

AUS Repository

A Formal Assisted Approach for Modeling and Testing Security Attacks in IoT Edge Devices

Item Type	Thesis
Authors	Bhanpurawala, Alifiya
Download date	2026-06-08 18:49:59
Link to Item	http://hdl.handle.net/11073/25489

A FORMAL ASSISTED APPROACH FOR MODELING AND
TESTING SECURITY ATTACKS IN
IOT EDGE DEVICES

by

Alifiya Bhanpurawala

A Thesis presented to the Faculty of the
American University of Sharjah
College of Engineering
In Partial Fulfilment
of the Requirements
for the Degree of

Master of Science in
Computer Engineering

Sharjah, United Arab Emirates

November 2023

Declaration of Authorship

I declare that this thesis is my own work and, to the best of my knowledge and belief, it does not contain material published or written by a third party, except where permission has been obtained and/or appropriately cited through full and accurate referencing.

Signed...Alifiya Bhanpurawala.....

Date.....04/12/2023.....

The Author controls copyright for this report.

Material should not be reused without the consent of the author. Due acknowledgement should be made where appropriate.

© Year 2023

Alifiya Bhanpurawala

ALL RIGHTS RESERVED

Approvals

We, the undersigned, approve the Master's Thesis written by Alifiya Bhanpurawala

Thesis Title: A Formal Assisted Approach for Modeling and Testing Security Attacks in IoT Edge Devices

Date of Defence: 20/11/2023

Name, Title and Affiliation	Signature
------------------------------------	------------------

Dr. Khaled El-Fakih Professor Department of Computer Science and Engineering Thesis Advisor	
--	--

Dr. Ra'afat Abu-Rukba Associate Professor Department of Computer Science and Engineering Thesis Examiner (Internal)	
--	--

Dr. Abdulrahim Shamayleh Associate Professor Department of Industrial Engineering Thesis Examiner (External)	
---	--

Accepted by:
Dr. Fadi Aloul
Dean
College of Engineering

Dr. Mohamed El-Tarhuni
Vice Provost for Research and Graduate Studies
Office of Research and Graduate Studies

Acknowledgements

I would like to thank my advisor Dr. Khaled El-Fakih for providing the motivation, guidance and knowledge throughout the research and implementation process. I would also like to thank Dr. Imran Zualkernan for guiding me throughout the process and providing deep insights on the subject. I am deeply grateful for their support and encouragement for the thesis.

I wanted to take a moment to express my heartfelt gratitude for your exceptional guidance and support during my time in the Computer Science and Engineering master's program. Your distinguished advice and encouragement have been invaluable to me, and I am grateful for the incredible level of education I have received under your tutelage.

Dedication

To my husband, son, parents and in-laws for their love and support...

Abstract

With the rapid growth in the number of IoT devices being added to the network, a major concern that arises is the security of these systems. As these devices are resource constrained, safety measures are difficult to implement on the edge. We propose a novel approach for the detection of IoT device attacks based on the use of formal modelling and mutation testing. Namely, we model the behaviour of small IoT devices such as motion sensors and RFID card reader as state machines with timeouts. We also model basic IoT attacks; namely, battery draining, sleep deprivation, data falsification, replay, and man in the middle attacks, as special mutants of these specifications. We also consider tests for detecting actual physical device manipulation. Mutation testing is then used to derive tests that distinguish these attacks from the original specifications. The behaviour of these mutants is tested in real environment by running the tests on the data collected while the edge device is still running. Our experiments show that derived number of attack mutants and tests is small and thus these tests can be executed many times with limited overhead on the physical device. Consequently, our approach is not deterred by related high costs of traditional mutation testing. Furthermore, we demonstrate that the tests generated by our method, encompassing the considered IoT attacks, do not adequately cover mutants derived through conventional mutation code-based operators. This highlights the necessity of employing our method. A framework that implements our approach is presented along with some other relevant case studies.

Keywords: Internet of things, edge devices, security threats, mutation testing, security faults, attacks, finite state machines with timeouts.

Contents

Abstract	6
List of Figures	9
List of Tables	10
List of Abbreviations	10
Chapter 1. Introduction	12
1.1. Problem Formulation	12
1.2. Overview	12
1.3. Thesis Objectives	14
1.4. Research Contribution	14
1.5. Thesis Organization	15
Chapter 2. Background and Literature Review	16
2.1. Internet of Things	16
2.2. Finite State Machines	17
2.3. Code and Model based Mutants.	21
2.3.1. Model based Mutants.	21
2.3.2. Code based mutants	22
2.4. Related Work	23
Chapter 3. Problem Statement and Methodology	30
3.1. Problem Statement	30
3.2. Problem Formulation	30
3.3. Modelling of IoT Devices as Timed Finite State Machines	31
3.3.1. Motion Sensor	31
3.3.2. Ultrasonic Motion Sensor	35
3.3.3. RFID Card Reader	38
3.4. IOT Attacks	44
3.4.1. Security Issues in IoT Devices	44
3.4.2. Attacks Modeled as state machine mutants	44
3.4.3. Physical Device Attacks	52
3.5. Proposed Testing Framework	53

3.5.1.	The Edge Device	54
3.5.2.	Data collection of the device behaviour	54
3.5.3.	Injecting the mutants into the microcontroller	55
3.5.4.	The Testing Framework	56
3.5.5.	Implementation tools	59
Chapter 4.	Evaluation	60
4.1.	Identify the coverage of security faults.	60
4.2.	Assess impact of threat in battery drainage	63
4.3.	Feasibility of applying the work in practice	64
4.4.	Applicability of the work to realistic systems	65
Chapter 5.	Conclusion and Future Work	66
	References	67
	Vita	73

List of Figures

Figure 1 Basic IoT architecture	17
Figure 2 Basic Timed Finite State machine model	19
Figure 3: Part of the timed FSM of the Motion Sensor (Normal transition)	22
Figure 4: Transition representing a model based mutant.....	22
Figure 5 An IoT Architecture for the PIR motion sensor	32
Figure 6:Timed extended FSM of the PIR Motion Sensor.....	34
Figure 7: An IoT Architecture for the Ultrasonic motion sensor.....	35
Figure 8: Timed FSM of the Ultrasonic Motion Sensor.....	37
Figure 9: Working of the RFID Card Reader	39
Figure 10: An IoT Architecture for the RFID Card Reader.....	39
Figure 11: Timed extended FSM of the RFID Card Reader.....	42
Figure 12: Part of the timed FSM of the Motion Sensor (Normal transition)	45
Figure 13: Mutated transition for battery draining attack.....	45
Figure 14: Part of the timed FSM of the Motion Sensor (Normal transition)	46
Figure 15: Mutated transition for sleep deprivation attack.....	46
Figure 16: Part of the timed FSM of the Motion Sensor (Normal transition)	47
Figure 17: Mutated transition for data falsification attack	47
Figure 18: Part of the timed FSM of the Motion Sensor (Normal transition)	48
Figure 19: Mutated transition for replay attack.....	48
Figure 20: Part of the timed FSM of the Motion Sensor (Normal transition)	49
Figure 21: Mutated transition for man in the middle attack	49
Figure 22: Experimental setup	54

List of Tables

Table 1: Summary of Intrusion detection systems with alternate techniques.....	27
Table 2: Description of the input and output of the motion sensor	34
Table 3: Description of the input and output of the ultrasonic motion sensor.....	38
Table 4 Description of the input and output of the RFID Card Reader.....	43
Table 5:Types of attacks in IoT devices	44
Table 6: Summary of generated security threat mutants	50
Table 7: Summary of modeled mutants for motion sensor.....	51
Table 8: Summary of modeled mutants for ultrasonic motion sensor.....	51
Table 9: Summary of modeled mutants for RFID Card Reader	52
Table 10: Hardware specification for the edge device	54
Table 11: Test cases for motion sensor.....	57
Table 12: Test cases for ultrasonic motion sensor	57
Table 13: Test cases for RFID Card reader	58
Table 14: Implementation tools and process automation	59
Table 15: Mutation score for code-based mutants for motion sensor	61
Table 16: Mutation score for code-based mutants for ultrasonic motion sensor	61
Table 17: Mutation score for code-based mutants RFID Card Reader	62
Table 18: Summary of Mutation score for code and modeled mutants.....	62
Table 19: Power consumption and comparison of IoT devices under attack	64
Table 20: Power consumption and comparison of IoT devices when running test cases	65

List of Abbreviations

ANN	Artificial Neural Network
BD	Battery Draining
DFA	Data Falsification Attack
DL	Deep Learning
DoS	Denial of Service
EFSM	Extended Finite State Machine
FSM	Finite State Machine
IDS	Intrusion Detection System
IoT	Internet of Things
MITM	Man in the Middle
ML	Machine Learning
MQTT	Message Queueing Telemetry Transport
NN	Neural Network
RA	Replay Attack
TC	Test Case
TFSM	Timed Finite State Machine

Chapter 1. Introduction

1.1. Problem Formulation

In this chapter, we will present an introductory overview on the types of attacks on small resource constrained IoT devices. We will also discuss the shortcomings that are faced in this field. Following this, we will describe the problem statement that is being addressed and solved in this thesis.

1.2. Overview

Internet of Things (IoT) systems are increasingly being deployed in smart homes [1], smart buildings [2], industrial internet of things (IIoT) [3], smart cities [4], intelligent farming and agriculture [5], smart transportation [6], supply chain management [7], and smart healthcare [8]. However, IIoT systems are vulnerable to security violations [9][10]. Specifically, IIoT edge devices can be hijacked (e.g., Botnet) and used to launch thousands of Distributed Denial of Service (DDoS) attacks [11]. IIoT edge devices can also be involved in node capture and replay attacks [12]. It is also relatively easy to eavesdrop on IIoT devices by monitoring the traffic going to and from a device [13], [14]. Further, sleep deprivation attacks [15] can be used to drain batteries. Default passwords while pairing [16], configuration and device authentication [17], [18], and legacy authentication mechanisms [19] can also compromise IIoT devices. Even the services on a device can be exposed and lead to security violations [20]. Many IIoT edge devices use mobile apps for configuration and these apps may suffer from overprivileged issues [21], [22] as well. Insecure hardware interfaces [23] is another key problem. Over the Air (OTA) updates for such devices can also be compromised [24] [25]. Finally, IIoT edge can fall prey to side channel, spyware and backdoor pin code injection attacks [26].

A variety of techniques to enhance security of IIoT systems have been proposed. These include better encryption for resource constrained IIoT devices [27], software defined networks [28], and using machine learning techniques for intrusion detection [29]. One key challenge faced by the solutions is the resource constrained nature of the simpler IIoT edge devices. Solutions to resource constraint problem include off-loading the

security task completely [30], using edge routers [31], or using a distributed fog approach [32]. A second challenge to the proposed solutions is to ensure that the solutions work in most situations. In this context, techniques using machine learning (e.g., [33],[34],[35],[36],[37]) are inherently limited by the quality and representativeness of the data the models are trained on. Machine learning models can over fit the data and are not necessarily generalizable [38]. Indeed, many approaches to address overfitting in machine learning models have been proposed [39], but overfitting remains an open problem. Some recent work in explainable AI (e.g., [40]) tries to address one aspect of this problem by building explainable systems to enhance trust, but do not solve the problem completely.

Therefore, we propose an intrusion detection approach for the small resource constrained IoT edge devices. The proposed approach tries to address two problems of the resource-constraints of IoT edge devices and generalization of the proposed solution. The approach is intended and limited to a wide variety of simple IoT devices including devices that collect temperature, humidity, pollution data, private entryways, infant screens, smart bulbs, proximity sensors, motion sensors, Radio-Frequency Identification (RFID) card readers, etc. The basic idea is to have an intrusion detection approach that can run on these small microcontroller-based edge devices efficiently with little overhead, and at the same time is systematic enough to generalize analytically. The proposed approach is novel as it is based on formal modeling of IoT devices and mutation testing. Formal modeling techniques provide mathematically rigorous approaches for the specification and validation of software and hardware systems whereas mutation testing deals with the generation of mutants from a given specification (or programming code) representing possible deviation from the original model (code), and then deriving tests that can distinguish each derived mutant that has a behaviour different than the original specification (code). However, an impediment to mutation testing that limits its use in many applications is the high costs of deriving and detecting many mutants. Accordingly, a related main line of research is focused on the selection of appropriate operators that yield less mutants and tests while providing high fault coverage.

1.3. Thesis Objectives

Our approach for the detection of IoT attacks focuses on modeling the behavior of small IoT edge devices as finite and extended finite state machines with timeouts. We also model common types of IoT device attacks as special mutants of these machines. Afterwards, we use mutation testing for deriving tests that detect or kill such mutants, i.e., distinguish the attacks from the normal behaviour of the specification machine. In addition, we present a framework for implementing our method and provide relevant case studies.

1.4. Research Contribution

The contribution of this work includes the following:

- Model the behaviour of small IoT edge devices (motion sensor, ultrasonic motion sensor, and the RFID card reader) as state machines with timeouts.
- Consider common types of IoT device attacks; namely, the basic battery draining, sleep deprivation, data falsification, replay, and man in the middle attacks, and show how each of these attacks can be modeled as a collection of special mutants of the corresponding timed state machine specifications. Then, we use the mutants and their specifications to drive tests that cover these attacks. We also consider tests for detecting actual physical device manipulation.
- Build an environment that incorporates the proposed work and assess the work using this environment. The environment includes related hardware architecture, an IoT framework for running a program on microcontroller, data collection for observing the behaviour of the program and a testing framework for the derivation and injection of mutants/faults into the program and for detection of these faults in practice.
- We assess the feasibility of applying the work in practice; in addition, we assess the impact of threats on battery drainage, determine the costs of running the tests on battery drainage, and determine the coverage of security tests against traditional code-based faults showing the need for considering such tests.

1.5. Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides an introduction on timed state machines, IoT device architecture, mutation testing. Chapter 3 then provides state machine models for the IoT devices (Motion sensor, Ultrasonic motion sensor, and RFID). It also shows how each of these threats can be modeled as a special mutant of the state machine models and presents the overall architecture of how (mutation) testing is implemented for the entire system. Chapter 4 provides the evaluation of the proposed framework, including the identification of the coverage of security faults, impact of threat on battery drainage, feasibility of applying the work in practice and applicability of the work to realistic systems. Chapter 5 provides the conclusion and the future work.

Chapter 2. Background and Literature Review

In this chapter, we will look at the basic terminologies and techniques used to build our framework. We then describe the type of mutants that are being used for testing our framework. Additionally, we present previous work done in the field of modelling, intrusion detection systems and analyse where our work can be mapped.

2.1. Internet of Things

Internet of things is the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data. It has wide applications in fields like healthcare, transportation, smart homes, etc. An edge device is a device that collects data from sensors like motion sensor, temperature sensor, humidity sensor and transmits data to the network. The working of a basic edge device is simple. It performs limited set of actions.

Figure 1 explains the basic IoT architecture with different layers. The physical layer is the basic layer that encompasses physical devices, such as sensors and actuators. Sensors capture data from the surrounding environment, while actuators enable IoT devices to interact with the physical world by executing actions. Instances include temperature sensors, motion detectors and cameras. The network layer transmits the data received from the sensing layer to the middleware. The data can be transferred using different protocols like Wi-Fi, Bluetooth, Zigbee, RFID and cellular networks. The middleware layer processes raw data to produce meaningful results. This layer is pivotal in standardizing and normalizing data, simplifying the understanding and utilization of information by applications. It also facilitates communication and interaction between diverse devices and platforms. The application layer outputs the data processed by the middleware layer. The concept of smart homes, smart vehicle, smart transportation works on top of this layer. The business layer produces flow diagrams, graphs and statistics based on the data derived from the application layer. This layer empowers businesses to extract valuable insights, optimize operations, and make informed decisions based on the data generated by the IoT ecosystem. Top level analysis can be obtained from the results generated from this layer.

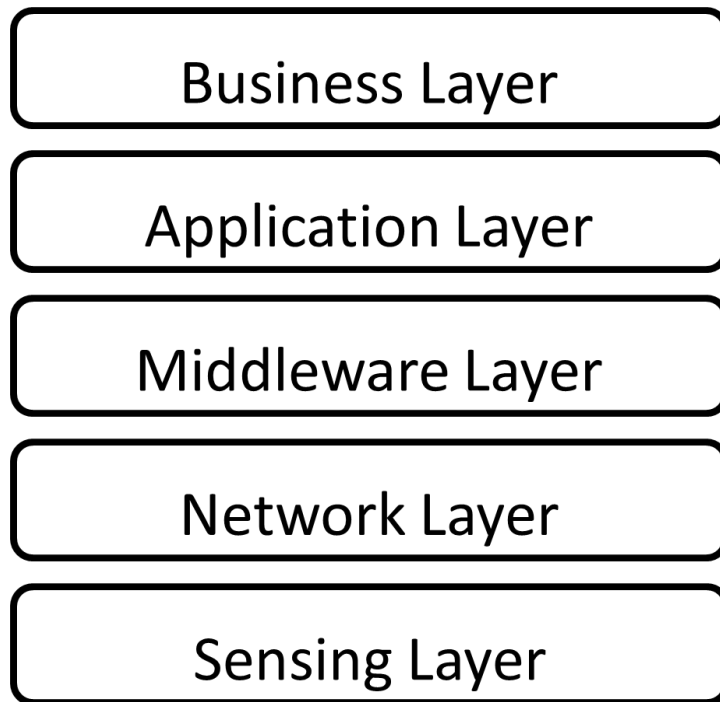


Figure 1: Basic IoT Architecture

2.2. Finite State Machines

A Finite State Machine (FSM) model of a system is specified as a finite set of states, inputs, outputs, and transitions between states each labeled by an input/output interaction. The Extended FSM (EFSM) model extends the FSM model with parameterized inputs/outputs, variables, predicates, and update statements. Thus, a transition can be labelled by a pair of possibly parameterized input/output interactions and the transition can be guarded by a predicate (guard) that must hold true for the transition to be executed. Upon its execution, new values can be assigned to variables based on the transition update statements. Both these models can be extended with a clock to represent the inclusion of time aspects. The clock specifies the timeout at the state, and it can be used for limiting the stay at the state. This means that if no input is applied at the current state before the timeout expires then the timeout is executed, and the machine resets the timer while moving to the target state. State machines are underlying models of industrial strength tools such as SDL and UML state charts, etc. In the following sections, we define these models as we use them to precisely describe the behaviour of the IoT devices and their attacks. For more information about state

machine models, mutation testing approach and applications the reader may refer to Mathur [41] for foundation of testing, Merayo et. al [42] for extended timed FSM models, Bresolin et al. [43] for dealing with deterministic timed FSM models, and Bresolin et al. [45] and Tvardovskii et al.[78] for equivalence checking and testing of timed FSMs.

