Fault Injection in the Internet of Things Applications

Mohammad Amin Alipour
University of Houston
Houston, Texas, USA
alipour@cs.uh.edu

ABSTRACT

Internet of Things comprises a large proportion of cyber-physical systems where a group of interconnected sensors and actuators, usually with cloud backends, are used to perform a task. Vendors of Internet of Things devices provide programming frameworks to help users—usually with little knowledge of embedded or distributed systems—to develop their applications. Most of these frameworks provide an event-based abstraction of the underlying cyber-physical systems.

In this paper, we propose a preliminary set of faults for fault injection in event-based Internet of Things as a part of our ongoing investigation into the testing of Internet of Things applications. These faults intend to enhance the awareness of the programmers about the situations that might arise in the wild.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; • **Computer systems organization → Embedded systems**;

KEYWORDS

Fault Injection, Internet of Things, Smart Home

ACM Reference format:
Mohammad Amin Alipour. 2017. Fault Injection in the Internet of Things Applications. In *Proceedings of 2017 ACM International Workshop on Testing Embedded and Cyber-Physical Systems, Santa Barbara, CA, USA, July 13, 2017 (TECPS'17),* 3 pages.
https://doi.org/10.1145/3107091.3107095

1 INTRODUCTION

Internet of Thing systems are sets of interconnected, cyber-physical systems (usually small, inexpensive sensors and actuators). These systems are typically connected to a service on the Internet for sharing information with other services, configuration management systems, or monitoring systems. The interplay between several devices on the local network and the Internet has sparked the creation of innovative solutions to automate and improve many household chores, such as saving energy by adjusting the temperature of rooms based on the location of residents. McKinsey predicted that more than 30 billion Internet of Things devices would be in use by 2020, and between 15% and 20% growth annually [1]. At this rate, Internet of Things is becoming an important tool in the daily life of people.

To fulfill the increasing demand for IoT systems, several vendors offer IoT kits that allow users to define the behavior of devices as they desire. These DIY (Do It Yourself) kits are supported by programming frameworks that enable users to program the devices, connect them together, and aggregate data of the system components. For example, users can define when certain sensors should be read and the actions that should be taken accordingly.

To simplify the development of these systems, vendors and several open source organizations provide high-level abstractions of the systems for the programmer. The common abstraction available in the frameworks is event-based abstraction [3]. That is, users only implement modules and functions that respond to individual events, such as switch-on or switch-off events, or change in the room temperature.

While these DIY kits accelerate the innovation of new IoT systems and make them more accessible to the public, they present new challenges in safety and security. The users of theses systems are not necessarily well aware of problematic situations that may arise in the distributed, embedded systems, which can leave the systems they implement vulnerable and unsafe.

In this paper, we propose three faults for event-based IoT systems that emulate some of the potentially dangerous conditions that can happen in the environment. These faults are: event loss, event duplication, and delay. The goal of these faults and the fault injection systems that supports them is to enhance the user's understanding of the situation that might arise in the wild. Finally, we report how we plan to use Samsung's SmartThings as our subject for fault injection. SmartThings is an IoT framework to connect, configure, and program a group of compatible devices that aim for home automation. It provides an event-based framework along with a web IDE for programming the IoT applications.

The rest of this paper is organized as follows: Section 2 provides preliminary concepts, Section 3 introduces the faults, finally, Section 4 describes future work.

2 PRELIMINARY CONCEPTS

Fault Injection. Fault injection tools emulate potential problems in the environment that a system will operate within [4], by injecting an error state into the running program.

Broadly speaking, there are two primary fault injection techniques: hardware-based fault injection and software-based fault injection. In the first, additional hardware is used to inject faults. This hardware introduces an error via contact with a target circuit by changing voltage or current or some other physical phenomena (such as radiation) to introduce a fault.

TECPS'17, July 13, 2017, Santa Barbara, CA, USA

ACM ISBN 978-1-4503-5112-6/17/07…$15.00
https://doi.org/10.1145/3107091.3107095

Software-based fault injection approaches use code instrumentation or runtime injection. In the code instrumentation approach, additional code is inserted into the source code to emulate faults. In runtime injection, events in the computational environment such as interrupts and timeouts are used to introduce the fault.

Abstraction. We describe an event-based abstraction of IoT systems.In this abstraction, modules subscribe to events, which they are usually from sensors or the cloud back-end. Depending on the type and value of events a device decides to perform a computation.

More formally, events of type **Event**, are in charge of changing the state of the system, represented by a type **State**. **State** abstracts the state of the IoT system which includes the state of devices, the network topology, and connectivity of devices. **Event** abstracts the events that a developer intends to monitor and act based upon. We expand the notion of events to include the change in the configuration of the system, for example, the addition of a new device or a new network SSID to the system is an event.Since the age and type of a sensor device can predict the accuracy of the device, we associate each component of the system with an age and include it in the state of the system. We also include events that change the age of system components.

