IoTC^2: A Formal Method Approach for Detecting Conflicts in Large Scale IoT Systems

Abdullah Al Farooq*, Ehab Al-Shaer*, Thomas Moyer*, Krishna Kant†
*The University of North Carolina at Charlotte
{afarooq,ealshaer,tom.moyer}@uncc.edu
†Temple University
{kkant}@temple.edu

***Abstract*—Internet of Things (IoT) has become a common paradigm for different domains such as health care, transportation infrastructure, smart homes, smart shopping, and e-commerce. With its interoperable functionality, it is now possible to connect all domains of IoT together to provide comprehensive services to the users. Because numerous IoT devices can connect and communicate at the same time, there can be events that trigger conflicting actions for an actuator or an environmental feature. This paper provides a formal method approach, IoT Confict Checker (IoTC^2), to ensure safety of controller and actuators' behavior with respect to conflicts. Any policy violation results in detection of the conflicts. We define the safety policies for controller, actions, and triggering events and implement them in Prolog to prove the logical completeness and soundness. In addition to that, we have implemented the detection policies in Matlab Simulink Environment with its built-in Model Verification blocks. We created a smart home environment in Simulink and showed how the conflicts affect actions and corresponding features. The scalability, efficiency, and accuracy of our method are tested in this simulated environment.**

***Index Terms*—Internet of Things(IoT), Formal Method, Conflicts, Policies, Simulation, Safety, Security**

I. Introduction

Recently, the security of Internet of Things (IoT) has become a hot topic in the technology community. The deployment and application of IoT devices covers a variety of fields, ranging from the smart homes to transportation management system. It is projected that there will be as many as 50 billion connected devices by 2020 [1]. IoT presents a unique advantage in that is connects devices within and across domains, e.g. smart homes, traffic route guidance systems, shopping systems, ride sharing, and parking management systems. IoT applications and devices share required data and provide unified services to the users.

However, the distributed nature of IoT leaves devices and communication channels exposed to attackers and many of these devices and protocols are resource-constrained. This, combined with the fact that many of these devices receive infrequent updates leaves them highly susceptible to attack. An attacker can trigger an event that leads to conflicting actions for the same object or feature of the environment. As for example, an attacker can create multiple events that trigger a thermostat to increase and decrease temperature of a room at the same time. Sending two different commands in the thermostat at the same time continuously can damage it, by artificially shortening the device's lifespan. In this way, the attacker not only damages an asset, but also may drive the occupants of the room to leave due to fluctuations in the comfort level of the room. Moreover, misconfiguration is possible as there are numerous rules or policies for taking actions by the controllers after events have occurred.

Attackers can leverage these conflicts and vulnerabilities to gain physical access to a smart building. For example, an attacker may compromise a carbon monoxide sensor and falsely trigger the alarm indicating the presence of carbon monoxide, which in turn sends a command to the smart windows to open allowing fresh air into the building. Occupants will leave the building due to the alarm. A thief can target the opened windows to enter the building. Additionally, an attacker can create series of attacks (cascading attack) [2]. Even with IoT technology in the early stages of development and deployment, the guarantees of maintaining safe and secure operation of an environment through IoT devices can attract more users. Even legacy, or dumb devices can be attached to the system and be operated through a controller.

Due to the limited computational and memory capacities of IoT devices, it is not always possible to secure and monitor each and every device and communication channel. A controller or a group of controllers provide the computational and storage capacity to make decisions based on events coming from the edge devices (i.e. sensors) and issue commands to the appropriate actuators. The automated decisions made by a controller may try to command a device which is already performing a different action. Moreover, devices connect to the IoT controller intermittently. Events can occur any time or an attacker can force specific events to happen in order to take advantage of misconfigurations (i.e. rule conflicts) in the IoT system. Furthermore, the operational policies for an IoT system can change over time based on the requirements of the building and its occupants. Hence, it is important to check whether a recently triggered event causes a conflict based on the ruleset that is stored in the controllers. ProvThings [3] proposes a provenance-based approach for explaining anomalies in IoT systems. ProvThings and IoTC^2 can work in concert to provide a solid security foundation for IoT systems.