An FSM is a 5-tuple (S, I, O, h_S, s_1) where I and O are input and output alphabets, S is a finite non-empty set of states with the designated initial state s_0 , and $h_S \subseteq (S \times I \times O \times S)$ is the transition relation. A transition in h_S , given in the form (s, i, o, s') indicates that if the machine is at current state s , upon receiving the input i , it produces the output o while moving to state s' . An *FSM with timeouts*, a timed FSM (TFSM) for short, is an FSM annotated with a *clock* that is reset to 0 at the execution of any transition. Such a TFSM can have input timeout transitions. When an input timeout expires at a state, the TFSM can spontaneously move to the destination state of the timeout transition while resetting the time to 0. Thus, a TFSM is a 6-tuple $Q = (I, S, O, h_S, \Delta_S, s_1)$ where $\Delta_S: S \rightarrow S \times (N \cup \{\infty\})$ is the timeout function, where N is the set of positive integers: for each state, this function specifies the maximum time for waiting for an input. Given state s of Q such that $\Delta(s) = (s', t_{out})$, if no input is applied before t_{out} expires, then at t_{out} , Q moves to state s' and the clock is set to 0. If $s = s'$ then the clock is set to zero when the timeout is expired. A transition $(s, i, o, s') \in h_S$ means that Q being at state s accepts an input i applied at time $t < t_{out}$, measured from the moment when the clock was reset at state s , then Q produces o and the clock then set to 0.

Given a TFSM Q , a *timed input* is a pair (i, t) where $i \in I$ and t is a real; a timed input (i, t) means that input i is applied to Q at time instance t where t is a local time (at the state). A timed input of a timeout transition is written as (ε, t_{out}) and its output is ε , where ε is the empty string. A sequence of timed inputs $\alpha = (i_1, t_1) (\varepsilon, t_{out}) \dots (i_n, t_n)$ is a *timed input sequence*. A sequence $\alpha/\gamma = (i_1, t_1)/o_1 (\varepsilon, t_{out})/\varepsilon \dots (i_n, t_n)/o_n$ of consecutive pairs of timed inputs and outputs starting at the state s is a (*timed*) *trace* of Q at state s . A trace of Q starts from the initial state s_0 . A *test case* of TFSM M is a finite length trace of M . A *test suite* is finite set of test cases. Note for simplicity of presentations, in the figures, we use $s - t_{out} \rightarrow s'$ to denote the timeout transition from s which leads at t_{out} to state s' .

As a simple example, consider the TFSM of the motion sensor in Figure 2. Firstly, at the initial state S_1 , the clock starts from 0, then at time $t_{out} = 2$ it moves to state S_2 while resetting the clock to 0. Then at S_2 if i_1 is not provided at time $t < t_{out} = 2$, then at $t = 2$, t_{out} is executed, and the machine moves back to S_1 while resetting the clock to 0. However, if at S_2 , i_1 is provided at time $t < 2$, then the machine executed transition t_3 where it produces o_1 and moves to state s_3 while resetting the clock to 0. The traces that correspond to these two scenarios are $(\varepsilon, t_{out} = 2) (\varepsilon, t_{out} = 2)$, and $(\varepsilon, t_{out} = 2) ((i_1, t < 2), o_1)$, respectively.

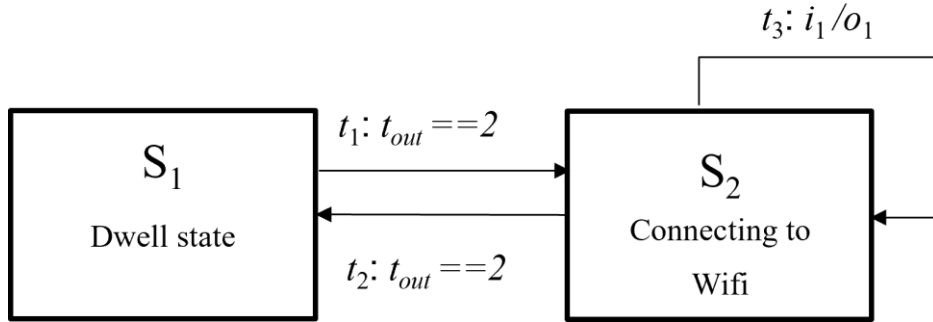


Figure 2: Basic Timed Finite State Machine

In this thesis, we also use the extended (timed) FSM model for modeling IoT devices with variables and updates. The extended FSM (EFSM) model extends the FSM model with variables V , update statements, guards, and parameterized inputs and outputs. Thus, an EFSM M has also finite sets of variables V and transitions T . Let us assume that there are n variables. V is a set of n variables and \mathbf{v}^0 is the vector of initial values of these variables. We write $\mathbf{v} = (v_1, v_2, \dots, v_n)$ for an assignment of values to the n variables of M . The initial value assignment of the variables is called the *initial valuation* \mathbf{v}^0 . A pair (s, \mathbf{v}) is called a *configuration* and (s, \mathbf{v}^0) is the initial configuration of the machine representing the initial values of variables.

A transition $T_j \in T$ has the form $T_j = (s, i(p), [G], up, o, s')$, where s and s' are the *current and ending* states of T_j , $i \in I$ is an input interaction with the parameter p , $[G]$ is the *enabling guard* (or *predicate*) of T_j which depends on the values of the state variables and the value ρ of parameter p , and up is a concurrent update statement which defines new values for certain variables in V as functions of the current values of all variables of M and ρ . An update statement of variable v in V is written in the form $\{v: = \text{expr};\}$, where expr is an expression over the variables in V and the parameter p . The

EFSMs presented in this thesis use interactions with single parameters. Note an input i may not be parameterized; thus, written as i .

The meaning of T_j is the following: If M is in state s , then M may make a transition to state s' by receiving the input i with parameter value ρ if $[G]$ is True for the current values \mathbf{v} of the variables and ρ . If the transition is executed, the values of \mathbf{v} will be changed to \mathbf{v}' according to up , and afterward the output o will be produced. An EFSM *with timeouts*, a timed EFSM for short, is an EFSM with a single clock that is reset at the execution of each transition. In addition, at each state the machine has a timeout function Δ_s like TFSMs.

Given a timed EFSM N , a *timed input* is a pair $(i(\rho), t)$ where $i \in I$, ρ is a value of parameter p , and t is a real; a timed input $(i(\rho), t)$ means that input i carrying the parameter value ρ is applied to the machine at time instance t where t is a local time. A timed input of a timeout transition is defined above for TFSMs. A sequence of timed inputs $\alpha = (i_1(\rho_1), t_1) (\varepsilon, t_{out}) \dots (i_n, t_n)$ is a *timed input sequence*. A sequence $\alpha/\gamma = (i_1(\rho_1), t_1)/o_1 (\varepsilon, t_{out})/\varepsilon \dots (i_n, t_n)/o_n$ of consecutive pairs of timed inputs and outputs starting at the initial configuration is a *timed trace* of timed EFSM at state s . A *test case* of timed EFSM M is a finite length trace of M . A *test suite* is finite set of test cases. For detailed formal definitions of timed FSM and EFSM models, the reader may refer to [42][44].

As an example, consider the timed EFSM of the RFID reader in Figure 11 with variables *counter* (integer) *rchar* (char with possible values $[a\text{---}z, 0\text{---}9]$). At S_{11} , assume *counter* = 0 and the string *code* is empty (ε), if the input *data_byte*('B'); i.e., *rchar* = 'B', is received at time t_1 , as the guard $[counter] < 10$ of t_{23} holds, then $t_{23} = (S_{11}, \text{byte_code}(rchar), [counter < 10], \{ counter := counter + 1 ; code := code + rchar \}, o_{13}, S_{11})$ can execute, where *counter* is incremented by one ($counter := counter + 1$) and the received character 'B' is appended to *code*, $code := code + rchar$, then the output o_{13} is produced, and the machine moves to state S_{11} again while setting clock to 0. Thus, in fact the machine moves from configuration $(S_{11}, 0, \varepsilon)$ to $(S_{11}, 1, \text{"B"})$. Then, if *data_byte*('A') is received at time t_2 then t_{23} is executed again and the machine moves to $(S_{11}, 2, \text{"BA"})$. This is realized by the timed trace $(\text{data_byte}('B'), t_1)/o_{13} (\text{data_byte}('A'), t_2)/o_{13}$.

2.3. Code and Model based Mutants.

Mutants represent a collection of variants, derived from the original code/model with slight variations. We cover code-based and model-based mutants for our scope of work. Additionally, mutation testing is a structural testing methodology that leverages code's structure to direct the testing procedure.

2.3.1. Model based Mutants.

Given a state machine (timed FSM or timed EFSM) N representing the behaviour of an IoT device, we model the behaviour of typical device attack as a mutant of N . Then, we use the derived mutant and N to derive test that *detects/kills/distinguishes* such attacks from N . A test case *distinguishes* N from a mutant if the output sequences of N and the mutant, with respect to the input sequence of the test case, are different. Approaches for deriving such tests are elaborated in many books and papers including [44][46][47][48].

A mutant of N has a *single output fault* if it has a transition with an output different than that specified at the corresponding transition of N . That is, for certain transition of N with output o , the corresponding transition in the mutant has an output $o' \neq o$, $o \in O$.

Additionally, a mutant has a *timeout fault* if it has a state with a different timeout than that specified at the corresponding state in machine N ; i.e. for certain s of Q with timeout t_{out} , state s in the mutant has timeout t'_{out} such that $t'_{out} \neq t_{out}$. A mutant has an *added transition fault* if the mutant is obtained by adding a new transition to N . A mutant has a *new state fault* if it is obtained from N by adding a new state with related outgoing transitions and modifying the ending states of some other transition(s) such that the added state becomes reachable through this (these) transitions.

In the example, Figure 3 represents the normal state where the state S_2 has a self-loop with the input i_1 and output o_1 . This is a small subpart of the TFSM of a motion sensor which will be studied in detail in the next chapter. In Figure 4, at transition t_3 if we change the output from o_1 to o_5 , it will be a mutated state.

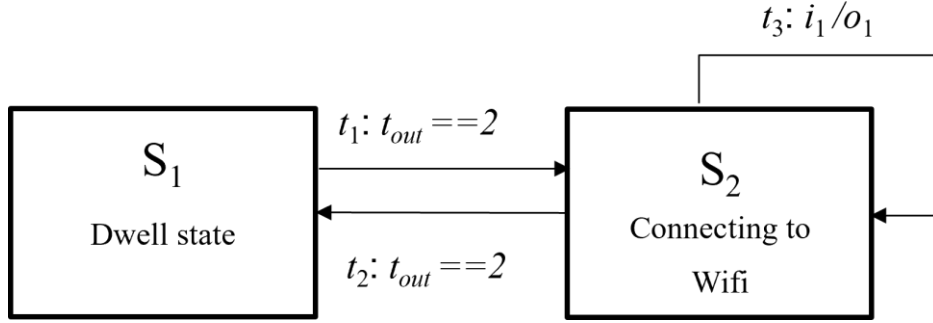


Figure 3: Part of the timed FSM of the Motion Sensor (Normal transition)

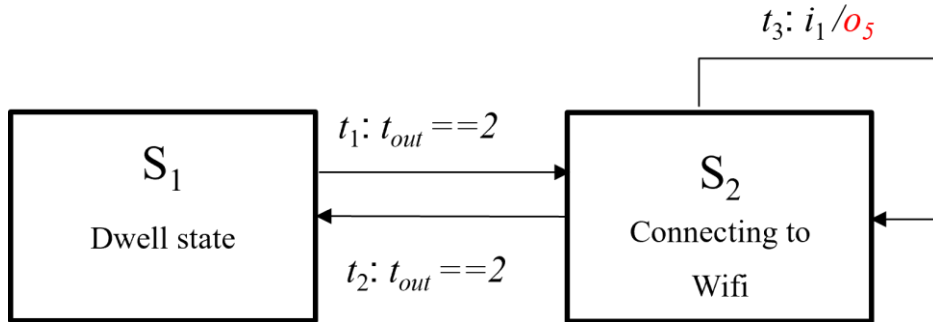


Figure 4: Transition representing a model-based mutant

2.3.2. Code based mutants

We present C++ implementations of the considered IoT devices. For example, if the program has a line written as `String msg1 = "Health Check"`, then its mutated file would change the line to `String msg1 = "&Health Check"`.

In fact, we consider testing the attacks in a real system. For each considered attack, we inject the attack into the code implementation creating a code mutant of that implementation. Then, we run the corresponding test case on the mutant to check if such an attack is detected in practice. Furthermore, we create mutants from the implementation considering traditional C based mutation operators illustrated in [41] to assess the effectiveness of derived (attack) test suites in detecting traditional (none attack based) code faults. The mutation score of a certain test suite [41], with respect to a given collection of mutants, is computed based on the percentage of the number of mutants killed by the suite divided by the number of all derived mutants minus the number of alive mutants that cannot be killed by any test suite. Alive mutants are determined by running a collection of many test suites against the mutants and those

which not killed by any suite are considered alive. Thus, a test suite that kills all non-alive mutants has a 100% mutation score.

2.4. Related Work

A comprehensive survey of the latest IDSs designed for the IoT model is presented in [68], with a focus on the corresponding methods, features, and mechanisms. This article also provides deep insight into the IoT architecture, emerging security vulnerabilities, and their relation to the layers of the IoT architecture. This work demonstrates that despite previous studies regarding the design and implementation of IDSs for the IoT paradigm, developing efficient, reliable, and robust IDSs for IoT-based smart environments is still a crucial task. Cybersecurity in IoT is thoroughly reviewed in [15] and it explains the cybersecurity architecture and terminologies. It describes the various attacks like replay attacks, timing attacks, Node Capture attacks, malicious data attacks, DOS attacks, side channel attacks, Man in the middle, Sybil attacks etc. The attack taxonomy given in this paper gives a clear picture of the security issues in IoT devices. This has helped in identifying the types of attacks that can be created by generating mutations in the state model for the required project. An overview of the Internet of Things (IoT) with emphasis on enabling technologies, protocols, and application issues has been explained in [69]. It explains different application protocols, messaging protocols, infrastructure protocols. This paper has helped to gain a better insight of different components and protocols and has helped in understanding the overall architecture of the IoT. The vulnerabilities in the IoT systems has been explained in [70]. It also shows how each of these attacks were generated by either using some available tools, or by creating scripts. The results were compared based on the different protocols that are used for the edge devices like MQTT, MQTT+TLS, CoAP and CoAP+DTLS. This paper helped in understanding the ways in which the vulnerabilities were injected into the device to test it for intrusions. This paper will help in developing a better intrusion detection system for the project.

For the project implementation we make use of Timed Finite state machines (FSM) instead of normal finite state machines because IoT devices are based on the concept of timeout. This means if a device is not responding within a desired amount of time, we timeout and move to the next state. Furthermore, if the device is in dwell state, we time out after a certain period and go to the active state again where it is open to receive

messages. Finite timed FSMs described in [71] explains how to derive conformance tests for timed FSM with a single clock. Although the test suites derived from this approach can have many test cases, we make use of a limited set of test cases which are specific to the IoT security attacks to perform testing.