Let **execute** be a state transition function that takes a **State** and a **Event**s and computes the next state of the system. We can define **multiStep** for multiple-step state transition that takes a **State** and a list of **Event**s and computes the next state by sequential application of events on the state of the system, described below using Haskell programming language notation,

```
multiStep :: State -> [Event] -> State
multiStep d [] = d
multiStep d hd:: tl = multiStep (execute d hd) tl
```

System Overview for Samsung SmartThings. We are developing a fault injection framework for Samsung's SmartThings application. SmartThings's main hardware is a hub that all compatible devices such as Nest[1] and smart lightbulbs can connect to. SmartThings comes with a WebIDE to unify all the data from the devices and manage them. The WebIDE allows users to use a subset of the Groovy programming language to code for handling individual events. It also provides a simulator to test the systems in reaction to different events.

For each program, the simulator draws the devices and the corresponding events on an interactive webpage (for example, Figure 1). Accessible events in these systems can be represented as events in the interaction with this webpage. We use Selenium scripts to represent test cases as series of interactions with the the systems and each interaction triggers an event. To implement the faults described in this paper, we modify the Selenium scripts to emulate the faults described in this paper.

3 FAULTS

Loss of an event. An event can be lost due to errors in the sensor, connectivity, and/or power loss. To emulate this situation, we suggest the event loss fault to drop an event from the list of events. Suppose `evList = [ev1 ev2 ... evN]` is a list of events in the event queue. This fault ignores some of events in the `evList`. This

[1]http://www.nest.com

Figure 1: SmartThings simulator shows state of devices in the system (here a motion detector and a smart switch).

is achieved by skipping a random step in the Selenium script. The snippet below illustrates the semantic impact of this fault, where `ev2` has been dropped from the event list.

```
multiStep state [ev1 ev2]   -- Original
multiStep state [ev1]       -- Fault
```

In case of SmartThings, it would require removing some of the interactions with the WebIDE simulator in the Selenium script.

Event Duplication. An event can be duplicated in Internet of Things due to some faults in the sensors or switches. It is important for the developers to evaluate the impact of this duplication on their system. This fault duplicates an event, as illustrated in the following snippet.

```
multiStep state [ev1]     -- Original
multiStep state [ev1 ev1] -- Fault
```

In case of SmartThings, it would require duplicating an interaction with the WebIDE simulator in the Selenium script.

Delay Event. This fault inserts a pause before the execution of an event handler. This emulates the slow connections and hardware. The following code snippet shows the semantic impact of this fault, where `NOP` denotes a dummy event that induces a delay.

```
multiStep state [ev]      -- Original
multiStep state [NOP, ev] -- Fault
```

In case of SmartThings, it would include adding a `sleep` command before one of the interactions with the WebIDE in the Selenium script.

4 DISCUSSION AND FUTURE WORK

Techniques such as fault injection can benefit programmers to explore the different or sometimes unwanted behavior in the systems. It also can help us to understand potential faults in cyber-physical system and ways to analyze them. In this section, we describe limitations of the approach and future work.

Fault injection, as described, can be expensive in systems with many possible events. In this effort, we use fault injection as a means to understand and model faults in programming for event-based cyber-physical systems. This technique might not scale well to large networked cyber-physical systems with diverse types of devices, but at least we hope that it helps provide greater understanding of such system.

Another problem with this approach is the rhe relationship between the behavior of the simulation and the behavior of the real system. When developers create simulations, they are abstracting and simplifying the behavior of the devices. Thus, simulators might not be a faithful representation of the real system. As the behaviors of the simulator and real system diverge, the need to run tests and inject the faults on the real hardware systems increases.

In the future, we plan to automate the test generation for SmartThings systems. Although it is easy to generate a sequence of events by extracting web elements in the simulator,the challenge lies in designing the oracle. An approach could be asking users to write general oracles similar to properties in QuickCheck [2], and the tests can be checked against those properties. For example, if the effect of events are communitive, a property similar to `communitive` in the snippet below, where `permutation` is a function that permutes a list of events, can be used to generate and evaluate many test cases:

```
communitive :: multiStep evList == multiStep permutation evList
```

Another approach, especially in case of SmartThings, can be to exploit the techniques in GUI/web testing to infer a state machine from interaction with the simulator and compare it with a state machine inferred from analysis of the event based systems. Unjustified differences between these two state machines can suggest bugs either in the system or in the simulator.

ACKNOWLEDGMENTS

Author would like to thank Rahul Gopinath, Nicholas Nelson, and Souti Chattopadhyay for their comments on the earlier drafts of this paper.

REFERENCES

[1] Harald Bauer, Mark Patel, and Jan Veira. 2014. The Internet of Things: Sizing up the opportunity. *McKinsey Company* (2014).

[2] Koen Claessen and John Hughes. 2011. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. *SIGPLAN Not.* 46, 4 (May 2011), 53–64. https://doi.org/10.1145/1988042.1988046

[3] Sebastian Frischbier, Erman Turan, Michael Gesmann, Alessandro Margara, David Eyers, Patrick Eugster, Peter Pietzuch, and Alejandro Buchmann. 2014. Effective Runtime Monitoring of Distributed Event-Based Enterprise Systems with ASIA. In *Proceedings of the 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications (SOCA '14)*. IEEE Computer Society, Washington, DC, USA, 41–48. https://doi.org/10.1109/SOCA.2014.25

[4] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. 1997. Fault injection techniques and tools. *Computer* 30, 4 (1997), 75–82.