978-3-903176-15-7 © 2019 IFIP

In this paper, we propose IoTC^2, a formal methods approach to ensure safety properties for the controllers and actuators in an IoT system. The main contributions of the paper are:

- a formal approach to defining the safety properties of an IoT system
- a technique for detecting conflicts within the rulesets of the IoT system that violate the safety properties of the system
- an implementation of IoTC^2 that can be used in real-time to ensure the safety of the IoT system

An IoT system provider, or the app developer, can leverage our framework while writing rules for an IoT system. Once the conflict creating actuators/controllers are distinguished by our framework, the app developer can modify operational rules to avoid the conflicts as much as possible. Furthermore, IoT vendors can instrument the controller or app with our policies to monitor how many safety property violations have occurred in an IoT environment. This helps in classifying and differentiating anomalies in an IoT system. It should also be noted that actuation commands are issued from the controllers. Hence, multiple controllers that try to command the same actuator are in scope for this work. IoTC^2 contains all the rules for the IoT system so that when events trigger a rule or a set of rules, IoTC^2 makes sure the safety properties are maintained. When a violation of the safety properties occurs, it is due to conflicts in the rules defined for the system. The framework is also capable of detecting conflicts that are caused by misconfiguration of operational rules. The conflicts are detected even before the conflicting actions take place. IoTC^2 can identify the specific type of violation that has occurred. We have implemented an IoT environment simulation in Matlab's Simulink environment to understand the impact of conflicts in an IoT system.

II. IoTC^2 Framework

The architecture of the IoTC^2 is shown in Figure 1. All of the rules used for operation in each controller of the IoT system are the input for IoTC^2. Whenever there is a change in any rule or an additional rule is added to a controller, it is also configured in IoTC^2. As mentioned earlier, because these rules are generated by a human, they are finite in number. Therefore, it is feasible to accommodate all rules in IoTC^2. The second type of input for our framework is the sensor measurement with timestamps. To start, our framework receives copies of the traffic from the sensors to the controllers. As soon as IoTC^2 receives the sensor measurement, by using the rules from the controller, IoTC^2 determines what actions the controller will emit for the actuators. Next, IoTC^2 determines the list of actuators, affected features, and issuing controllers. Using these lists, IoTC^2 determines whether or not these activities violate the safety properties (or create conflicts) within the IoT system. IoTC^2 has the capability to output the number of conflicts and their type in the IoT system.

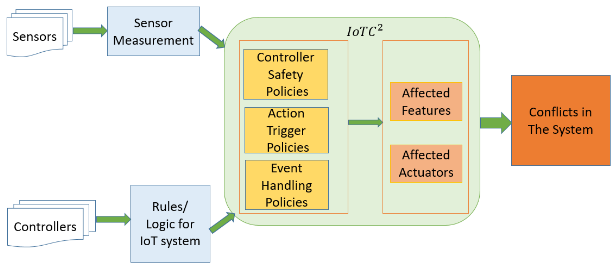

Fig. 1: IoTC^2 Framework for Conflict Detection

III. Formal Method for Detecting Conflicts

In this section, we present our formal method for detecting safety property violations in IoT systems. In the following description, we rely on the notation defined in Table I. We start by defining some general properties of the IoT system model, events, triggers, and actions.

Notation	Explanation
e_i^t	Event i generated at time t
$a_{m,n}^{l,f}(t)$	Actuator m taking action n on feature f at location l
c^p	Controller p
x'	Object x' can be the same or different from object x
$\bar{x}$	Object $\bar{x}$ must be different from x

TABLE I: Notation used

Definition 1: If there are features f_x and f_y such that changes in f_x affect f_y, then these features are dependent. It can be the case that feature f_x and feature f_y are not directly dependent, however feature f_x affects f_z and f_z affects f_y. In this case, f_y is indirectly dependent on f_x. Regardless, they are noted as:

$$f_x \overset{d}{=} f_y \tag{1}$$

Definition 2: If two events are the same or similar by their characteristics and functionality and they occur within a bounded time frame, these events are called overlapping events. They are noted as:

$$e_1 \overset{o}{=} e_2 \tag{2}$$

Whenever two events are not overlapping, they are considered disjoint events.

A. Controller Safety Policies

The controller is a crucial component of an IoT system that receives measurements from sensors and based on those measurements, it generates actuation commands for the appropriate actuators. We define the controller safety policies as follows:

- There are no two rules where two or more controllers can trigger the same actuator at the same time.

$$C_1 : \neg((e_i^t \Rightarrow a_{m \in c_p, n}^{l,f}) \land (e_{i'}^t \Rightarrow a_{m \in c_{\bar{p}}, n'}^{l,f'})) \tag{3}$$

An IoT system may have a number of rules where the same actuator m is controlled by more than one controller c_p and

$c_{\bar{p}}$. If the same actuator is accessed at the same time t, a conflict occurs. The actions (denoted by subscript n and n') on the actuators, and the affected features (superscript f and f') can be same or different, which does not change this policy. The following are examples of conflict scenarios that can be captured by this rule.