Fu et al. [53] describes an automata-based intrusion detection system for IoT devices. While they use automata models to automatically detect jam-attack, false-attack, and reply-attack we have used a finite state machine to detect attacks like battery draining, sleep deprivation, data falsification, replay attack and man in the middle attack. Arrington et al. [54] use behavioral anomalies to create an intrusion detection system for IoT devices with behavioral modelling. Their method captures the sensor data as a set of events to give a numerical representation for behavioral identity. They have created NPC (Nonplaying characters) with various roles (owner, attacker, etc). The behavioral patterns are created based on the behavior of these NPCs. The results and conclusion of this method is through simulation whereas we have made use of the devices in real time, injected mutants and created intrusion in real time rather than a simulated environment. Sedjelmaci et al. [55] used game theory to create a game model of the IoT device for a normal user and an intruder. This technique was used to detect intrusion into a system. Anomaly detection gets activated only when a new attack is most likely to be expected. The attack signatures are recorded based on a learning algorithm that builds a rule based on newly attacked pattern it discovers during learning. This was then used to extract the Nash equilibrium value which was in turn used to decide the use of the intrusion detection system for that scenario. Ge et al. [56] used a graphical security model to detect vulnerabilities in cross protocol devices by grouping the IoT devices with same communication protocol. They used a multi-layer hierarchical attack representation models by combining attack graphs and attack. They have tried to detect the node controlling and sinkhole attack with this approach and have also analysed the denial-of-service attacks using the same approach. While this is also a model-based approach, our methods focus on several types of attacks. Loise et al. [57] introduced 15 general security mutation operators for Java programs, different than those we introduced here for edge devices, and they assessed these operators in respect to traditional Java operators. A probabilistic approach was proposed in [59] to detect the fault in the state transition for an FSM. This algorithm does not care about the input or the order of the states. It considers the deviation from the stationary

distributed probabilities that are needed from a fault free machine. This implementation can be useful in the scenarios where the time delays and synchronization delays corrupt the order of observations. Thus, our work complements previous related work considering the first-time modeling of IoT devices and attacks using the timed finite and extended finite state machine models. In addition, we used the models to create dedicated threat mutants and derive tests that kill these mutants. The dedicated tests are few in numbers and this enables the use of the proposed work in contrast to deriving tests using arbitrary mutation operators or deriving complete tests. [47][58] were referenced for detecting arbitrary faults in the implementation under test.

There are various other alternate techniques that have been investigated to detect attacks/ intrusions into the IoT devices. Some of the techniques include using Random Neural Networks (RNNs), Artificial Neural Networks (ANNs), Kmeans algorithm, Support Vector machines (SVMs), machine learning etc. At the end a summary of all the techniques is given that compares the accuracy of each of these methods used.

An intrusion detection and prevention mechanism has been explained in [60] by implementing an intelligent security architecture using random neural networks (RNNs). The application's source code is also instrumented at compile time to detect out-of-bound memory accesses. It is based on creating tags, to be coupled with each memory allocation and then placing additional tag checking instructions for each access made to the memory. This paper was used to analyse the different methods that exist to create an intrusion detection system. An intrusion detection system using data mining techniques has been proposed in [61]. The author compares the results of ANN, KNN and SVM and claims that the random forest technique outperforms other techniques with a total accuracy of 98.94. The author has used various parameters like F measure, AUC (Area under the ROC (Receiver operating characteristic) curve) and Kappa value. From these values he concludes that the data mining techniques can be used successfully for intrusion detection in the smart grid environment. This paper tells us about the different techniques that can be used for intrusion detection in smart system which are different from our idea of implementation. It also explains the various parameters that were used for evaluation like F measure, AUC and kappa value. Since supervised machine learning algorithms require high volumes of training data, [62] proposes that using a semi supervised model can perform well as compared to several

supervised machine algorithms as it will need little labelled data and the training time will be very less. The author thus uses a hybrid technique that uses Active learning support vector machine (ASVM) and Fuzzy Cmeans(FCM) clustering for designing an efficient Intrusion Detection System. The performance measures used are True positive, True Negative, False Positive, False Negative and the accuracy is calculated using $(TP+TN)/(TP+TN+FP+FN)$. When compared with other hybrid methods like SVM+k-means, SVM+FCM, ASVM+k-means, the proposed solution out beats others by achieving a detection rate of 99.6%.

In [63], the author uses Raspberry Pi that runs Snort – an open-source Intrusion detection system to detect and evaluate the attacks done on the IoT system. The author suggests on using a Raspberry Pi instead of any other computing device as it is portable, easy to use, has minimum configuration. The main goal of the paper was to assess the performance of Raspberry Pi as a host for Snort. It acts as monitor node to analyse different types of network traffic. Vulnerabilities of the MQTT protocol are discussed in [64]. It demonstrates a few attack scenarios and proposes certain security measures to prevent the attacks. Attacks like publishing and subscribing to unauthorized topics, information gathering, packet capture using Wireshark, modifying the data in transit, botnet over MQTT, etc. It also explains the loopholes in the MQTT protocol. These loopholes are used to generate mutants for the project. An Intrusion detection technique using Online Sequential Extreme Learning Machine (OSELM) is proposed in [65]. The proposed solution tries to detect an attack at the fog level instead of computing it on the cloud for getting faster results. The packet arriving at the fog can be classified as a normal packet or an attack with the help of the training data. The attack details are also sent to the cloud server for summarizing the results and future training purposes. A hybrid learning approach is used in [72] where the data collection is done in 2 stages – local and global. The local detection is done by dedicated sniffers using supervised learning approach and the global stage collects data from different local nodes. It applies linear regression to generate profiles for normal and corrupt nodes. The data is collected by simulating data from 35 nodes. A discrete network simulator called Cooja was used for simulating this environment.

A unique system named as Kalis is introduced in [67] that does not follow any fixed methodology for applications and protocols. Instead, it adapts the detection technique

based on the network structure. It collects knowledge about the features of the monitored network and leverages such knowledge to create a set of detection techniques. This system is implemented using Java on an Odroid xu3 development board. A TelosB wireless sensor mote with a custom TinyOS is used to interact with IEEE 802.15.4 traffic. The concept of Principal Component Analysis (PCA) is used in [73] to reduce the number of features from a large dataset. Irrelevant/noisy data is eliminated using dimension reduction algorithm. Then a classification is model built using Softmax regression algorithm and k-nearest neighbour (KNN) algorithm which is used to classify the type of attack. The alternate intrusion detection techniques have been summarized in Table 1.

Table 1: Summary of Intrusion detection systems with alternate techniques

Technique used	Attacks prevented	Accuracy %
Random Neural Network [60]	Performance degradation attacks and illegal memory access attacks	97.23%
Data mining[61]	Denial of service attack	98.94%
Semi supervised network[62]	Not specified	99.6%
Snort[63]	Network level	Not specified
Online sequential extreme machine learning[65]	Fog level	96.54%
Supervised learning [66]	Network attacks	100%
Knowledge driven [67]	Probing attack, DOS attack, user to Root attack, Remote to local attack	84.406%

We have also studied related work done in the field of digital twins that have been used to replicate the working of the actual device. Using digital twin has been proved to be useful at scenarios where accessing the physical device directly is a challenge. It behaves in the same way as the physical device, so if anything goes wrong with the physical device it can be replicated with the digital twin. This approach is very useful, because digital twins can be more helpful in fixing the issue. It can also be used to optimize and improve the working of the physical device because solution will be easier to find on the twin rather than the device itself.

The purpose of doing a literature review on the use of digital twins in different domains was to get an idea about the technologies that were used in creating a twin. In [74] they use Experimentable Digital Twins (EDTs) in the Model Driven Engineering process to replicate how system functions would be executed by physical parts of the developed product. In Closed loop Systems Engineering (CLOSE) the functions are represented from the Essential System that are executed “insitu” by the model’s control logic using available real-time information from the domain models. This allows virtual verification and validation of the designed parts and their behavior in an assembly or product, respectively. The paper also describes a case study of an Underwater Autonomous vehicle. The parameters (like mass, inertia, center of gravity, etc) for this case are calculated from 3D CAD Data. These parameters can change if the properties of the actual physical system changes. The basic simulation model was developed in python. The parameters are then fed into the simulation model. The requirements are expressed in the code as conditional threshold statements for all the state variables. The results are then analyzed by varying the parameter values, and a graph is generated with these values for analysis. In [75] the digital twin is built using an object oriented realtime database that contains details of the simulation model. For example, for an optimization problem, the simulated controller varies the relevant parameters based on a predefined cost function. The results for different variation in the parameters can be analyzed which can then be applied to the real system. A virtual test bed is created that represents a system of integrated digital twins. The simulators that simulate the entire VTB (Virtual Test Bed) is generally a software program that reproduces the systems dynamic behavior. The microkernel used is a VSD (Versatile Simulation Database), an object-oriented realtime database holding a description of the underlying simulation model. VSD integrates different simulation systems like MATLAB.

The use of digital twin in a vehicular system is demonstrated in [76]. It is divided into 3 sections physical level, cyber level, and integration of sensor-services fusion. The real time decisions need to be made on the physical layer. However, the data from this layer can then be used by the digital twin which is implemented at the cyber level for delayed operations. For example, the real time data like location is then used by the digital twin to give recommendations like nearby hotel/parking, accident statistics etc. The implementation of the digital twin is done in java. In [77], the properties of a digital twin to replicate different states and analyse their outcomes for comparison in. Since

DTs are completely transparent as they are created virtually, their internal state is available for analysis at every point and analysis becomes easy. They have used the scenario for an autonomous driving car where there is a truck taking a turn into the lane of the car. The digital twin tries to simulate different state trajectories to predict the outcome for the given situation. A similar approach can be followed while building the intrusion detection system where the digital twin can be used to see how the sensors would behave when there is an abnormal behaviour and which states can it possibly go to.

Chapter 3. Problem Statement and Methodology

In this chapter, we will describe how the working of simple IoT devices can be modelled into timed FSMs/EFSMs and how their behaviour can be mapped into different states with certain inputs and outputs. For our study we have modelled the working of a motion sensor, RFID card reader and ultrasonic motion sensor. Additionally, different types of security attacks are modeled into state machine mutants. We will consider 5 different type of attacks that would be then injected into our modelled device FSMs as mutants.

3.1. Problem Statement

IoT devices being resource constrained complicates the process of adding safety measures. Also, most of the data that is used to train intrusion detection systems are collected on a random basis. This needs long computation hours and higher processing machines to process, label and train the network data. We propose a novel technique that detects attacks on IoT devices based on formal modeling and mutation testing. This approach enables us to collect the data smartly from the network reducing the computation time.

In this research we have modelled the behaviour of a motion sensor, RFID card reader and ultrasonic motion sensor as a Timed Finite State Machine. With an aim of detecting security breaches in these devices, different types of attacks, namely battery draining, sleep deprivation, data falsification, replay and man in the middle attack has also been modelled into special mutants that correspond to the specifications of the Timed Finite State Machine.

3.2. Problem Formulation

Here we have demonstrated an experimental setup including various components and how they communicated with each other. Different types of sensors/actuators (motion sensor or RFID) are connected to an ESP32 microcontroller which in turn is connected to the MQTT broker using WiFi. The messages sent and received from the broker can be viewed by the Middleware.

Our experimental setup is divided into 2 sections. One section describes how the code is mutated and put up in the microcontroller while the other section describes how the test sequence runs to detect the mutated code. The mutated code is derived from the device's TFSM (model-based mutants) as well as using a mutated code generator which is implemented to generate code-based mutants. We then derive the test cases from the TFSM model using formal methods which are injected as a sequence into the framework to kill the generated mutants. The formal methods show that the test cases that were injected into the framework are capable to classify the different types of attacks.

The following section will describe how the working of the IoT devices were modelled into a TFSM.

3.3. Modelling of IoT Devices as Timed Finite State Machines

In this section we show how simple IoT devices, such as the motion sensor, ultrasonic motion sensor, and the RFID card reader, can be modeled using timed FSMs/EFSMs.

3.3.1. Motion Sensor

3.3.1.1. Description

The Passive Infrared motion (PIR) sensor is used as an electronic device to measure the infrared light radiating from objects or human bodies nearby to detect whether the user is approaching or not [50]. Once the motion is detected, a signal is generated which is then amplified and digitalized by the IC. The output generated by the sensor is analog in nature which is then converted to a digital pulse. If the signal is high that means the motion has been detected. There is a threshold that has been set, above which any signal generated will produce the output as 1. Similarly, if the signal is below the threshold, it will output 0 showing that there is no motion.

A Passive Infrared motion sensor (PIR) [49] is a classic example of a sensor often used in IoT scenarios especially in the context of smart homes. Figure 5 shows a typical IoT architecture for a PIR motion sensor. The motion sensor is connected to a microcontroller that in turn is using Wi-Fi to connect to a home router which is connected to the Internet. Upon detecting motion, the microcontroller uses the MQTT protocol to send a message to the application server and potentially to apps running on a consumer's laptop or a mobile phone. From a communication perspective, the

microcontroller first must connect to the WiFi network and then connect to the MQTT Broker using TCP. A microcontroller used in such a scenario will typically not be multi-threaded and run in a single loop. Once connected to WiFi and the MQTT broker, the microcontroller continuously checks for a movement signal from the PIR motion detector. Once a signal is detected debouncing is done by sleeping for a few milliseconds. This ensures that multiple messages are not generated for the same movement event. For each movement event, an MQTT message is then published.

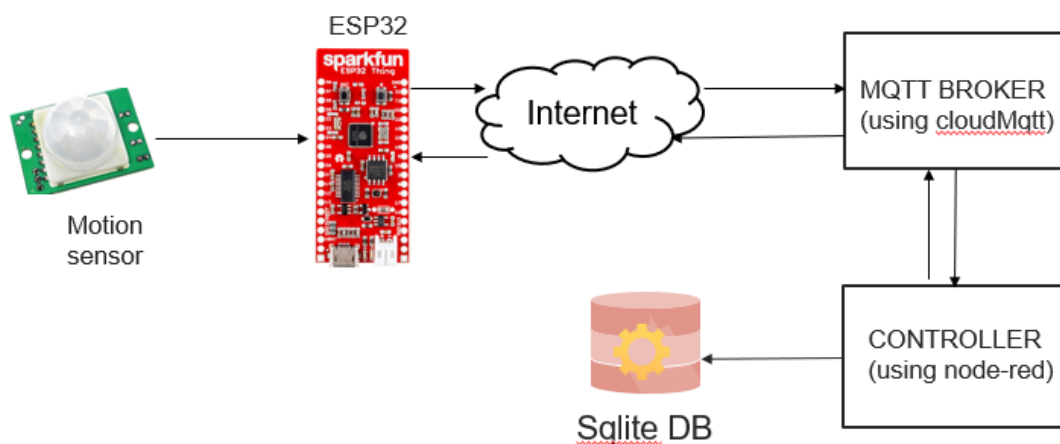


Figure 5: An IoT Architecture for the PIR motion sensor

3.3.1.2. Motion sensor Timed FSM Model

The working of the motion sensor has been modelled as a timed FSM in Figure 6. It is essentially a representation of the C++ program running on the microcontroller. Table 1 shows the various inputs and outputs of the state machine.

To describe the Table 1 in detail, input i_1 has parameterized input of $ws(0)$ that indicates that WiFi is not connected and is trying to connect. The corresponding output for this is o_1 which is a message displayed on the output console that says, “Connecting to WiFi”. The input i_2 has parameterized input of $ws(1)$ that indicates that the WiFi is connected. The corresponding output for this is o_2 which is a message displayed on the output console with string value “Connected to WiFi”. The input i_3 has parameterized input of $cc(0)$ that indicates that MQTT queue (client) is not connected and is trying to

connect. The corresponding output for this is o_3 which is a message displayed on the output console that says “Connecting to MQTT”. The input i_4 has parameterized input of $cc(1)$ that indicates that MQTT queue (client) is connected. The corresponding output for this is o_4 which is a message displayed on the output console that says, “Connected to MQTT”. The next set of inputs are events ($i_5, i_6, i_7, i_8, i_9, i_{10}$) from the application server. The input i_5 sends a message to do a health check on the motion sensor and corresponding output o_5 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Sensor is Alive”. Similarly, the input i_6 is an invalid request (A random string) and the corresponding output received for it is an MQTT message displayed on Node-Red (MQTT display server) that says “Error. Invalid message”. The input i_7 is an input requesting to stop the publishing of the messages on the MQTT queue indicated by the String “Stop Publishing” by the edge device and corresponding output o_7 received for it is an MQTT message displayed on Node-Red (MQTT display server) with string value “Publishing has stopped”. This message is displayed only after the request for Stop Publish has been processed. The input i_8 is modelled to ask for specific information from the device. In our model we have used it to ask for the WiFi name on which the device is connected. Once this request is processed, the output o_{11} displays the `device_ssid` on the Node-Red display screen. The next input i_9 is generated whenever a button is pressed on the edge device. This type of input has been considered if there is a need for an event generated by a trigger in real time. The output for this event is o_8 which will display the message that the “Button has been pressed”. The input i_{10} is an input requesting to start the publishing of the messages on the MQTT queue indicated by the String “Start Publishing” on the motion sensor and corresponding output o_9 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Publishing has started again”. The next input i_{11} is a physical event that is triggered whenever the motion has been detected by the motion sensor. Whenever the value of `motionstatus` is set to 1, the output message displayed on Node-Red is o_{11} that “Motion has been detected”.

The program is single threaded and runs in a continuous loop. There are timeouts for each state to ensure that the microcontroller does not get hung up in any state. For example, failing to connect to WiFi, the microcontroller goes back to the dwell state (S_1). Once the WiFi connection is established, the microcontroller moves to the connection to MQTT state (S_4) to try to connect to the MQTT broker. Once connected,

the microcontroller moves between states S_5 and S_{10} . This is because it is single threaded, the microcontroller needs to check for any incoming messages from the MQTT broker and to periodically check for motion. While in S_5 , many input events like $i_5 = \text{"Health Check"}$ or $i_6 = \text{"Unexpected Request"}$ are received and responded to with appropriate messages (e.g., $o_5 = \text{"Sensor is Alive"}$). Once the motion is detected in S_{10} (i.e., $i_{11} = \text{Motionstatus}(1)$), the microcontroller moves into the dwell state (S_{11}) for debouncing the signal after issuing $o_{10} = \text{"Motion has been detected"}$ message.

