE 802.15. 4 Summary. *UC Berkeley, September* 10, 17 (2004), 11.

[9] Vahid Garousi, Ali Mesbah, Aysu Betin-Can, and Shabnam Mirshokraie. 2013. A systematic mapping study of web application testing. *Information and Software Technology* 55, 8 (Aug. 2013), 1374–1396. https://doi.org/10.1016/j.infsof.2013.02.006

[10] Ahmad Nauman Ghazi, Kai Petersen, and Jürgen Börstler. 2015. Heterogeneous Systems Testing Techniques: An Exploratory Survey. In *Lecture Notes in Business Information Processing*. Springer International Publishing, 67–85. https://doi.org/10.1007/978-3-319-13251-8_5

[11] Zhijie Gui, Hui Shu, Fei Kang, and Xiaobing Xiong. 2020. FIRMCORN: Vulnerability-Oriented Fuzzing of IoT Firmware via Optimized Virtual Execution. *IEEE Access* 8 (2020), 29826–29841. https://doi.org/10.1109/ACCESS.2020.2973043

[12] Stephen Hilt, Numaan Huq, Martin Rösler, and Akira Urano. 2017. Cybersecurity Risks in Complex IoT Environments: Threats to Smart Homes, Buildings and Other Structures. (2017). Accessed: 2022-01-10.

[13] William E Howden. 1980. Functional program testing. *IEEE Transactions on Software Engineering* SE-6, 2 (1980), 162–169.

[14] Hiun Kim, Abbas Ahmad, Jaeyoung Hwang, Hamza Baqa, Franck Le Gall, Miguel Angel Reina Ortega, and JaeSeung Song. 2018. IoT-TaaS: Towards a Prospective IoT Testing Framework. *IEEE Access* 6 (2018), 15480–15493. https://doi.org/10.1109/ACCESS.2018.2802489

[15] Chang Liu. 2005. Enriching software engineering courses with service-learning projects and the open-source approach. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* 613–614. https://doi.org/10.1109/ICSE.2005.1553612

[16] Jonathan Metzman, László Szekeres, Laurent Simon, Read Sprabery, and Abhishek Arya. 2021. FuzzBench: An Open Fuzzer Benchmarking Platform and Service. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1393–1403. https://doi.org/10.1145/3468264.3473932

[17] Nitin Naik. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 1–7.

[18] Anton Okolnychyi and Konrad Fögen. 2016. A study of tools for behavior-driven development. *Full-scale Software Engineering/Current Trends in Release Engineering* (2016), 7.

[19] Ciprian Păduraru, Rareș Cristea, and Eduard Stăniloiu. 2021. RiverIoT - a framework proposal for fuzzing IoT applications. *International Conference on Software Engineering ICSE 2021, Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT)* (2021), 52–58.

[20] Kazi Masum Sadique, Rahim Rahmani, and Paul Johannesson. 2020. IMSC-EIoTD: Identity Management and Secure Communication for Edge IoT Devices. *Sensors (Basel, Switzerland)* 20, 22 (16 Nov 2020), 6546. https://doi.org/10.3390/s20226546 33207820[pmid].

[21] Tetsuya Yokotani and Yuya Sasaki. 2016. Comparison with HTTP and MQTT on required network resources for IoT. In *2016 international conference on control, electronics, renewable energy and communications (ICCEREC)*. IEEE, 1–6.

[22] Wei Zhou, Chen Cao, Dongdong Huo, Kai Cheng, Lan Zhang, Le Guan, Tao Liu, Yan Jia, Yaowen Zheng, Yuqing Zhang, Limin Sun, Yazhe Wang, and Peng Liu. 2021. Reviewing IoT Security via Logic Bugs in IoT Platforms and Systems. *IEEE Internet of Things Journal* 8, 14 (July 2021), 11621–11639. https://doi.org/10.1109/jiot.2021.3059457