- A smoke detector and a water-leak detector (controlled by different controllers) can each trigger an alarm. But if the same alarm sounds at the same time, it will be hard to distinguish which event triggered the alarm. The policy formalized here restricts multiple controllers from triggering the same action at the same time.

- There are no two rules where two controllers can trigger actions that affect the same, or dependent, features at the same time.

$$C_2\colon \neg((e_i^t \Rightarrow a^{l,f}_{m\in c_p,n}) \wedge (e^t_{i'} \Rightarrow a^{l,f'}_{m'\in c_{\bar{p}},n'}) \wedge (f = f' \vee f \overset{d}{=} f')) \tag{4}$$

There can be more than one rule that can trigger different actions (the subscript m and m' denote different actuators). However, these different actuators can impact the same, or dependent features. The same features are identified with '$=$' . On the other hand, the notion $\overset{d}{=}$ denotes the dependency among two features f and f'. An example of such violation a is:

- A window opener and a thermostat can be two different actuators controlled by different controllers, but can affect the same feature (temperature) of the room.

B. Multiple Action Trigger Policies

When an actuator is issued commands to perform multiple actions at the same time, conflicts can occur. In order to prevent conflicts, we have the following safety property:

- There are no two rules where two or more overlapping events (from any sensor) can trigger multiple actions (different action n' / opposite action $\bar{n}$ / dependent action $\hat{n}$) on the same actuator.

$$C_3\colon \neg((e_i^t \Rightarrow a^{l,f}_{m,n}) \wedge (e^t_{i'} \Rightarrow (a^{l,f'}_{m,\hat{n}} \wedge a^{l,f'}_{m,n^*} \wedge a^{l,f'}_{m,\bar{\bar{n}}})) \wedge (i \overset{o}{=} i')) \tag{5}$$

An action n on actuator m is the reference action and any action (n', $\bar{n}$, or $\hat{n}$) other than n is considered as the conflicting action on the same actuator m. An example of this safety policy violation is given below:

- Both room one and room two have temperature sensors but no thermostat. The corridor that joins both room has a thermostat, but no sensor. Temperature decreases in room one and temperature increases in room two can trigger the same thermostat. Hence, the thermostat can be triggered to increase the temperature and decrease the temperature at the same time.

- There are no two rules where overlapping events can trigger actions that affect the same or dependent features.

$$C_4\colon \neg((e_i^t \Rightarrow a^{l,f}_{m,n}) \wedge (e^t_{i'} \Rightarrow a^{l,f'}_{m,\bar{n}}) \wedge (i \overset{o}{=} i') \wedge ((f = f') \vee (f \overset{d}{=} f'))) \tag{6}$$

We differentiate two features by f and f'. There can be two rules that get activated at the same time by two overlapping events, resulting in two different actions, n and $\bar{n}$. These two actions affect features f and f' which are either the same or dependent. The following are examples of conflicts when this safety policy is violated:

- Luminance level can be impacted by overlapping events. Window blind and room lights are two different actuators that impact luminance.

- No two or more completely disjoint events can trigger multiple action on the same actuators

$$C_5\colon \neg((e_i^t \Rightarrow a^{l,f}_{m,n}) \wedge (e^t_{i'} \Rightarrow (a^{l,f'}_{m,\hat{n}} \vee a^{l,f'}_{m,n^*} \vee a^{l,f'}_{m,\bar{\bar{n}}})) \wedge \neg(i \overset{o}{=} i'))) \tag{7}$$

As disjoint events are not easy to relate, it is possible that these are overlooked when devising the IoT operational rules. Examples for such conflicts are:

- Management can impose a rule stating that when it is after 6 pm, the temperature need not be controlled.This means that the temperature of a smart building will follow the basic thermal model of the building. On the other hand, movement in a room will trigger the thermostat to increase/decrease the temperature for better occupant comfort.