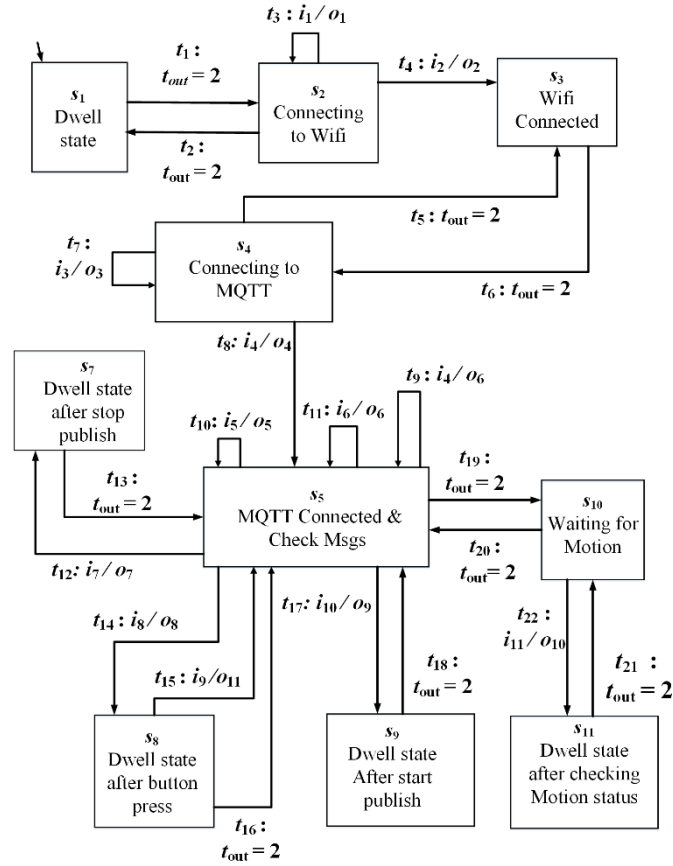


Figure 6: Timed FSM of the motion sensor

Table 2: Description of the input and output of the motion sensor

Input		Output	
i_1	ws(0)	o_1	"Connecting to WiFi"
i_2	ws(1)	o_2	"Connected to WiFi"
i_3	cc(0)	o_3	"Connecting to MQTT"
i_4	cc(1)	o_4	"Connected to MQTT"

i_5	“Health Check”	o_5	“Sensor is Alive”
i_6	“Unexpected Request”	o_6	“Error. Invalid message”
i_7	“Stop Publishing”	o_7	“Publishing has stopped”
i_8	“Give me wifi Name”	o_8	“Press any button”
i_9	buttonPress(low)	o_9	“Publishing has started again”
i_{10}	“Start Publishing”	o_{10}	“Motion has been detected”
i_{11}	Motionstatus(1)	o_{11}	device_ssid

3.3.2. Ultrasonic Motion Sensor

3.3.2.1. Description

The ultrasonic motion sensor is an electronic device that makes use of the SONAR technique to detect the distance of any object around it. It comprises of 2 ultrasonic transmitters which are namely, a receiver and a control circuit. The transmitter emits a sound with high ultrasonic frequency. The echo produced from this sound is recorded by the receiver to calculate the time difference between the sound emitted and the echo received. HC-SR04 ultrasonic motion sensor was used for this for research. The architecture for an ultrasonic motion sensor shown in Figure 7 is like that of a PIR sensor except that instead of a signal for movement a number representing the estimated distance is received by the microcontroller.

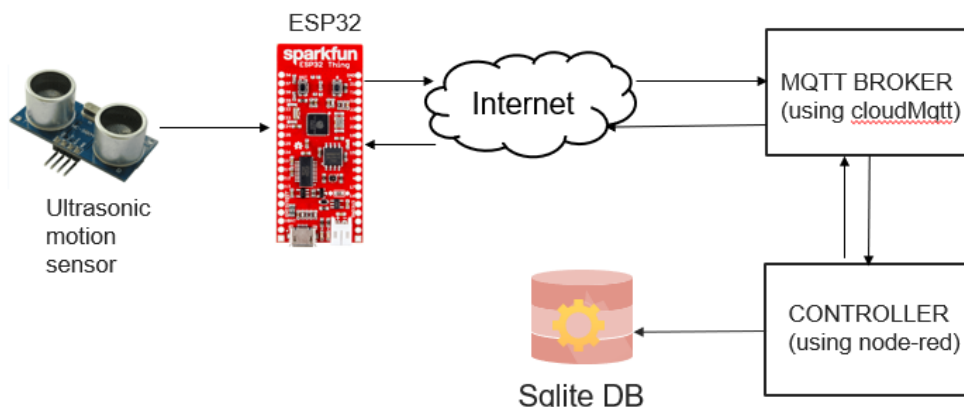


Figure 7: An IoT architecture of the Ultrasonic motion sensor

3.3.2.2. *Ultrasonic motion sensor Timed Extended FSM Model*

Figure 8 shows the time FSM for the ultrasonic motion sensor. States S_1 to S_9 of the ultrasonic motion sensor are like those of the motion sensor given in Figure 6 with the same inputs i_1 to i_9 and i_{11} and outputs o_1 to o_9 .

To describe the Table 3 in detail, input i_1 has parameterized input of $ws(0)$ that indicates that WiFi is not connected and is trying to connect. The corresponding output for this is o_1 which is a message displayed on the output console that says, “Connecting to WiFi”. Input i_2 has parameterized input of $ws(1)$ that indicates that WiFi is connected. The corresponding output for this is o_2 which is a message displayed on the output console that says, “Connected to WiFi”. The input i_3 has parameterized input of $cc(0)$ that indicates that MQTT queue (client) is not connected and is trying to connect. The corresponding output for this is o_3 which is a message displayed on the output console that says, “Connecting to MQTT”. The input i_4 has parameterized input of $cc(1)$ that indicates that MQTT queue (client) is connected. The corresponding output for this is o_4 which is a message displayed on the output console that says, “Connected to MQTT”. The next set of inputs are events ($i_5, i_6, i_7, i_8, i_9, i_{10}$) from the application server. The input i_5 sends a message to do a health check on the ultrasonic motion sensor and corresponding output o_5 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Sensor is Alive”. Similarly, the input i_6 is an invalid request (A random string) and the corresponding output received for it is an MQTT message displayed on Node-Red (MQTT display server) that says “Error. Invalid message”. The input i_7 is an input requesting to stop the publishing of the messages on the MQTT queue indicated by the String “Stop Publishing” on the motion sensor and corresponding output o_7 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Publishing has stopped”. This message is displayed only after the request for Stop Publish has been processed. The input i_8 is modeled to ask for specific kind of information from the device. In our model we have used it to ask for the WiFi name on which the device is connected. Once this request is processed, the output o_{11} displays the `device_ssid` on the Node-Red display screen. The next input i_9 is generated whenever a button is pressed on the edge device. This type of input has been considered if there is a need for an event generated by a trigger in real time. The output for this event is o_8 which will display the message that the “Button has been pressed”. The input i_{10} is an input requesting to start the publishing of the messages on

the MQTT queue indicated by the String “Start Publishing” on the motion sensor and corresponding output o_9 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Publishing has started again”.

The inputs and outputs described so far are same as that of the motion sensor. However, the ultrasonic has different S_{10} , S_{11} and S_{12} as compared to the PIR motion sensor. When the sensor goes into the active read state (S_{10}), it reads in transition t_{21} the value from the echo pin through the parameterized input $i_{11} = \text{echoPin}(\text{duration})$ which receives the (integer) parameter duration; then based on that it calculates (integer variable) distance using the update $\text{distance} := (\text{duration} * 0.034)/2$; and the outputs $o_{10} = \text{“duration”}$ and moves to state S_{11} . Afterwards, using input and $i_{12} = \text{DistanceStatus}(1)$, the sensor outputs $o_{12} = \text{“distance”}$ and moves to the dwell state S_{12} .

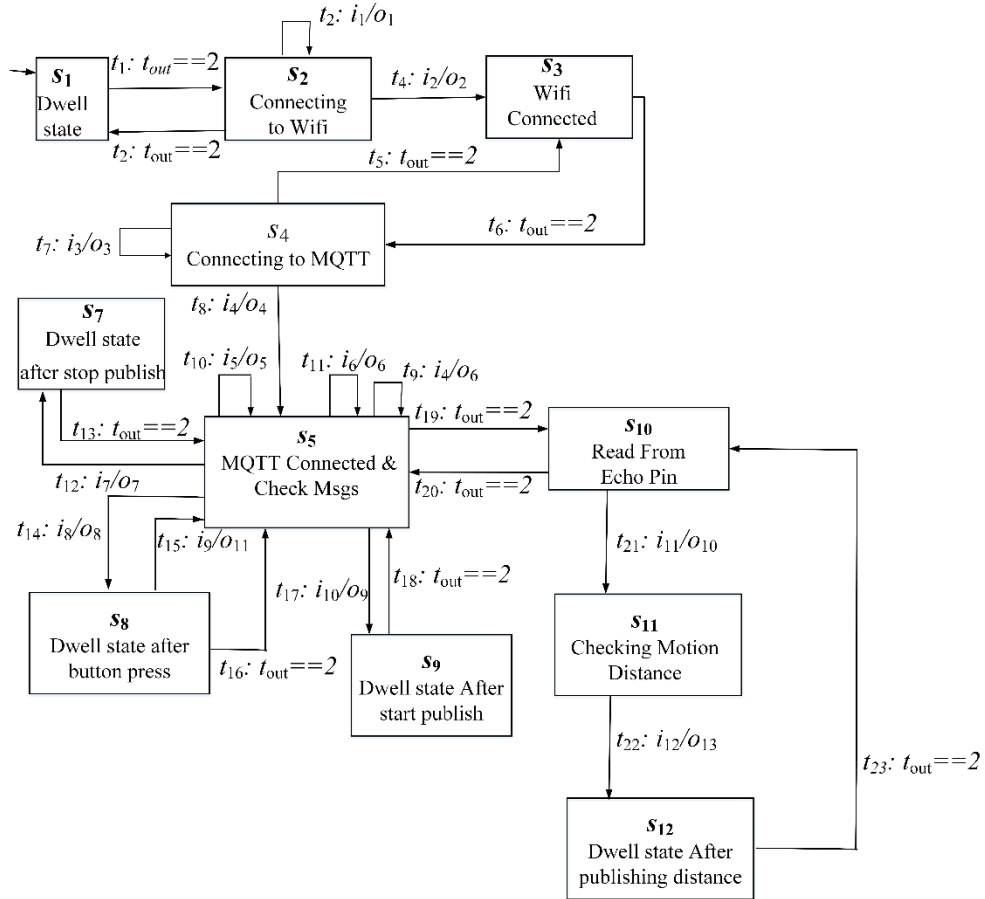


Figure 8: A Timed FSM model of the Ultrasonic motion sensor

Table 3: Description of the input and output of the ultrasonic motion sensor

Input		Output	
i_1	ws(0)	o_1	“Connecting to WiFi”
i_2	ws(1)	o_2	“Connected to WiFi”
i_3	cc(0)	o_3	“Connecting to MQTT”
i_4	cc(1)	o_4	“Connected to MQTT”
i_5	“Health Check”	o_5	“Sensor is Alive”
i_6	“Unexpected Request”	o_6	“Error. Invalid message”
i_7	“Stop Publishing”	o_7	“Publishing has stopped”
i_8	“Give me wifi Name”	o_8	“Press any button”
i_9	buttonPress(low)	o_9	“Publishing has started again”
i_{10}	“Start Publishing”	o_{10}	“duration”
i_{11}	echopin(duration)	o_{11}	device_ssid
i_{12}	DistanceStatus(1)	o_{12}	“distance”

3.3.3. RFID Card Reader

3.3.3.1. Description

Radio-Frequency identification (RFID) is often used in IoT applications for identification [51]. When an RFID tag comes near an RFID card reader, a carrier signal is generated by the reader which is then sent to the tag. This tag then modulates the signal. The modulated signal is interpreted by the reader again which is interfaced to a microcontroller (ESP32). The microcontroller receives this data and compares it to the data in its database. If the result matches it will return a positive response. The working of an RFID card reader has been shown in Figure 9.

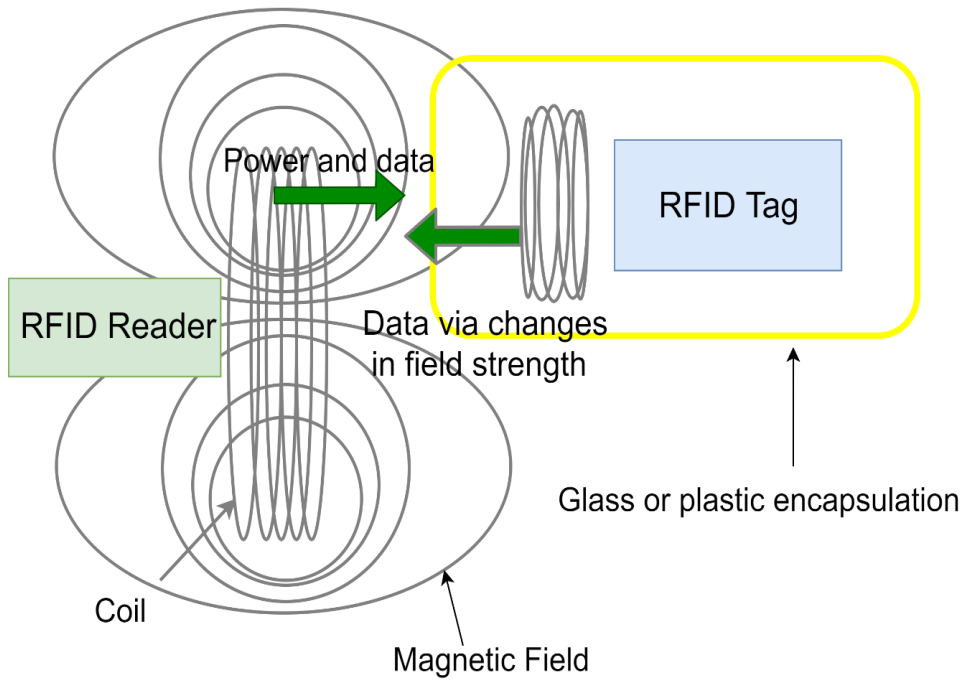


Figure 9: Working of an RFID card reader

The architecture of an RFID card reader as shown in Figure 10 is very similar to the PIR motion sensor. The only difference is that the PIR sensor is replaced by an RFID card reader and the microcontroller receives 10 digits instead of signal for motion.

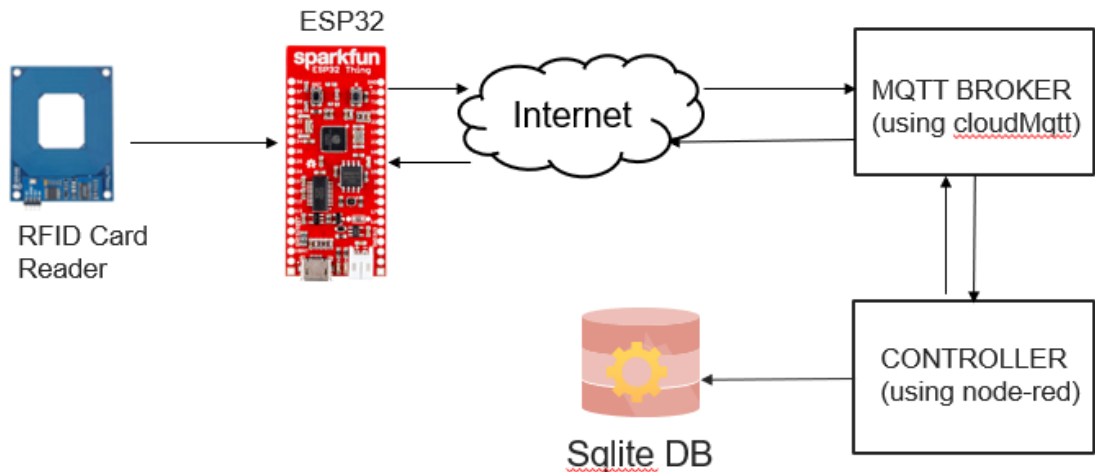


Figure 10: IoT architecture for an RFID Card Reader

3.3.3.2. *RFID Card Reader timed extended FSM model.*

The working of the RFID card reader has been modeled as a timed extended Finite State Machine as shown in Figure 11.

Table 4 shows all the inputs and outputs in the state diagram. The state diagram assumes that the ‘valid’ number for the card is known in advance and hence represents a constant for the state diagram. This constant is used to determine if the 10-digit number received is a valid card or not. Most of the state diagram is like that for PIR motion sensor.

To describe Table 4 in detail, input i_1 has parameterized input of $ws(0)$ that indicates that WiFi is not connected and is trying to connect. The corresponding output for this is o_1 which is a message displayed on the output console that says, “Connecting to WiFi”. The input i_2 has parameterized input of $ws(1)$ that indicates that WiFi is connected. The corresponding output for this is o_2 which is a message displayed on the output console that says, “Connected to WiFi”. The input i_3 has parameterized input $cc(0)$ that indicates that MQTT queue (client) is not connected and is trying to connect. The corresponding output for this is o_3 which is a message displayed on the output console that says, “Connecting to MQTT”. The input i_4 is a parameterized input indicated with $cc(1)$ that indicates that MQTT queue (client) is connected. The corresponding output for this is o_4 which is a message displayed on the output console that says, “Connected to MQTT”. The next set of inputs are events ($i_5, i_6, i_7, i_8, i_9, i_{10}$) from the application server. The input i_5 sends a message to do a health check on the RFID Card reader and corresponding output o_5 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Sensor is Alive”. Similarly, the input i_6 is an invalid request (A random string) and the corresponding output received for it is an MQTT message displayed on Node-Red (MQTT display server) that says “Error. Invalid message”. The input i_7 is an input requesting to stop the publishing of the messages on the MQTT queue indicated by the String “Stop Publishing” on the RFID Card Reader and corresponding output o_7 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Publishing has stopped”. This message is displayed only after the request for Stop Publish has been processed. The input i_8 is modeled to ask for specific information from the device. In our model we have used it to ask for the WiFi name on which the device is connected. Once this request is processed, the output o_{11} displays the `device_ssid` on the Node-Red

display screen. The next input i_9 is generated whenever a button is pressed on the edge device. This type of input has been considered if there is a need for an event generated by a trigger in real time. The output for this event is o_8 which will display the message that the “Button has been pressed”. The input i_{10} is an input requesting to start the publishing of the messages on the MQTT queue indicated by the String “Start Publishing” on the RFID Card Reader and corresponding output o_9 received for it is an MQTT message displayed on Node-Red (MQTT display server) that the “Publishing has started again”. This message is displayed only after the request for “Start Publish” has been processed.

The primary difference of the RFID Card reader in comparison with the motion sensor and ultrasonic motion sensor is reflected in states S_{10} , S_{11} , and S_{12} . The data byte “\$0A” triggers the start of reading the 10 digits from an RFID card. Subsequent characters are read in state S_{11} (bit by bit). Each time a character is received, it gets appended the existing string and the counter is incremented by 1. This process is repeated until it receives all the 10 digits. If the counter < 10, for every input i_{11} , the corresponding output o_{13} will be “Card is reading data bytes”. If the counter = 10, but the stop byte has not been received, the output will still be o_{13} displaying the message “Card is reading data bytes”. When the counter reaches 10 [$counter = 10$], the microcontroller goes into state S_{12} to wait for the stop byte. If it receives the stop byte, it compares if the received digits are same as the digits of the card in the database. Depending on if the card is acceptable [$code = card$], the microcontroller moves back to state S_{10} with different messages after resetting $counter$ to 0. If [$code=card$] then the output received is o_{15} (“Card Accepted”). However if [$code!=card$], then the output received is o_{14} (“Card rejected”). For our experiment, we have pre-defined an array of codes that are accepted cards. Whenever a new card is scanned, the code is compared with this set. If the code matches the one of the codes defined in the array, then the message displayed on the console is “Card Accepted”. If the code doesn’t match, it will send a message “Card Rejected”.

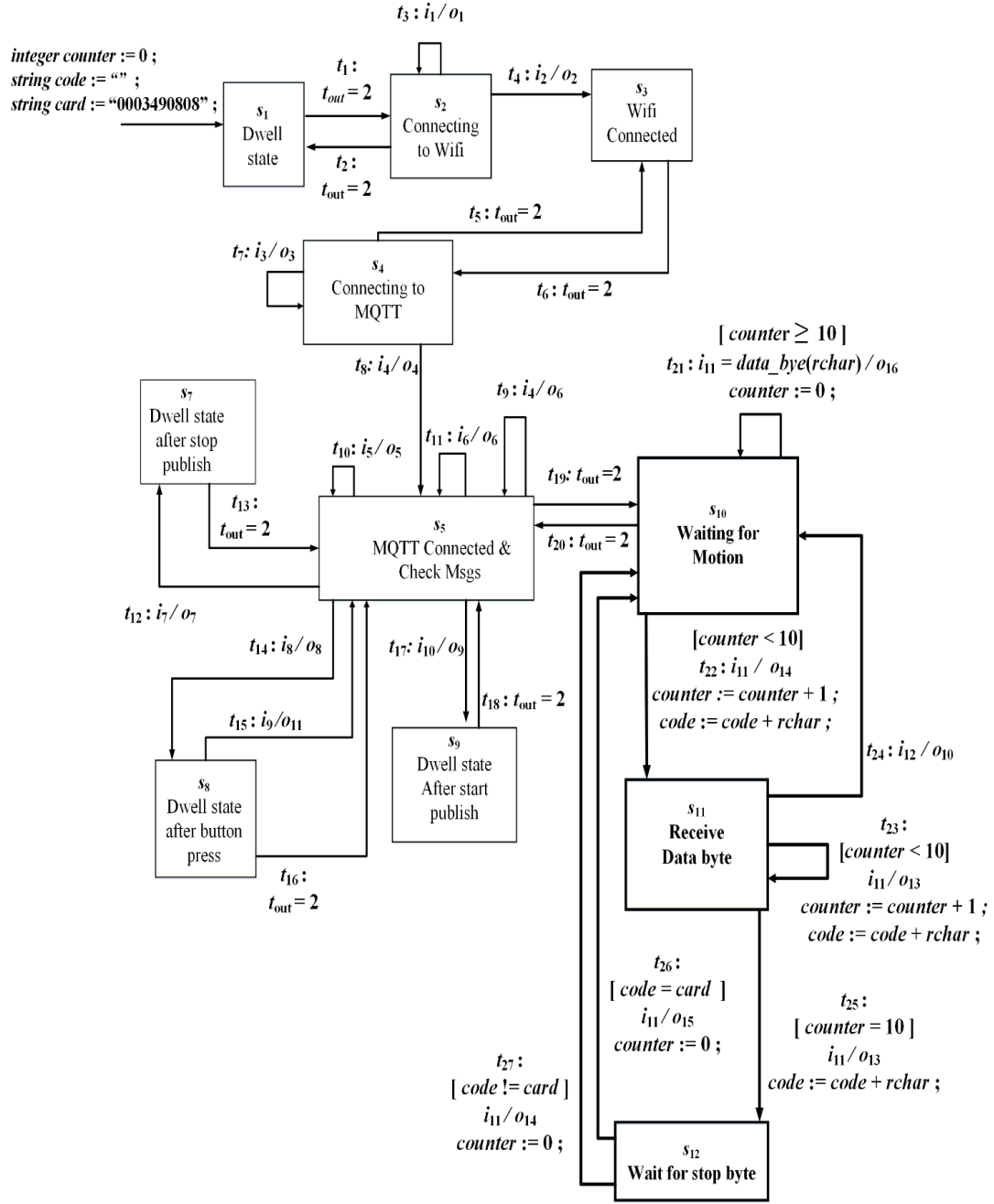


Figure 11: Timed extended FSM of the RFID Card Reader.

Table 4 provides a summary of the inputs, outputs, and transitions depicted in the figure above. In contrast to the motion sensor, the RFID card reader model incorporates variables and parameterized input. This feature proves beneficial in accommodating various types of devices with diverse operational mechanisms.

Table 4: Description of the inputs and outputs of the RFID Card Reader

Variables:			
code(string)			
card = "0003490808" (a constants of type string representing the card number) – In reality it will be variable, but for this purpose we are keeping it constant (preprogrammed to only accept one card)			
counter (integer),			
<i>Parameterized input:</i> $i_{11} = data_byte(rchar)$ with parameter <i>rchar</i> of type String			
Input		Output	
i_1	ws(0)	o_1	“Connecting to wifi”
i_2	ws(1)	o_2	“Connected to wifi”
i_3	cc(0)	o_3	“Connecting to MQTT”
i_4	cc(1)	o_4	“Connected to MQTT”
i_5	“Health Check”	o_5	“Sensor is Alive”
i_6	“Unexpected Request”	o_6	“Error. Invalid message”
i_7	“Stop Publishing”	o_7	“Publishing has stopped”
i_8	“Give me wifi Name”	o_8	“Press any button”
i_9	buttonPress(low)	o_9	“Publishing has started again”
i_{10}	“Start Publishing”	o_{10}	“Start byte received”
i_{12}	“\$0D”	o_{11}	device_ssid
		o_{12}	“Request coming from invalid IP”
		o_{13}	“Card is reading data bytes”
		o_{14}	“Card rejected”
		o_{15}	“Card accepted”
		o_{15}	“Null”

3.4. IOT Attacks

3.4.1. Security Issues in IoT Devices

Table 5 summarizes some common security issues in IoT devices addressed in our thesis[15] [52]. The modeling of each attack is described below.

Table 5: Types of attacks in IoT devices

Input	Type of Attack	Description
A ₁	Battery Draining	Forcing the edge device to execute power-consuming subroutines.
A ₂	Sleep Deprivation	Preventing the node from going to sleep.
A ₃	Data falsification	In this type of attack the information known about a systems operation allows the attacker to falsify data by planting malicious nodes or turning a normal node into a malicious node
A ₄	Replay attack	A Replay Attack is made by spoofing, altering, or replaying the identity information of smart devices in the IoT network
A ₅	Man in the middle	In this attack, an attacker or hacker intercepts communication between two systems.

3.4.2. Attacks Modeled as state machine mutants

In previous sections, we modelled the behaviour of the motion sensor, ultrasonic motion sensor, and RFID card reader as timed Finite State Machines and introduced common types of IoT device attacks. In this section, we show how these attacks can be represented/modelled as mutants of the considered state machines. This allows the derivation of tests that detect such attacks.

3.4.2.1. Battery Draining Attack

At some states, decreasing the value of t_{out} forces the machine to leave the state early decreasing the possibility of providing the user with a desired output, i.e., forcing the user to resend the intended request multiple times. This causes the battery to drain. In our case, mutants with such anomaly can be derived by decreasing the t_{out} period at appropriate states. For the motion sensor, we consider states S_5 , S_7 , S_8 , S_9 , and S_{10} , then for each state, the timeout period of the state is decreased from 2 to 1 second, creating a corresponding mutant with timeout fault. Figure 12 shows normal transition for motion sensor between S_5 and S_8 whereas Figure 13 shows a mutated transition. Similarly, for the ultrasonic, the timeout period at states S_5 , S_7 , S_8 , S_9 , S_{10} and S_{12} are

changed in creating the collection of related mutants, and for RFID timeouts at states S_5 , S_7 , S_8 , S_9 , and S_{10} , are changed creating the related collection. In total, 5, 6, and 5 mutants are created for the motion sensor, ultrasonic sensor, and the RFID, respectively.

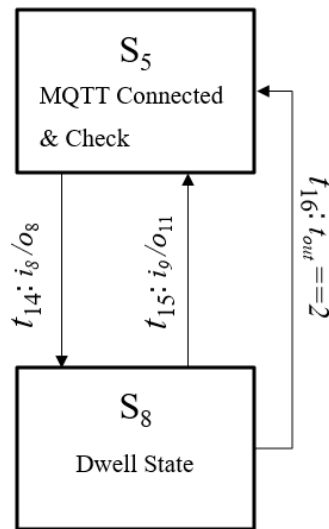


Figure 12: Part of the timed FSM of the Motion Sensor (Normal transition)

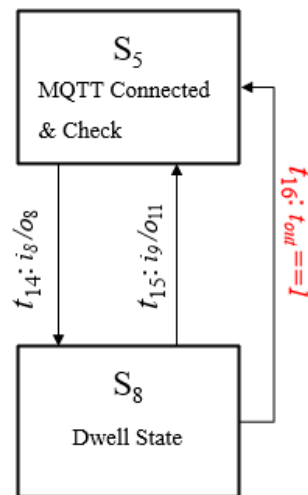


Figure 13: Mutated transition for battery draining attack.

3.4.2.2. Sleep Deprivation Attack

This type of attack can be simulated by a mutant that prevents the device from sleeping at certain states. For instance, in the motion sensor as shown in Figure 14, at state S_{10}

(Waiting for motion) the device wakes up after every 2 seconds to check if there is motion and then goes back to sleep. For ultrasonic motion sensor it happens at the state S_{10} and for the RFID reader this can happen at state S_{10} (Ready to Read). In each case, we derive the corresponding mutant by reducing the timeout at these states from 2 to 0.001 seconds as shown in Figure 15. Though the reduction of the tout time can be done at every state; yet the above selected states are the most relevant to sleep deprivation attacks.

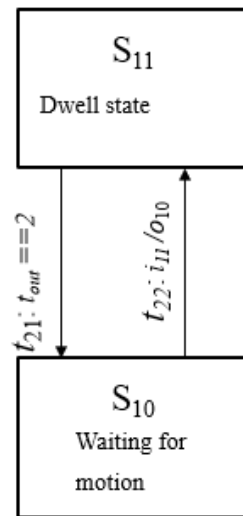


Figure 14: Part of the timed FSM of the Motion Sensor (Normal transition)

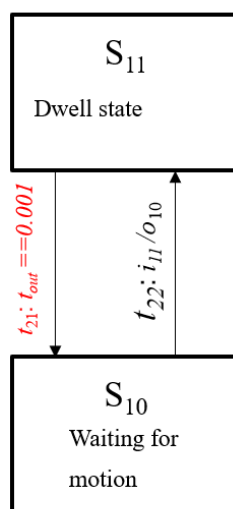


Figure 15: Mutated transition for sleep deprivation attack

3.4.2.3. Data Falsification Attack

This attack can be simulated by changing the output of certain transition at certain state (i.e., by creating a mutant with an output fault). For example, if the sensor is supposed to reply (output) the health status, it would reply with an error message instead. For the motion sensor, this can happen by replacing the output to o_6 by another output at transitions t_7 , t_8 , t_{12} , t_{14} , t_{15} , t_{17} , and t_{22} . For each replaced output, the corresponding mutant is created. This scenario has been illustrated in Figure 17. It can be compared to a normal transition as shown in Figure 16. Similarly, for the ultrasonic, this can happen by changing the output to o_6 at transitions t_7 , t_8 , t_{12} , t_{14} , t_{15} , t_{17} , t_{21} and t_{22} . For the RFID, output o_6 is changed at transitions t_7 , t_8 , t_{12} , t_{14} , t_{15} , t_{17} , t_{22} , t_{23} , t_{24} , t_{25} , t_{26} , and the corresponding mutants are included in a related manner. In total, 7, 8, and 11 mutants are created considering for the motion sensor, ultrasonic sensor, and the RFID, respectively.

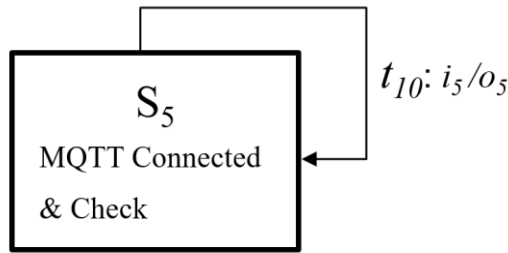


Figure 16: Part of the timed FSM of the Motion Sensor (Normal transition)



Figure 17: Mutated transition for data falsification attack for motion sensor

3.4.2.4. Replay Attack

In the motion sensor, input i_4 at transition t_8 takes the machine from state S_4 (Connecting to MQTT) to S_5 (Connected to MQTT) while providing the output o_4 (Connected to MQTT). A part of this transition has been shown in Figure 18. This indicates that the connection has been established; and then afterwards, if the input i_4 is provided at state

S_5 , the machine responds with the output o_6 . However, once the connection is established, an attacker can simulate again at state S_5 the behaviour of t_8 ; this can be done by creating a mutant by adding as a self-loop transition at S_5 with the same input and output as that of t_8 . Figure 19 shows the mutated transition. The same can be replicated for the ultrasonic motion sensor and RFID card reader at the same state S_5 .

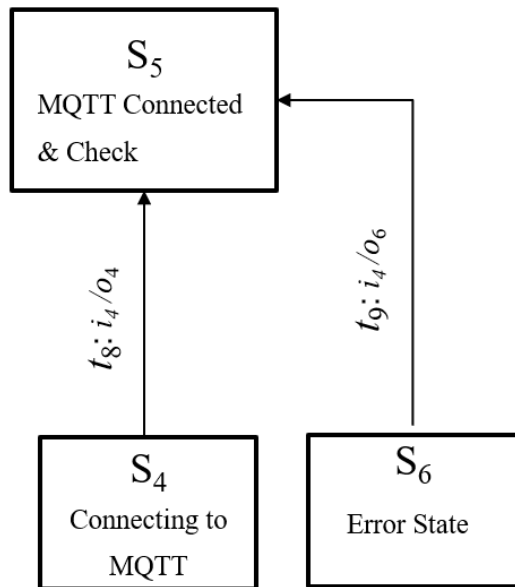


Figure 18: Part of the timed FSM of the Motion Sensor (Normal transition)

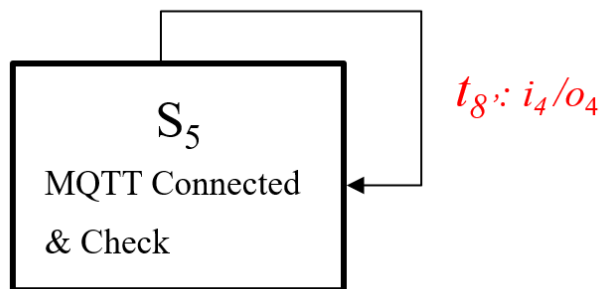


Figure 19: Mutated transition for replay attack for motion sensor

3.4.2.5. *Man in the Middle Attack*

For the motion sensor, at state S_{10} (Waiting for motion), if the input i_{11} (Motionstatus(1)) is applied the machine outputs o_{10} (Motion has been detected), and moves to a Dwell state (S_{11}). Figure 20 shows the normal transition between state S_{10}

and S_{11} . To model man in the middle, a new state S_{new} is added such that i_{11} takes the machine from S_{10} to S_{new} instead of the Dwell state S_{11} . A timeout transition $t_{out} = 2$ is also added connecting S_{new} back to S_{10} ; thus, preventing the user to reach S_{11} if the user does not provide within 2 seconds the input i_{11} again. S_{new} takes this information and passes it on to S_{11} . This scenario has been illustrated in Figure 21. This type of attack can happen at many states; however, we have kept the most relevant states for this type of attack as knowing the status of the motion in a sensor would be the most valuable information that an attacker would have. However, this S_{new} state can be easily replicated between states S_5 and S_7 , S_5 and S_8 , S_5 and S_9 for all 3 devices: the motion sensor, the RFID card reader, and the ultrasonic motion sensor as well.