- No two completely disjoint events can trigger multiple actions that affect the same, or dependent features.

$$C_6\colon \neg((e_i^t \Rightarrow a^{l,f}_{m,n}) \wedge (e^t_{i'} \Rightarrow a^{l,f'}_{m,\bar{n}}) \wedge \neg(i \overset{o}{=} i') \wedge ((f = f') \vee (f \overset{d}{=} f'))) \tag{8}$$

Two rules can be triggered by completely disjoint events at the same time. The actuator and its location are kept unchanged by notation m and l, respectively. The actions are differentiated by n and n'. At the same location and with the same actuator, two features f and f' got affected. When these two features are dependent, $f \overset{d}{=} f'$ will return true.

- A window opening or closing and a thermostat turning on or off are two completely disjoint events that impact the temperature of the room

C. Multiple Event Handling Policies

A controller actuates an IoT device when an event occurs. There is relatively little control over how and when an event is generated. However, the way multiple events are handled, can be controlled. Therefore, we focus on formalizing event handling policies. The formalization is as follows:

- No single sensor with single objective can create more than one event within a specific time limit.

$$C_7\colon \neg((e_i^t \Rightarrow a_{m,n}^{l,f}) \wedge (e_j^{\bar{t}} \Rightarrow a_{m,n}^{l,f})) \tag{9}$$

Here, two events i and i' are prohibited from same sensor j within a time limit t'. A sensor might send the same measurement to the controller more than once due to any physical or communication issue. This should be handled in a proper way so that same actions are not taken by the same actuator.

A sensor can send same temperature measurement (e.g. 60F) twice to the controller within a 30 second interval. The controller would instruct the thermostat to increase the temperature by 10F each time it receives the input from the sensor. Therefore, the temperature of the room is increased to 80F.

D. Completeness of IoT Safety Properties

Definition 3: If an IoT system, comprised of sensors $s_{1...m}$, controllers $cntrl_{1...n}$, and actuators $a_{1...o}$, violates any safety properties $c_{1...p}$, a conflict $c^* \in Conflict$ has occurred.

Completeness means that you can prove anything that's true. The safety policies were implemented using Prolog. If there exists a conflict $c^* \in Conflict$ in the IoT system operations, IoTC^2 finds it using *Selective Linear Definite-clause with Negation as Failure* SLDNF [4]. However, the Dept First Search (DFS) strategy of Prolog makes it logically incomplete. We followed the way proposed by [5] where the built-in dept-first search of Prolog was overruled by iterative deepening. In doing so, we have used tail-recursion. In each recursive call, the number of actions triggered, or the controllers associated with it, or the events that triggered the actions are stored in three separate lists. In this way, the search space is being completed in lists. Then this list is traversed to find the conflicts in the system. Whenever, a resolution refutation is found, our Prolog implementation finds it and adds 0 (zero) to the accumulator rather than stopping the search process on that node.

E. Soundness of IoT Safety Properties

Definition 4: The safety properties of IoTC^2 is sound, if for all sensors $s_{1...m}$, controllers $cntrl_{1...n}$, and actuators $a_{1...o}$, all possible operations in the system are a subset of the authorized operations allowed by IoTC^2.

With the definition of soundness from 4, we can conclude that all safety properties, expressed in conjunctive normal form (CNF) make it logically sound as given in equation 10.

$$C_f\colon C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7 \tag{10}$$

IV. Evaluation

For our evaluation, we created an IoT environment using Matlab's Simulink. The basic purpose was to observe the interaction among the sensors, actuators, and controllers. Simulink is a robust and widely accepted simulation tool for power electronics, nuclear energy, manufacturing production, aerospace, transportation, supply chain management, etc. IoTC^2 was implemented in the Simulink testbed to monitor the testbed and detect conflicts. We designed a smart house with three rooms with corridors attaching each of the rooms. The thermal model of the house was adapted from [6]. The rooms have facilities for smoke detection, carbon monoxide detection, smart lights, smart window shutters and blinds, smart doors, etc. Operational rules for automating these IoT devices are implemented using appropriate Simulink blocks. We run each simulation several times to observe whether IoTC^2 can detect conflicts in the smart house successfully. We have used the built-in model verification blocks (e.g., Assertion, Check Dynamic Gap, Check Dynamic Range, etc) of Simulink to help our detection. IoTC^2 outputs the total count of each different type of detected conflict in a given simulation step regardless of how many events have occurred there. Not only are the detected conflicts shown, but also their effects on features or actuators. The Simulink model for the testbed is available in [7].