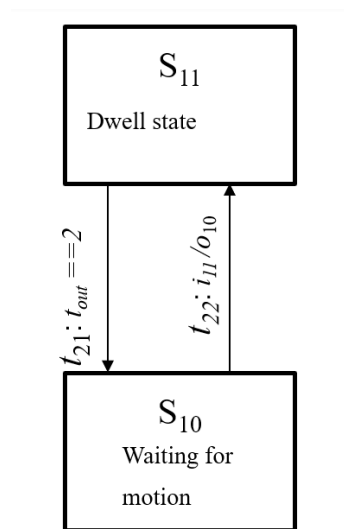


Figure 20: Part of the timed FSM of the Motion Sensor (Normal transition)

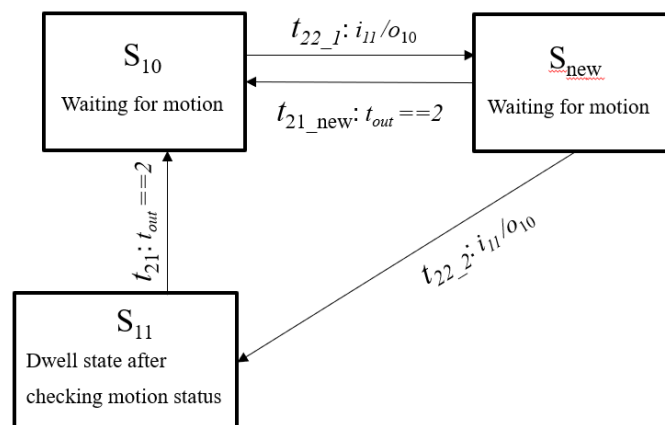


Figure 21: Mutated transition for replay attack for motion sensor

3.4.2.6. Mutants with increased timeout for both battery draining and sleep deprivation.

At some states, we increased the value of t_{out} such that the device spends more time in performing the operations and would let to spend more time in the dwell state. This causes the device to spend more time in each of the states. Mutants with such anomaly can be derived by increasing the t_{out} period at appropriate states. For the motion sensor, we consider states $S_5, S_7, S_8, S_9,$ and S_{10} , then for each state, the timeout period of the state is increased from 2 seconds to 5 seconds, creating a corresponding mutant with timeout fault. Similarly, for the ultrasonic, the timeout period at states $S_5, S_7, S_8, S_9, S_{10}, S_{12}$ are changed in creating related collection of mutants, and for RFID Card Reader timeouts at states $S_5, S_7, S_8, S_9,$ and S_{10} are changed for creating related mutants. In total, 5, 6, and 5 mutants are created for the motion sensor, ultrasonic sensor, and the RFID Card Reader, respectively.

Table 6 summarizes the threat mutants generated for all the security attacks covered in the scope of our work. Table 7 summarizes all the modeled mutants created at different transitions and states for motion sensor. Table 8 summarizes all the modeled mutants created at different transitions and states for ultrasonic motion sensor. Table 9 summarizes all the modelled mutants created at different transitions and states for RFID Card Reader.

Table 6: Summary of generated security threat mutants

Type of attack	Motion Sensor	RFID Card Reader	Ultrasonic motion sensor
Battery Draining Attack	5	5	6
Sleep deprivation Attack	1	1	1
Data Falsification Attack	7	11	8
Replay Attack	1	1	1
Man in the Middle Attack	1	1	1

Table 7: Summary of modeled mutants for the motion sensor

Type of attack	Mutants
Data Falsification Attack	M ₁ : at transition t ₇ , o ₃ is changed to o ₆
	M ₂ : at transition t ₈ , o ₄ is changed to o ₆
	M ₃ : at transition t ₁₂ , o ₇ is changed to o ₆
	M ₄ : at transition t ₁₄ , o ₈ is changed to o ₆
	M ₅ : at transition t ₁₅ , o ₁₂ is changed to o ₆
	M ₆ : at transition t ₁₇ , o ₁₀ is changed to o ₆
	M ₇ : at transition t ₂₂ , o ₁₁ is changed to o ₆
Man in the Middle Attack	M ₁ : added state after state S ₉
Battery Draining Attack	M ₁ : S ₅ , Timeout at transition t ₁₉ changed from 2 to 1s
	M ₂ : S ₇ , Timeout at transition t ₁₃ from 2 to 1s
	M ₃ : S ₈ , Timeout at transition t ₁₆ from 2 to 1s
	M ₄ : S ₉ , Timeout at transition t ₁₈ from 2 to 1s
	M ₅ : S ₁₀ , Timeout at transition t ₂₀ from 2 to 1s
Sleep Deprivation Attack	M ₁ : Timeout at transition t ₂₀ decreased from 2s to 0.001s
Replay Attack	Mutant_RA ₁ : t ₉ output fault to o ₄

Table 8: Summary of modeled mutants for the ultrasonic motion sensor

Type of attack	Mutants
Data Falsification Attack	M ₁ : at transition t ₇ , o ₃ is changed to o ₆
	M ₂ : at transition t ₈ , o ₄ is changed to o ₆
	M ₃ : at transition t ₁₂ , o ₇ is changed to o ₆
	M ₄ : at transition t ₁₄ , o ₈ is changed to o ₆
	M ₅ : at transition t ₁₅ , o ₁₂ is changed to o ₆
	M ₆ : at transition t ₁₇ , o ₁₀ is changed to o ₆
	M ₇ : at transition t ₂₁ , o ₁₀ is changed to o ₆
	M ₈ : at transition t ₂₂ , o ₁₁ is changed to o ₆
Man in the Middle Attack	M ₁ : added state after state S ₉
Battery Draining Attack	M ₁ : S ₅ , Timeout at transition t ₁₉ changed from 2 to 1s
	M ₂ : S ₇ , Timeout at transition t ₁₃ from 2 to 1s
	M ₃ : S ₈ , Timeout at transition t ₁₆ from 2 to 1s
	M ₄ : S ₉ , Timeout at transition t ₁₈ from 2 to 1s
	M ₅ : S ₁₀ , Timeout at transition t ₂₀ from 2 to 1s
	M ₆ : S ₁₂ , Timeout at transition t ₂₃ from 2 to 1s
Sleep Deprivation Attack	M ₁ : Timeout at transition t ₂₃ decreased from 2s to 0.001s
Replay Attack	Mutant_RA ₁ : t ₉ output fault to o ₄

Table 9: Summary of modeled mutants for the RFID Card Reader

Type of attack	Mutants
Data Falsification Attack	M ₁ : at transition t ₇ , o ₃ is changed to o ₆
	M ₂ : at transition t ₈ , o ₄ is changed to o ₆
	M ₃ : at transition t ₁₂ , o ₇ is changed to o ₆
	M ₄ : at transition t ₁₄ , o ₈ is changed to o ₆
	M ₅ : at transition t ₁₅ , o ₁₁ is changed to o ₆
	M ₆ : at transition t ₁₇ , o ₉ is changed to o ₆
	M ₇ : at transition t ₂₂ , o ₁₄ is changed to o ₆
	M ₈ : at transition t ₂₃ , o ₁₃ is changed to o ₆
	M ₉ : at transition t ₂₄ , o ₁₀ is changed to o ₆
	M ₁₀ : at transition t ₂₅ , o ₁₃ is changed to o ₆
	M ₁₁ : at transition t ₂₆ , o ₁₅ is changed to o ₆
Man in the Middle Attack	M ₁ : added state after state S ₉
Battery Draining Attack	M ₁ : S ₅ , Timeout at transition t ₁₉ changed from 2 to 1s
	M ₂ : S ₇ , Timeout at transition t ₁₃ from 2 to 1s
	M ₃ : S ₈ , Timeout at transition t ₁₆ from 2 to 1s
	M ₄ : S ₉ , Timeout at transition t ₁₈ from 2 to 1s
	M ₅ : S ₁₀ , Timeout at transition t ₂₀ from 2 to 1s
Sleep Deprivation Attack	M ₁ : Timeout at transition t ₂₀ decreased from 2s to 0.001s
Replay Attack	Mutant_RA ₁ : t ₉ output fault to o ₄

3.4.3. Physical Device Attacks

One of the critical issues about edge devices is that they can be physically tampered. There is a need to detect if the device has been compromised physically. For example, the attacker can just remove the edge device and replace it with its own or the attacker can just disconnect the device from the system so that it stops sending signal. First, the code is tested against the mutant's behaviour and checked that it works well and thus can detect all the mutants. Then, we replicated the working of a motion sensor on a second ESP which is connected to the first ESP. Whenever we want to send an input that is like the one generated from the motion sensor, we simulate it from the second ESP instead. Once it is in complete working state, we inject (the considered attack) faults into this replica and analyse this behaviour. Since we have complete control on the simulation, we can look at a step-by-step analysis of where exactly the system fails if a fault is detected. It is not just the regular behaviour of the replica that can help us in the analysis, we also analyse the behaviour when there are faults introduced. Thus, if the replica does not respond like the motion sensor to the test cases; we conclude that this replica is not the original sensor and thus it might have been physically tampered

and replaced. Also, another way to check for a physical device compromise is by injecting faults on a compromised device and compare if the abnormal behaviour of the compromised device is the same as the abnormal behaviour of a replica. The comparison of these two can give us more insights and help us conclude if the device has been physically changed. At some states, we increased the value of t_{out} such that the device spends more time in performing the operations and would let to spend more time in the dwell state. This causes the device to spend more time in each of the states. Mutants with such anomaly can be derived by increasing the t_{out} period at appropriate states. For the motion sensor, we consider states S_5 , S_7 , S_8 , S_9 , and S_{10} , then for each state, the timeout period of the state is increased from 2 seconds to 5 seconds, creating a corresponding mutant with timeout fault. Similarly, for the ultrasonic, the timeout period at states S_5 , S_7 , S_8 , S_9 , S_{10} , S_{11} , and S_{12} are changed in creating related collection of mutants, and for RFID Card Reader timeouts at states S_5 , S_7 , S_8 , S_9 , and S_{10} are changed for creating related mutants. In total, 5, 7, and 8 mutants are created for the motion sensor, ultrasonic sensor, and the RFID Card Reader, respectively.

3.5. Proposed Testing Framework

Figure 22 shows the experimental setup including various components and how they communicated with each other. As show in Figure 22, the sensors/actuators (motion sensor or an RFID Card Reader) are connected to the edge device (ESP32) which in turn is connected to the MQTT broker using WiFi. The messages sent and received from the broker can be viewed by the Middleware which was implemented using Node-Red.

CloudMQTT was used for creating the MQTT queue the messages are monitored using Node-Red with MQTT QoS level 2. The edge devices have been programmed in a way that correspond to the modeled FSMs that are shown in the above sections. When the device is running, an input is given via the Node-red interface, which is same as an input given to a state of the FSM.

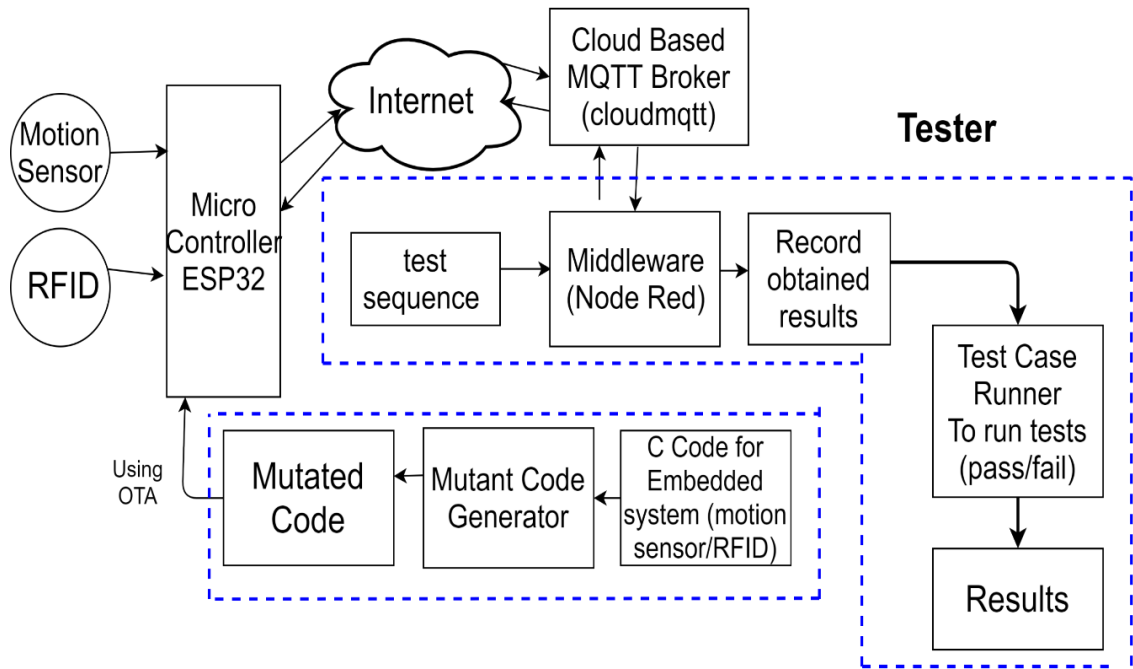


Figure 22: Experimental setup

3.5.1. The Edge Device

The sensors/actuators used for this experiment were PIR motion sensor and Parallax RFID Card Reader. We used the *The SparkFun ESP32 Thing* Microcontroller as the edge device because of its in-built WiFi and OTA capability. The hardware specification for ESP32 is given in Table 10.

Table 10: Hardware specification for the edge device

Specification	SparkFun ESP32 Thing
Processor	Dual-core Tensilica LX6 microprocessor
SRAM	520kB
Networking	802.11 b/g/n/e/i
Storage	4MB Flash memory
I/O	28 GPIO
On-board Peripherals	LED PWM

3.5.2. Data collection of the device behaviour

After the input is sent to the edge device using an MQTT message, the edge device responds with an output using MQTT, which is eventually recorded using the Node-red Middleware. The outputs have also been programmed in a way that corresponds to the

states of the FSM. For example, if the input given to the motion sensor is asking for the health status, the message is sent via MQTT on the Node-red interface to the sensor. The sensor's response that it is alive is also recorded using MQTT and the result is displayed on Node-red. The behaviour and the input/output sequences are then recorded and stored into a database for further analysis.

3.5.3. Injecting the mutants into the microcontroller

3.5.3.1. Mutants Generator

We generate the mutants that must be injected into the microcontroller in two different ways. The first type of mutants are the code mutants that are generated by creating variations of the C++ code written for the working of the edge devices. The mutant code generator takes the C++ code files and creates mutated code files which contains one mutant per file. This was implemented using Python and C++ that runs a program to scan the C++ code and then programmatically tweak/change the input file (a variable change, alter an arithmetic operator, comment a line, etc.). The mutated file is regenerated and saved in a folder. This program runs several times based on the possibilities found for code alteration and creates a separate mutated file with one mutant in each. The security mutants were created manually, by changing the code at some specific places that correspond to the security attacks mentioned in the above sections.

3.5.3.2. OTA Module

While the system is running under the normal circumstances, the attacks are done in real time. Hence to keep our testing scenarios as real as possible, we wanted to inject the mutants while the system is up and running. This was done using the Over the Air (OTA) Transfer functionality provided by Arduino. While the system is running using the original code, we take the mutated C file and inject it into the hardware using the OTA functionality. Using this method, the mutated program file is uploaded to the edge device using a WiFi connection instead of serial port. So, while the system is still running, the mutated file is uploaded and accepted by the system without stopping the device. The system now takes the mutated code, and the inputs are provided by the MQTT queue like it would under a normal environment. The outputs of these inputs are captured and saved to see the impact of the mutated code on the results.

3.5.4. The Testing Framework

The framework includes test derivation, deployment, execution, and analysis. Tests are derived considering the IoT specification machines and their corresponding mutants. The tests are then deployed via MQTT(Node-red) on the ESP32. The response from the devices with respect to the inputs of the test cases are captured again via the MQTT and stored in the SQL database. Tests are run (test case runner) one after the other and the corresponding output sequences are recorded and consequently a verdict is issued whether the test kills the considered mutant or not by comparing the actual output response of the implementation to the expected response of the test case. The process repeats considering the mutants one after the other.

3.5.4.1. Tests for detecting security attacks.

In general, deriving a test that can distinguish a given state machine from a given (attack) mutant can be carried out using the traditional methods elaborated in [43][44]. However, in our case, as the number of mutants is rather small, we derive these tests by hand. That is, for each attack type, we consider all mutants in that category one after the other; then for each mutant, we derive the test suite that kills the mutants. The input sequence of the test is then run against all other unconsidered mutants to identify those that can also be killed by the test. This reduces the number of derived tests.