First, we evaluate the effects of conflicts on actuators or the environment feature. Later, we discuss how well IoTC^2 can detect the conflict in a simulated environment. Our first experiment was conducted to observe the effects of conflicts as mentioned in the example of equation 6. The simulation was run for 500 units of time (Figure 2(a)). The second experiment observes the example mentioned for equation 8 and given in Figure 2(b). The blue line indicates the expected temperature reading while the red line indicates how much the temperature reading deviates due to the conflicts. The third experiment captures the impact on temperature as stated in the example under equation 5. The temperature, as shown by the red line in Figure 2(c), is the temperature of the corridor, calculated by Simulink. The blue line is the temperature reading during different period of time from room two.

At this point, we move our focus on counting the number of conflicts in a given time by varying different parameters. First, we consider the case where the same alarm (actuator) gets triggered when smoke is detected or a water leak is detected (operated by two controllers). We vary the probability of smoke detection and water leak detection by a small percentage in each time unit. The simulation was run for 2000 time units. It is shown in Figure 3(a) that the same alarm gets triggered by different events at the same time with the increase of simulation time. In the next experiment, we count the number of times the window shutters of the smart home are open and the thermostat turns on at the same time (Figure 3(b)). Next, we run a simulation as if the humidity is not affected by temperature. Then, we re-run the simulation with humidity being dependent on temperature. As can be seen in Figure 3(c) when there is greater temperature variation outside the smart home, the humidifier inside the house gets turned on and off more often. In our next experiment, as shown in Figure 3(d), IoTC^2 counts the number of times the luminance of a room exceeded a comfortable range due to conflicting actions between the smart lights and the smart window blinds. The next experiment measures the number of conflicts between management rules and operational rules as mentioned in the example under equation 7. As expected

(Figure 3(e)), more actuations on the thermostat are needed when conflicts occur. In our last experiment, IoTC^2 counts the additional count of humidifier actuations due to conflicts between operational rules and management rules. The results are shown in Figure 3(f).

V. Related Work

Most of the research efforts in IoT has been on management, efficiency, interoperability, and deployment of these systems in the real world. Recently, confidentiality, access control, privacy, and trust issues of IoT technology have been discussed in [8]–[10]. In IoTSAT [11], a formal framework was proposed for security analysis based on device configurations, network topologies, user policies, and IoT-specific attack surface. Recently, the work by [12] proposes a verification framework with satisfiable module theory (SMT) for a smart environment with respect to event-condition-action (ECA). However, this research did not propose either safety properties or address conflicts for the smart environment. The work by [13] made an effort classify conflicts into a set of finite categories. They specified the relation among all building management rules and classified all type of rule conflicts into five categories. Our approach is quite different. Rather than classifying conflicts, we specified safety properties for the components of IoT and the violation of those properties leads to conflicts.

Some preliminary work in the areas of formal modeling and verification for IoT driven domains has been done [14], [15]. The closest to this work in terms of detecting conflicts are Depsys [16], and HomeOS [17]. Depsys specified and detected conflicts after collecting the functionalities of 35 smart apps used in smart home. It detects the conflicts after they have occurred and in order to address the conflicts priorities have been set to the apps so that no two apps can access the same actuator. HomeOS is a large-scale application running on a centralized server that enables devices to talk to one another. That is, it acts as an event handler which aids in resolving device conflicts. Our approach, on the other hand, detects conflicts as soon as an event is generated which may immediately cause an action or a set of actions that result in conflicts. We have left the automated resolution of conflicts as future work.

VI. Conclusion

The conflicts that are possible in IoT system are often overlooked both in the design phase and during operations. IoT is an automated system and hence the accumulated effects of conflicts on an environment feature or actuator can have more effects initially anticipated. The safety and security of IoT systems is largely dependent of its conflict-less behavior. Hence, the safety properties we formalized in IoTC^2 consider conflicts as the preeminent threat to the safety and security of IoT system. Furthermore, our model has shown how conflicts can lead to additional actuations which eventually result in more energy consumption. In addition to the contributions mentioned above, we believe our proposed framework will have significant impacts when employed in the *policy monitor*