Here we include an example of such test cases using the motion sensor. Consider the scenario for attack A_3 mentioned in Table 5. A mutant M_{A_3} of the state machine in Figure 6 is created such that the output of transition t_{11} is changed to o_6 at S_5 . The test case $(i_5, t=1)/\epsilon (i_5, t=2)/o_5$ detects such a fault as the expected behaviour according to the specification is $(i_5, t=1)/\epsilon (i_5, t=2)/o_5$. That is after applying the input i_5 at $t=1$ then applying again i_5 at $t=2$, the mutant responds to the second input with the output o_6 while the expected output is o_5 . Table 11, 12 and 13 summarizes the test cases generated for motion sensor, ultrasonic motion sensor and RFID Card Reader respectively.

Table 13: Test cases for RFID Card reader

Case	Test Case sequence
TC ₁	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_5, t=1/o_5)S_5$
TC ₂	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_6, t=1/o_6)S_5$
TC ₃	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (\varepsilon, t=1, \varepsilon)S_{10}; (i_{11}, t=1/o_{14})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (i_{12}, t=1, o_{13})S_{12}$
TC ₄	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (\varepsilon, t=1, \varepsilon)S_{10}; (i_{11}, t=1/o_{14})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (i_{12}, t=1, o_{13})S_{12};$
TC ₅	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (\varepsilon, t=1, \varepsilon)S_{10}; (i_{11}, t=1/o_{14})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (rchar, t=1, o_{13})S_{11}; (i_{12}, t=1, o_{13})S_{12}; (i_{11}, t=1, o_{14})S_{10}$
TC ₆	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_8, t=1/o_8)S_8; (i_9, t=1/o_{11})S_5$
TC ₇	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_8, t=1/o_8)S_8; (\varepsilon, t=1, \varepsilon)S_8; (\varepsilon, t=2, \varepsilon)S_5$
TC ₈	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_4, t=1/o_6)S_5$
TC ₉	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_4, t=1/o_6)S_5; (i_4, t=2/o_6)S_5$
TC ₁₀	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (\varepsilon, t=1, \varepsilon)S_{10}; (\varepsilon, t=2, \varepsilon)S_5$
TC ₁₁	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5; (i_5, t=1/o_5)S_5; (\varepsilon, t=1, \varepsilon)S_{10}$
TC ₁₂	$S_1(\varepsilon, t=2, \varepsilon)S_2; (i_2, 1/o_2)S_2; (i_1, t=1/o_1)S_3; (\varepsilon, t=2, \varepsilon)S_4; (i_3, t=1/o_3)S_4; (i_4, t=1/o_4)S_5;$

3.5.4.2. Tests for detecting actual device manipulation

The test cases remain the same for testing the physical device. For example, the attacker can just remove the edge device and replace it with its own. In this situation, the system will behave normally. We can test this suspicious device by our test cases that we have created for all states. However, there is a possibility, that even then the system could pass all the test cases correctly. The mechanism can be reverse engineered by injecting faults/mutants in the suspicious device. We already know how a device behaves when subjected to different types of faults and we have those results. With these results, how would the suspicious device behave when subjected to those exact same faults. Is the behaviour of the suspicious device same as compared to a normal device under a faulty environment? Does it break in the same way that the normal device would break? Does it produce the same kind of error that a normal device had produced when it was tested with these faults? Hence detecting the physical manipulation cannot just be verified with the correct behaviour of the system, but it can also be detected by the incorrect behaviour.

3.5.4.3. Tests for detecting code mutants

These mutants are generated using a tool that creates changes to the code in different ways (changing the assignment operators, changing the variable names, variable value, etc.). To generate code mutants, the program written for the workflow shown in Figure 2 is fed into a mutant tool generator. The mutant tool generator creates different files with one mutant each. For Example: If the program has a line written as `String msg1 = "Health Check"`, then its mutated file would have the line written as `String msg1 = "&Health Check"`.

3.5.5. Implementation tools

The framework explained in the above sections has been implemented with different interfaces and tools as shown in Table 14.

Table 14: Implementation tools and process automation

No	Implementation steps	Tools Used
1	Generating the mutants	Python and C++
2	Injecting it on the embedded system Over The air	Python scripts and Arduino-cli
3	Running the mutated code on the system	Arduino-cli
4	Capturing the behavior of the system for different inputs/outputs	MQTT, Node-red, sqlite scripts
5	Running the test cases on these values captured	Gtest and C++
6	Displaying the test result	C++
7	Automating Step 2-6	Bash Script

Chapter 4. Evaluation

In this chapter, we evaluate the proposed work from several perspectives.

4.1. Identify the coverage of security faults.

The proposed approach worked very well and resulted in a perfect mutation score of 100% of security attack code mutants. This performance can be compared to an alternative approach that does not consider the type of security threat but is based on known code-based mutants. For each device, code-based mutants that were generated using a mutant generator were injected in the hardware one by one in real time. Then the security test cases derived for each of these devices were run against each of the generated mutants.

For the each of the devices (motion sensor, ultrasonic motion sensor and RFID card reader) 40 code-based mutants using the automatically generated mutation operators from a tool [41] were used. Each of these mutants was then injected into the real hardware, and then the 12 security test cases are run against the mutant.

Table 15 describes the mutation score for code-based mutants for the motion sensor where $M_1, M_2, M_3, \dots, M_{40}$ indicate the 40 code-based mutants that were randomly generated using a mutant generator. $TC_1, TC_2, TC_3, \dots, TC_{12}$ indicate the test cases that were created for the security mutants. R indicates the result stating whether the code mutant was killed by the test case or not. If the table shows 1 against any of the test cases for Mutant M , that means that mutant was killed, otherwise it was not killed. Similarly, Table 16 describes the mutation score for the ultrasonic motion sensor and Table 19 describes the mutation score for the RFID Card Reader.

For example, in Table 15, the mutant M_2 was killed by TC_4 . The entry of M_2 against TC_4 has been marked with 1. Even if one test case kills a mutant, the R column has been marked with Y, indicating that the mutant was killed by at least one of the test cases. If no test case was able to detect the mutant, the R column has been left blank. The overall mutation score was derived based on the total number of mutants that were killed by the test cases.

Table 15: Mutation score for code-based mutants for motion sensor

	TC ₁	TC ₂	TC ₃	TC ₄	TC ₅	TC ₆	TC ₇	TC ₈	TC ₉	TC ₁₀	TC ₁₁	TC ₁₂	R
M ₁			1							1	1		Y
M ₂				1									Y
M ₃													
M ₄													
M ₅													
M ₆				1									Y
M ₇											1		Y
M ₈													
M ₉													
M ₁₀													
M ₁₁													
M ₁₂													
M ₁₃													
M ₁₄											1		Y
M ₁₅													
M ₁₆													
M ₁₇						1							Y
M ₁₈													
M ₁₉													
M ₂₀													
M ₂₁													
M ₂₂													
M ₂₃													
M ₂₄													
M ₂₅						1	1						Y
M ₂₆													
M ₂₇													
M ₂₈						1	1						Y
M ₂₉													
M ₃₀													
M ₃₁													
M ₃₂													
M ₃₃	1										1		Y
M ₃₄													
M ₃₅													
M ₃₆													
M ₃₇													
M ₃₈													
M ₃₉													
M ₄₀													

Table 16: Mutation score for code-based mutants for ultrasonic motion sensor

	TC ₁	TC ₂	TC ₃	TC ₄	TC ₅	TC ₆	TC ₇	TC ₈	TC ₉	TC ₁₀	TC ₁₁	TC ₁₂	R
M ₁			1							1	1		Y
M ₂													
M ₃													
M ₄													
M ₅													
M ₆				1									Y
M ₇											1		Y
M ₈													
M ₉													

M ₁₀													
M ₁₁													
M ₁₂													
M ₁₃													
M ₁₄													
M ₁₅													
M ₁₆													
M ₁₇						1							Y
M ₁₈													
M ₁₉													
M ₂₀										1			Y
M ₂₁													
M ₂₂													
M ₂₃													
M ₂₄													
M ₂₅						1	1						Y
M ₂₆													
M ₂₇													
M ₂₈						1	1						Y
M ₂₉													
M ₃₀													
M ₃₁													
M ₃₂													
M ₃₃	1										1		Y
M ₃₄													
M ₃₅													
M ₃₆													
M ₃₇													
M ₃₈													
M ₃₉													
M ₄₀													

Table 17: Mutation score for code-based mutants for RFID Card reader

	TC ₁	TC ₂	TC ₃	TC ₄	TC ₅	TC ₆	TC ₇	TC ₈	TC ₉	TC ₁₀	TC ₁₁	TC ₁₂	R
M ₁													
M ₂													
M ₃													
M ₄													
M ₅													
M ₆													
M ₇											1		Y
M ₈													
M ₉													
M ₁₀													
M ₁₁													
M ₁₂													
M ₁₃													
M ₁₄	1										1		Y
M ₁₅													
M ₁₆													
M ₁₇						1							Y
M ₁₈													
M ₁₉													
M ₂₀													
M ₂₁													

M ₂₂													
M ₂₃													
M ₂₄													
M ₂₅						1	1						Y
M ₂₆													
M ₂₇													
M ₂₈						1	1						Y
M ₂₉													
M ₃₀													
M ₃₁													
M ₃₂													
M ₃₃	1										1		Y
M ₃₄													
M ₃₅													
M ₃₆													
M ₃₇													
M ₃₈													
M ₃₉													
M ₄₀													

To summarize the 3 tables above, the mutation score for the motion sensor was 22.5 percent. For the ultrasonic motion sensor, we obtained a 20 percent mutation score. For the RFID Card Reader the mutation score was 15 percent. Thus, on average, 19 percent mutation score was obtained. This gives us an idea of the general fault coverage of security tests. In other words, in general, our considered security faults do not correspond to arbitrary mutants (faults) derived using the traditional mutation C testing operators; and thus, there is a need to consider specific tests for these attacks as we do in our implementation. Table 18 describes a summary of the mutation score with model based and code-based mutants. The modelled mutants were all killed by the test cases with a 100% mutation score for all the edge devices. In contrast, the test cases performed poorly when they were run against the code mutants.

Table 18: Summary of Mutation score for code and modeled mutants

	Motion sensor	RFID	Ultrasonic motion sensor
Modelled mutants	100%	100%	100%
Code mutants	22.5%	15%	20%

4.2. Assess impact of threat in battery drainage

We conducted an experiment to assess the severity of battery drainage attacks on the battery performance. A big factor contributing to draining is if we make the device

perform power consuming sub routines at shorter intervals and more often. We measured the power consumption of the device by injecting mutants that lead to battery drainage using an inline hardware device. The power consumption was calculated using 2 scenarios 1) When the system was running under normal conditions 2) when the system was running under a battery draining attack. The experiment ran for 30 minutes for each device collected at a frequency of every 3 seconds. The results are shown in Table 19.

Table 19: Power consumption and comparison for IoT devices under attack

Device	Energy (in watt-hour) normal conditions	Energy (in watt-hour) under battery draining attack	% increase in energy consumption
Motion Sensor	900 Watt-hour	1740 Watt-hour	48%
RFID Card Reader	1674 watt-hour	2909 Watt-hour	42%
Ultrasonic motion sensor	1152 watt-hour	2070 Watt-hour	44%

Summarizing the table above, there was a 48% increase in the energy consumption for motion sensor and 42% and 44% for RFID card reader and ultrasonic motion sensor respectively. Such an increase can be controlled if battery drainage attacks can be spotted in real time.

4.3. Feasibility of applying the work in practice

As the number of security tests per attack type is rather small, then it is reasonable for assessing a certain anomaly, to run related tests many times as needed without causing battery drainage. We conduct a simple experiment to assess the cost of running the tests on battery drainage. We measured the power consumption by running all the three edge devices for 30 minutes each (motion sensor, RFID card reader and ultrasonic motion sensor) using an inline hardware device to calculate the current. The power consumption was calculated using 2 scenarios 1) When the system was running under normal conditions and performing normal tasks 2) When the system was subjected to continuous testing – This means that the test cases ran on the system continuously for 30 minutes and the power consumption was calculated. The results are shown in Table

20. It is clear that the percentage of energy consumption was below 25% for all the 3 devices when the tests were being run continuously without a break. Additional modifications can be made to run the tests at regular intervals instead of running them continuously if the power consumption needs to be reduced further.

Table 20: Power consumption and comparison for IoT devices when running test cases.

Device	Energy (in watt-hour) normal conditions	Energy (in watt-hour) when the tests are continuously running	% increase in energy consumption
Motion Sensor	900 Watt-hour	1170 Watt-hour	23%
RFID Card Reader	1674 watt-hour	2106 Watt-hour	20.5%
Ultrasonic motion sensor	1152 watt-hour	1368 Watt-hour	15.7%

4.4. Applicability of the work to realistic systems

In this regard, experiments were conducted to determine if the security tests derived by our method could detect the considered anomalies in real implementations. In total, 29, 35, 32 attack mutants were derived for motion sensor, ultra-sonic motion sensor, and RFID, respectively. Each of these anomaly mutants were injected into the real hardware and our experiments showed that each of these anomalies was detected when running the corresponding test case on the real hardware. This model has certain limitations as well. If there is an attack that is outside the scope of the modelled behaviour, then the attack will not be detected. Additionally, the current research caters to only 5 types of security attacks (Battery draining, sleep deprivation, data falsification, Man in the middle and replay attack). However, in real time, IoT devices can be subjected to many other types of attacks that would not be detected by the proposed model. To detect the other attacks, we must model them as security mutants and then create test cases that will be able to detect them. Real time interruption such as network failure may also not be detected for our proposed model. Furthermore, our model cannot distinguish between false attack alarm.

Chapter 5. Conclusion and Future Work

To summarize the work done in this research, we modeled the behaviour of a motion sensor, ultrasonic motion sensor and RFID Card Reader as finite state machines with timeouts. We also modeled common IoT attacks such as battery draining, sleep deprivation, data falsification, replay, and man in the middle IoT as special mutants of these machines. Mutation testing is then used to derive tests that distinguish these threats from the common behaviour of the machines. The behaviour of these mutants is tested in real environment by running the tests on them. We also consider tests for detecting actual physical device manipulation. We implemented and assessed the work on a real environment. The environment includes related hardware architecture, an IoT framework for running a program on microcontroller, data collection for observing the behaviour of the program and a testing framework for the derivation and injection of mutants/faults into the program and for detection of these faults in practice. The impact of threats on battery drainage is assessed and it is shown to be quite high. On average it yields 44.6 percent increase in power consumption. However, according to our experiments, the security tests obtained by our method are few in numbers and thus can be run many times with limited overhead, on average we obtained 25 percent energy consumption overhead when continuously running tests run on the system. Another experiment showed that on an average, security tests have only 18 percent coverage of arbitrary code-based faults/mutants showing that security faults do not correspond to arbitrary code faults; and thus, there is a need to consider the tests we proposed in our thesis for detecting IoT threats.

As a part of our future work, a stronger test suite can be built that is able to detect more security-based attacks, i.e., consider more types of attacks. Another interesting direction is to assess the work on more IoT devices and conduct relevant case studies. Another area for future work is creating a digital twin of the edge device to get more information on every state. Since the entire workflow is modelled and implemented as a finite state machine, every step in the code can be considered as a state. Whenever a test case fails, the digital twin should be able to give more information by pinging the actual device and gathering data at every state. This can help in localizing the exact place of the fault occurrence.

References

- [1] D. Mocrii, Y. Chen, and P. Musilek, "IoT-based smart homes: A review of system architecture, software, communications, privacy and security," *Internet Things*, vol. 1–2, pp. 81–98, Sep. 2018, doi: 10.1016/j.iot.2018.08.009.
- [2] M. Jia, A. Komeily, Y. Wang, and R. S. Srinivasan, "Adopting Internet of Things for the development of smart buildings: A review of enabling technologies and applications," *Autom. Constr.*, vol. 101, pp. 111–126, May 2019, doi: 10.1016/j.autcon.2019.01.023.
- [3] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Comput. Ind.*, vol. 101, pp. 1–12, Oct. 2018, doi: 10.1016/j.compind.2018.04.015.
- [4] H. Arasteh *et al.*, "Iot-based smart cities: A survey," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, Jun. 2016, pp. 1–6. doi: 10.1109/EEEIC.2016.7555867.
- [5] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming," *IEEE Access*, vol. 7, pp. 156237–156271, 2019, doi: 10.1109/ACCESS.2019.2949703.
- [6] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, "A Review of Machine Learning and IoT in Smart Transportation," *Future Internet*, vol. 11, no. 4, Art. no. 4, Apr. 2019, doi: 10.3390/fi11040094.
- [7] M. Ben-Daya, E. Hassini, and Z. Bahrour, "Internet of things and supply chain management: a literature review," *Int. J. Prod. Res.*, pp. 1–24, 2017.
- [8] B. Farahani, F. Firouzi, and K. Chakrabarty, "Healthcare IoT," in *Intelligent Internet of Things: From Device to Fog and Cloud*, F. Firouzi, K. Chakrabarty, and S. Nassif, Eds. Cham: Springer International Publishing, 2020, pp. 515–545. doi: 10.1007/978-3-030-30367-9_11.
- [9] M. Frustaci, P. Pace, G. Aloï, and G. Fortino, "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2483–2495, Aug. 2018, doi: 10.1109/JIOT.2017.2767291.
- [10] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1606–1616, Apr. 2019, doi: 10.1109/JIOT.2018.2847733.
- [11] M. Antonakakis *et al.*, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2017, pp. 1093–1110. Accessed: Jun. 22, 2019. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>
- [12] L. Prathibha and K. Fatima, "Exploring Security and Authentication Issues in Internet of Things," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, Jun. 2018, pp. 673–678. doi: 10.1109/ICCONS.2018.8663111.
- [13] Z. Mohammad, T. A. Qattam, and K. Saleh, "Security Weaknesses and Attacks on the Internet of Things Applications," in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, Apr. 2019, pp. 431–436. doi: 10.1109/JEEIT.2019.8717411.

- [14] Z. Shouran, A. Ashari, and T. Priyambodo, "Internet of Things (IoT) of Smart Home: Privacy and Security," *Int. J. Comput. Appl.*, vol. 182, pp. 3–8, Feb. 2019, doi: 10.5120/ijca2019918450.
- [15] Y. Lu and L. D. Xu, "Internet of Things (IoT) Cybersecurity Research: A Review of Current Research Topics," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2103–2115, Apr. 2019, doi: 10.1109/JIOT.2018.2869847.
- [16] M. Schiefer, "Smart Home Definition and Security Threats," in *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*, 2015, pp. 114–118. doi: 10.1109/IMF.2015.17.
- [17] M. binti Mohamad Noor and W. H. Hassan, "Current research on Internet of Things (IoT) security: A survey," *Comput. Netw.*, vol. 148, pp. 283–294, Jan. 2019, doi: 10.1016/j.comnet.2018.11.025.
- [18] F. K. Santoso and N. C. H. Vun, "Securing IoT for smart home system," in *Proceedings of the International Symposium on Consumer Electronics, ISCE*, Madrid, Aug. 2015, pp. 1–2. doi: 10.1109/ISCE.2015.7177843.
- [19] I. Ali, S. Sabir, and Z. Ullah, "Internet of Things Security, Device Authentication and Access Control: A Review," *ArXiv190107309 Cs*, Jan. 2019, Accessed: Jun. 22, 2019. [Online]. Available: <http://arxiv.org/abs/1901.07309>
- [20] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Security Implications of Permission Models in Smart-Home Application Frameworks," *IEEE Secur. Priv.*, vol. 15, no. 2, pp. 24–30, Mar. 2017, doi: 10.1109/MSP.2017.43.
- [21] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, May 2016, pp. 636–654. doi: 10.1109/SP.2016.44.
- [22] Y. Tian *et al.*, "SmartAuth: User-Centered Authorization for the Internet of Things," in *USENIX Security Symposium*, Vancouver, BC, Canada, 2017, pp. 361–378. Accessed: September. 10, 2023. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tian>
- [23] I. Astaburuaga, A. Lombardi, B. La Torre, C. Hughes, and S. Sengupta, "Vulnerability Analysis of AR.Drone 2.0, an Embedded Linux System," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2019, pp. 0666–0672. doi: 10.1109/CCWC.2019.8666464.
- [24] I. Yaqoob *et al.*, "Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges," *IEEE Wirel. Commun.*, vol. 24, no. 3, pp. 10–16, 2017, doi: 10.1109/MWC.2017.1600421.
- [25] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart Nest Thermostat: A Smart Spy in Your Home," 2014, pp. 1–8. Accessed: September. 10, 2023. [Online]. Available: <https://www.blackhat.com/docs/us-14/materials/us-14-Jin-Smart-Nest-Thermostat-A-Smart-Spy-In-Your-Home-WP.pdf>
- [26] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and Logging in the Internet of Things," in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018, pp. 1–15. doi: 10.14722/ndss.2018.23282.
- [27] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions," *J. Ambient Intell. Humaniz. Comput.*, May 2017, doi: 10.1007/s12652-017-0494-4.
- [28] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures

- via Software-Defined Security,” *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10250–10276, Oct. 2020, doi: 10.1109/JIOT.2020.2997651.
- [29] S. Zaman *et al.*, “Security Threats and Artificial Intelligence Based Countermeasures for Internet of Things Networks: A Comprehensive Survey,” *IEEE Access*, vol. 9, pp. 94668–94690, 2021, doi: 10.1109/ACCESS.2021.3089681.
- [30] M. Bradbury, A. Jhumka, T. Watson, D. Flores, J. Burton, and M. Butler, “Threat-modeling-guided Trust-based Task Offloading for Resource-constrained Internet of Things,” *ACM Trans. Sens. Netw.*, vol. 18, no. 2, p. 29:1-29:41, Feb. 2022, doi: 10.1145/3510424.
- [31] Y. Lu, D. Wang, M. S. Obaidat, and P. Vijayakumar, “Edge-assisted Intelligent Device Authentication in Cyber-Physical Systems,” *IEEE Internet Things J.*, pp. 1–1, 2022, doi: 10.1109/JIOT.2022.3151828.
- [32] M. A. Rahman, A. T. Asyhari, L. S. Leong, G. B. Satrya, M. Hai Tao, and M. F. Zolkipli, “Scalable machine learning-based intrusion detection system for IoT-enabled smart cities,” *Sustain. Cities Soc.*, vol. 61, p. 102324, Oct. 2020, doi: 10.1016/j.scs.2020.102324.
- [33] Y. Fu, Y. Du, Z. Cao, Q. Li, and W. Xiang, “A Deep Learning Model for Network Intrusion Detection with Imbalanced Data,” *Electronics*, vol. 11, no. 6, Art. no. 6, Jan. 2022, doi: 10.3390/electronics11060898.
- [34] J. Yu, X. Ye, and H. Li, “A high precision intrusion detection system for network security communication based on multi-scale convolutional neural network,” *Future Gener. Comput. Syst.*, vol. 129, pp. 399–406, Apr. 2022, doi: 10.1016/j.future.2021.10.018.
- [35] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, “E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT,” in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2022, pp. 1–9. doi: 10.1109/NOMS54207.2022.9789878.
- [36] I. O. Lopes, D. Zou, I. H. Abdulqadder, F. A. Ruambo, B. Yuan, and H. Jin, “Effective network intrusion detection via representation learning: A Denoising AutoEncoder approach,” *Comput. Commun.*, vol. 194, pp. 55–65, Oct. 2022, doi: 10.1016/j.comcom.2022.07.027.
- [37] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, “Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices,” *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6882–6897, Aug. 2020, doi: 10.1109/JIOT.2020.2970501.
- [38] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, “Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets.” arXiv, Jan. 06, 2022. doi: 10.48550/arXiv.2201.02177.
- [39] C. F. G. D. Santos and J. P. Papa, “Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks,” *ACM Comput. Surv.*, vol. 54, no. 10s, p. 213:1-213:25, Sep. 2022, doi: 10.1145/3510413.
- [40] Z. A. E. Houda, B. Brik, and L. Khoukhi, ““Why Should I Trust Your IDS?”: An Explainable Deep Learning Framework for Intrusion Detection Systems in Internet of Things Networks,” *IEEE Open J. Commun. Soc.*, vol. 3, pp. 1164–1176, 2022, doi: 10.1109/OJCOMS.2022.3188750.
- [41] A. P. Mathur, *Foundations of Software Testing: Fundamental Algorithms and Techniques*, 1st edition. Delhi: Addison-Wesley Professional, 2008.

- [42] M. Merayo, M. Núñez, and I. Rodríguez, “Extending EFSMs to Specify and Test Timed Systems with Action Durations and Time-Outs,” *IEEE Trans. Comput.*, vol. 57, no. 6, pp. 835–844, Jun. 2008, doi: 10.1109/TC.2008.15.
- [43] D. Bresolin, K. El-Fakih, T. Villa, and N. Yevtushenko, “Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power,” *Electron. Proc. Theor. Comput. Sci.*, vol. 161, pp. 203–216, Aug. 2014, doi: 10.4204/EPTCS.161.18.
- [44] D. Bresolin, K. El-Fakih, T. Villa, and N. Yevtushenko, “Equivalence Checking and Intersection of Deterministic Timed Finite State Machines,” *ArXiv210304868 Cs*, Mar. 2021, Accessed: Apr. 01, 2022. [Online]. Available: <http://arxiv.org/abs/2103.04868>
- [45] D. Bresolin, K. El-Fakih, T. Villa, and N. Yevtushenko, “Equivalence checking and intersection of deterministic timed finite state machines,” *Form. Methods Syst. Des.*, Sep. 2022, doi: 10.1007/s10703-022-00396-6.
- [46] G. V. Bochmann and A. C. Petrenko, “Protocol testing: review of methods and relevance for software testing,” *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*. pp. 109–124, 1994. doi: 10.1145/186258.187153.
- [47] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, and N. Yevtushenko, “FSM-based conformance testing methods: A survey annotated with experimental evaluation,” *Information and Software Technology*, vol. 52, no. 12. pp. 1286–1297, 2010. doi: <https://doi.org/10.1016/j.infsof.2010.07.001>.
- [48] M. Gromov, K. El-Fakih, N. Shabdina, and N. Yevtushenko, “Distinguishing Non-deterministic Timed Finite State Machines.” Springer Berlin Heidelberg, pp. 137–151.
- [49] B. Song, H. Choi, and H. S. Lee, “Surveillance Tracking System Using Passive Infrared Motion Sensors in Wireless Sensor Network,” in *2008 International Conference on Information Networking*, Jan. 2008, pp. 1–5. doi: 10.1109/ICOIN.2008.4472790.
- [50] C. Tsai, Y. Bai, C. Chu, C. Chung, and M. Lin, “PIR-sensor-based lighting device with ultra-low standby power consumption,” *IEEE Trans. Consum. Electron.*, vol. 57, no. 3, pp. 1157–1164, Aug. 2011, doi: 10.1109/TCE.2011.6018869.
- [51] R. Want, “An introduction to RFID technology,” *IEEE Pervasive Comput.*, vol. 5, no. 1, pp. 25–33, Jan. 2006, doi: 10.1109/MPRV.2006.2.
- [52] S. Shapsough and I. A. Zualkernan, “IoT for Ubiquitous Learning Applications: Current Trends and Future Prospects,” in *Ubiquitous Computing and Computing Security of IoT*, N. Jeyanthi, A. Abraham, and H. Mcheick, Eds. Cham: Springer International Publishing, 2019, pp. 53–68. doi: 10.1007/978-3-030-01566-4_3.
- [53] Y. Fu, J. Cao, X. Cao, Z. Yan, and O. Kone, “An Automata Based Intrusion Detection Method for Internet of Things,” *Mobile Information Systems*, vol. 2017. 2017. doi: 10.1155/2017/1750637.
- [54] B. Arrington, L. Barnett, R. Rufus, and A. Esterline, “Behavioral Modeling Intrusion Detection System (BMIDS) Using Internet of Things (IoT) Behavior-Based Anomaly Detection via Immunity-Inspired Algorithms,” *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. pp. 1–6. doi: 10.1109/ICCCN.2016.7568495.
- [55] H. Sedjelmaci, S. M. Senouci, and M. Al-Bahri, “A lightweight anomaly detection technique for low-resource IoT devices: A game-theoretic methodology,” *2016*

- IEEE International Conference on Communications (ICC)*. pp. 1–6. doi: 10.1109/ICC.2016.7510811.
- [56] M. Ge, J. B. Hong, W. Guttman, and D. S. Kim, “A framework for automating security analysis of the internet of things,” *Journal of Network and Computer Applications*, vol. 83. pp. 12–27, 2017. doi: 10.1016/j.jnca.2017.01.033.
- [57] T. Loise, X. Devroey, G. Perrouin, M. Papadakis, and P. Heymans, “Towards Security-Aware Mutation Testing,” in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2017, pp. 97–102. doi: 10.1109/ICSTW.2017.24.
- [58] A. Tvardovskii, K. El-Fakih, and N. Yevtushenko, “Deriving Tests with Guaranteed Fault Coverage for Finite State Machines with Timeouts,” in *Testing Software and Systems*, Cham, 2018, pp. 149–154. doi: 10.1007/978-3-319-99927-2_13.
- [59] C. N. Hadjicostis, "Probabilistic detection of FSM single state-transition faults based on state occupancy measurements," *IEEE Transactions on Automatic Control*, vol. 50, no. 12, pp. 2078-2083, 2005, doi: 10.1109/TAC.2005.860270
- [60] A. Saeed, A. Ahmadiania, A. Javed, and H. Larijani, "Intelligent Intrusion Detection in Low-Power IoTs," *ACM Transactions on Internet Technology*, vol. 16, no. 4, pp. 1-25, 2016, doi: 10.1145/2990499.
- [61] A. Subasi et al., "Intrusion Detection in Smart Grid Using Data Mining Techniques," in *2018 21st Saudi Computer Society National Computer Conference (NCC): IEEE*, 2018, pp. 1-6
- [62] V. V. Kumari and P. R. K. Varma, "A semi-supervised intrusion detection system using active learning SVM and fuzzy c-means clustering," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 10-11 Feb. 2017 2017, pp. 481-485, doi: 10.1109/I-SMAC.2017.8058397
- [63] A. Sforzin, F. G. Mármol, M. Conti, and J. Bohli, "RPiDS: Raspberry Pi IDS — A Fruitful Intrusion Detection System for IoT," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCCom/IoP/SmartWorld)*, 18-21 July 2016 2016, pp. 440-448, doi: 10.1109/UICATC-ScalCom-CBDCCom-IoP-SmartWorld.2016.0080
- [64] S. Andy, B. Rahardjo, and B. Hanindhito, "Attack scenarios and security analysis of MQTT communication protocol in IoT system," in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 19-21 Sept. 2017 2017, pp. 1-6, doi: 10.1109/EECSI.2017.8239179
- [65] S. Prabavathy, K. Sundarakantham, and S. M. Shalinie, "Design of cognitive fog computing for intrusion detection in Internet of Things," *Journal of Communications and Networks*, vol. 20, no. 3, pp. 291-298, 2018, doi: 10.1109/JCN.2018.000041
- [66] A. Amouri, V. T. Alaparthy, and S. D. Morgera, "Cross layer-based intrusion detection based on network behavior for IoT," in *2018 IEEE 19th Wireless and Microwave Technology Conference (WAMICON)*, 9-10 April 2018 2018, pp. 1-4, doi: 10.1109/WAMICON.2018.8363921
- [67] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino, "Kalis — A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 5-8 June 2017 2017, pp. 656-666, doi: 10.1109/ICDCS.2017.104.

- [68] L. Santos, C. Rabadao, R. Goncalves, and C. t. I. C. o. I. S. a. T. C. th Iberian Conference on Information Systems and Technologies, "Intrusion detection systems in Internet of Things: A literature review," Iberian Conference on Information Systems and Technologies, CISTI, vol. 2018-June, pp. 1-7, 2018, doi: 10.23919/CISTI.2018.8399291.
- [69] Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, 2015, doi: 10.1109/COMST.2015.2444095.
- [70] S. Shapsough, F. Aloul, and I. A. Zualkernan, "Securing Low-Resource Edge Devices for IoT Systems," in 2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI), 6-7 Sept. 2018 2018, pp. 1-4, doi: 10.1109/ISSI.2018.8538135.
- [71] K. El-Fakih, N. Yevtushenko, and A. Simao, "A practical approach for testing timed deterministic finite state machines with single clock," Science of Computer Programming, vol. 80, pp. 343-355, 2014, doi: <https://doi.org/10.1016/j.scico.2013.09.008>.
- [72] A. Amouri, V. T. Alaparthi, and S. D. Morgera, "Cross layer-based intrusion detection based on network behavior for IoT," in 2018 IEEE 19th Wireless and Microwave Technology Conference (WAMICON), 9-10 April 2018 2018, pp. 1-4, doi: 10.1109/WAMICON.2018.8363921.
- [73] S. Zhao, W. Li, T. Zia, and A. Y. Zomaya, "A Dimension Reduction Model and Classifier for Anomaly-Based Intrusion Detection in Internet of Things," in 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), 6-10 Nov. 2017 2017, pp. 836-843, doi: 10.1109/DASCPICom-DataCom-CyberSciTec.2017.141.
- [74] M. D. Maio et al., "Closed-Loop Systems Engineering (CLOSE): Integrating Experimentable Digital Twins with the Model-Driven Engineering Process," in 2018 IEEE International Systems Engineering Symposium (ISSE), 1-3 Oct. 2018 2018, pp. 1-8, doi: 10.1109/SysEng.2018.8544392.
- [75] M. Schluse, L. Atorf, J. Rossmann, and Q. C. C. A. A. Annual Ieee International Systems Conference Annual Ieee International Systems Conference Montreal, "Experimentable digital twins for model-based systems 44 engineering and simulation-based development," in 2017 Annual IEEE International Systems Conference (SysCon): IEEE, 2017, pp. 1-8.
- [76] K. M. Alam and A. E. Saddik, "C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems," IEEE Access, vol. 5, pp. 2050-2062, 2017, doi: 10.1109/ACCESS.2017.2657006.
- [77] L. Atorf and J. Roßmann, "Interactive Analysis and Visualization of Digital Twins in High-Dimensional State Spaces," in 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), 18-21 Nov. 2018 2018, pp. 241-246, doi: 10.1109/ICARCV.2018.8581126.
- [78] A. Tvardovskii, K. El-Fakih, and N. Yevtushenko, 'Testing and incremental conformance testing of timed state machines', Science of Computer Programming, vol. 233, p. 103053, 2024.

Vita

Alifiya Taiyabali Bhanpurawala was born in 1991, in Mumbai, India. She received her primary and secondary education in Mumbai, India. She received her BE degree in Computer Engineering from Sardar Patel Institute of Technology, India in 2013. From 2013 to 2015, she worked as a Technology Analyst at JP Morgan.

In Jan 2019, she joined the Computer Engineering master's program in the American University of Sharjah as a graduate teaching assistant. Her research interests are in Formal methods, Internet of Things and Big Data.