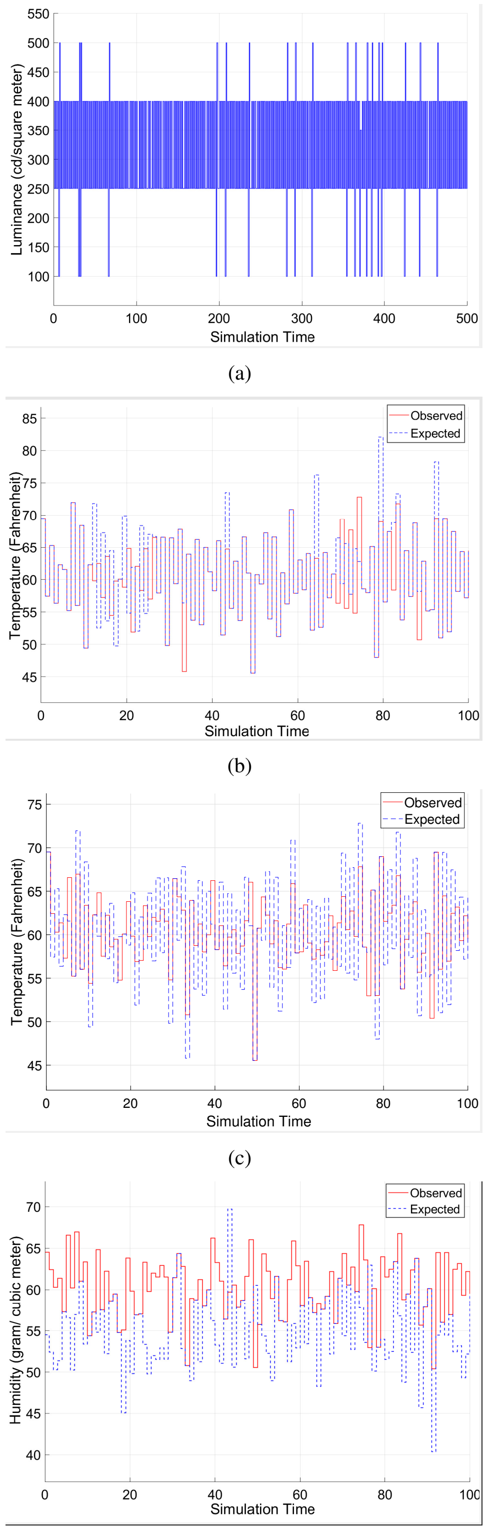

Fig. 2: (a)Luminance range of a room exceeds comfortable range, (b)Effect on Temperature when thermostat and window shutter works at the same time (c) Effect on temperature of the corridor when it is connected by two rooms (d) Effect on Humidity when thermostat and window shutter combinedly changes the temperature and humidity.

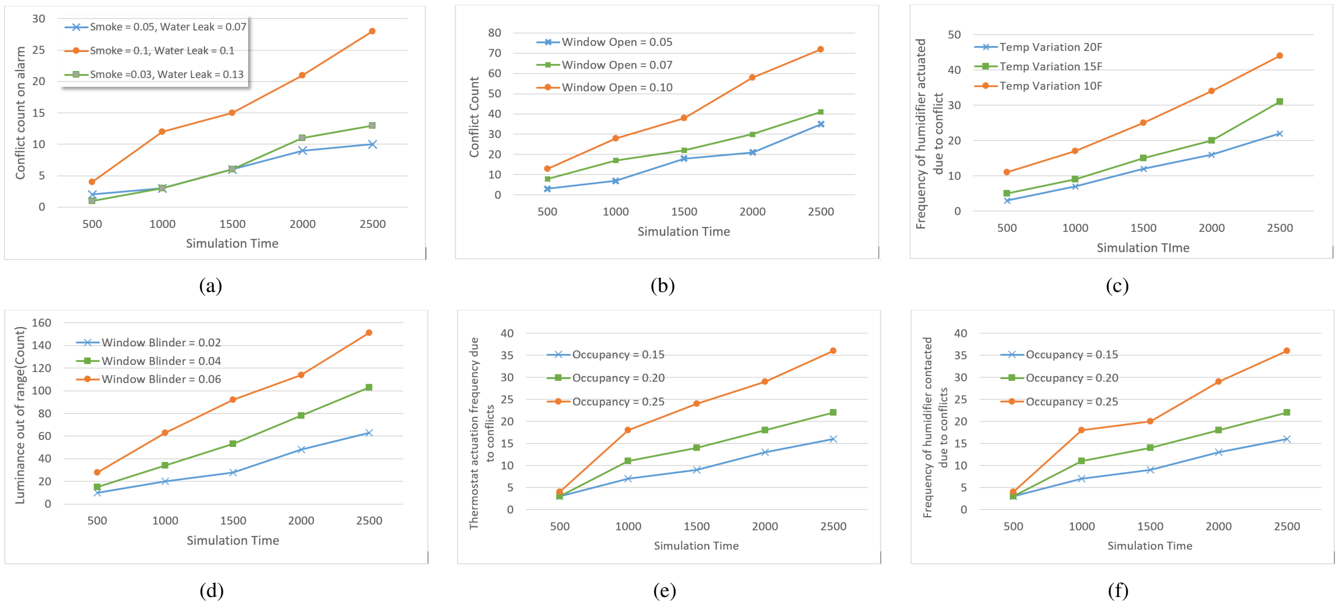

Fig. 3: (a) Conflict count when the same alarm is triggered by multiple events, (b) Frequency of thermostat being actuated more than usual due to the window being opened (c) Additional actuation count on the humidifier due to temperature difference in two adjacent rooms (d) Total count of the luminance range exceeding the comfortable range due to conflicts (e) Additional count of the thermostat being actuated due to conflicts between management rules and operational rules, (f) Additional frequency of humidifier being actuated due to conflicts between management rules and random occupancy.

block of *ProvThings* [3]. However, the enforcement of the proposed safety policies for a system is an open challenge. The implementation of an inlined reference monitor (IRM) [18] for enforcing the safety policies of our framework is another interesting research direction. IoT systems are emerging more and more in our daily life, and mechanisms are needed to ensure that these systems are safe, secure, and energy efficient if IoT systems are to be widely deployed and accepted.

REFERENCES

[1] E. Dave *et al.*, "How the next evolution of the internet is changing everything," *The Internet of Things*, 2011.

[2] S. COBB, "10 things to know about the october 21 iot ddos attacks," 2016. [Online]. Available: http://www.welivesecurity.com/2016/10/24/10-things-know-october-21-iot-ddos-attacks/

[3] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *ISOC NDSS*, 2018.

[4] M. Triska, "Theorem proving with prolog," https://www.metalevel.at/prolog/theoremproving, accessed: 2018-07-27.

[5] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "Swi-prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[6] M. Simulink, "Thermal model of a house," https://www.mathworks.com/help/simulink/examples/thermal-model-of-a-house.html, accessed: 2018-07-27.

[7] A. A. Farooq, "Iot testbed for iotc2," Dec. 2018. [Online]. Available: https://github.com/afarooq614/IoTC2

[8] Y. Fathy, P. Barnaghi, and R. Tafazolli, "Distributed spatial indexing for the internet of things data management," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE, 2017, pp. 1246–1251.

[9] F. Alsubaei, A. Abuhussein, and S. Shiva, "Quantifying security and privacy in internet of things solutions," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–6.

[10] H. Kinkelin, V. Hauner, H. Niedermayer, and G. Carle, "Trustworthy configuration management for networked devices using distributed ledgers," *arXiv preprint arXiv:1804.04798*, 2018.

[11] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman, "Iotsat: A formal framework for security analysis of the internet of things (iot)," in *Communications and Network Security (CNS), 2016 IEEE Conference on*. IEEE, 2016, pp. 180–188.

[12] C. Vannucchi, M. Diamanti, G. Mazzante, D. Cacciagrano, R. Culmone, N. Gorogiannis, L. Mostarda, and F. Raimondi, "Symbolic verification of event–condition–action rules in intelligent environments," *Journal of Reliable Intelligent Environments*, vol. 3, no. 2, pp. 117–130, 2017.

[13] Y. Sun, X. Wang, H. Luo, and X. Li, "Conflict detection scheme based on formal rule model for smart building systems," *Human-Machine Systems, IEEE Transactions on*, vol. 45, no. 2, pp. 215–227, 2015.

[14] F. Corno and M. Sanaullah, "Design-time formal verification for smart environments: an exploratory perspective," *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, no. 4, pp. 581–599, 2014.

[15] A. Coronato and G. De Pietro, "Formal design of ambient intelligence applications," *Computer*, vol. 43, no. 12, pp. 60–68, 2010.

[16] S. Munir and J. A. Stankovic, "Depsys: Dependency aware integration of cyber-physical systems for smart homes," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*. IEEE, 2014, pp. 127–138.

[17] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 25–25.

[18] F. B. Schneider, G. Morrisett, and R. Harper, "A language-based approach to security," in *Informatics*. Springer, 2001, pp. 86–101.