Precise Correlation Extraction for IoT Fault Detection with Concurrent Activities

GYEONGMIN LEE, Samsung Advanced Institute of Technology, Republic of Korea
BONGJUN KIM, POSTECH, Republic of Korea
SEUNGBIN SONG, Yonsei University, Republic of Korea
CHANGSU KIM, POSTECH, Republic of Korea
JONG KIM, POSTECH, Republic of Korea
HANJUN KIM, Yonsei University, Republic of Korea

In the Internet of Things (IoT) environment, detecting a faulty device is crucial to guarantee the reliable execution of IoT services. To detect a faulty device, existing schemes trace a series of events among IoT devices within a certain time window, extract correlations among them, and find a faulty device that violates the correlations. However, if a few users share the same IoT environment, since their concurrent activities make non-correlated devices react together in the same time window, the existing schemes fail to detect a faulty device without differentiating the concurrent activities. To correctly detect a faulty device in the multiple concurrent activities, this work proposes a new precise correlation extraction scheme, called PCoExtractor. Instead of using a time window, PCoExtractor continuously traces the events, removes unrelated device statuses that inconsistently react for the same activity, and constructs fine-grained correlations. Moreover, to increase the detection precision, this work newly defines a fine-grained correlation representation that reflects not only sensor values and functionalities of actuators but also their transitions and program states such as contexts. Compared to existing schemes, PCoExtractor detects and identifies 40.06% more faults for 4 IoT services with concurrent activities of 12 users while reducing 80.3% of detection and identification times.

CCS Concepts: • **Hardware** → **Emerging languages and compilers**; • **Security and privacy** → *Intrusion/anomaly detection and malware mitigation*.

Additional Key Words and Phrases: Internet of Things, Anomaly Detection, Compiler

ACM Reference Format:
Gyeongmin Lee, Bongjun Kim, Seungbin Song, Changsu Kim, Jong Kim, and Hanjun Kim. 2021. Precise Correlation Extraction for IoT Fault Detection with Concurrent Activities. 1, 1 (September 2021), 22 pages. https://doi.org/

1 INTRODUCTION

The proliferation of the Internet of Things (IoT) enables fascinating services such as home automation and a smart laboratory service. The IoT services collect physical states of a user environment from sensors, compute environment contexts such as occupancy of home and emergency state, and control

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2021.
Authors' addresses: Gyeongmin Lee, Samsung Advanced Institute of Technology, Republic of Korea, gm05.lee@samsung.com; Bongjun Kim, POSTECH, Republic of Korea, bong90@postech.ac.kr; Seungbin Song, Yonsei University, Republic of Korea, seungbin@yonsei.ac.kr; Changsu Kim, POSTECH, Republic of Korea, kcs9301@postech.ac.kr; Jong Kim, POSTECH, Republic of Korea, jkim@postech.ac.kr; Hanjun Kim, Yonsei University, Republic of Korea, hanjun@yonsei.ac.kr.

XXXX-XXXX/2021/9-ART $15.00
https://doi.org/

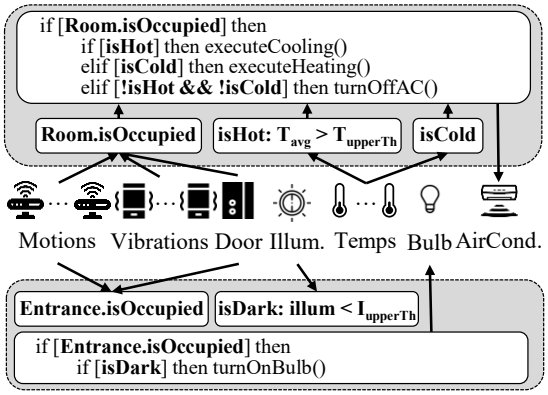


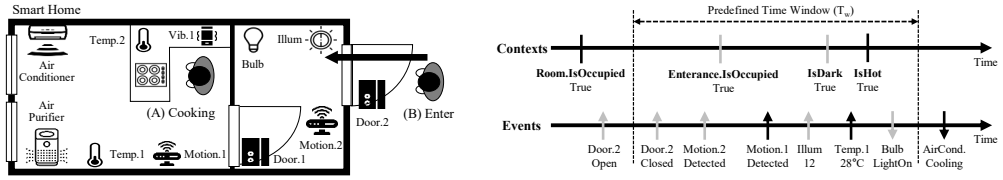
Fig. 1. IoT services: smart HVAC and smart light.

actuators such as smart bulb or air conditioner to provide predefined services for users. Figure 1 illustrates two IoT service examples such as a smart HVAC and a smart light. Multiple IoT sensors such as motion, vibration, door, and temperature sensors measure various physical states of the user environment. With the measured values, the smart HVAC service computes contexts such as `Room.isOccupied`, `isHot`, and `isCold`, and controls the air conditioner.

Despite the convenience of IoT services, a fault of an IoT device degrades the quality of the services and may cause crucial inconvenience. For example, if a door contact sensor in Figure 1 generates wrong data due to its fault, the smart HVAC service cannot correctly compute the `isOccupied` context. The wrong `isOccupied` value may unnecessarily activate the air conditioner, and thus causing excessive electricity bills. Therefore, detecting a faulty device in IoT environments is crucial to guarantee reliable execution of IoT services.

Faulty devices make problems not only disconnected from IoT services but sending wrong sensor values to the IoT services without disconnection. Therefore, the fault detection scheme should compare the values to find whether they satisfy trends of value increment over time or correlations between each other. Existing work [6, 19, 20, 25, 45–47] proposes correlation-based IoT fault detection schemes. FailureSense [25] and CLEAN [45–47] extract correlations between sensor-actuator or sensor-sensor and detect failures of binary sensors. SMART [19] and Idea [20] analyze correlations between sensors and user activities based on activity recognition techniques, and detect binary and numeric sensor failures. DICE [6] extracts a correlation as a set of device activation statuses, checks transitions between the correlations, and detects and identifies failures of generic devices.

However, applying the existing schemes to real-world IoT environments is challenging due to concurrently occurred multiple activities from multiple users. First, since they analyze correlations using a time window and assume that there exists at most one activity in a time window, they cannot differentiate multiple activities in the same time window. Second, since they trace only the occurrence of device reactions without inspecting the reasons why the devices react, they cannot differentiate activities that cause different reactions such as opening a door and leaving a door open. Third, although the same activity may cause different reactions depending on its contexts like room occupancy, the existing schemes do not reflect the contexts in the high level information into the correlation. Finally, since they include all the devices in the IoT environment in a correlation, a reaction of an non-correlated device causes a false-positive faulty device detection.



(a) Floor plan of the smart home for Figure 1

(b) Observed events at the smart home in Figure 2a

Fig. 2. Up-arrow means an event that is transferred from a device to the IoT platform, and down-arrow means the opposite.

To precisely detect and identify a faulty device in real-world environments, this work proposes a new compiler-assisted correlation extraction scheme, named PCoExtractor. This work designs a fine-grained correlation representation that reflects various reaction types of IoT devices and program status such as contexts. To easily reflect the high level information in service programs, the PCoExtractor compiler automatically extracts contexts and analyzes correlated devices in the programs. The PCoExtractor runtime continuously traces reactions of sensors, actuators, and contexts during a series of multiple concurrent activities without any time window, excludes non-correlated devices that inconsistently react for the same activities, and constructs fine-grained correlations. Finally, the runtime detects a faulty device if finding an abnormal event pattern that violates trained correlations.

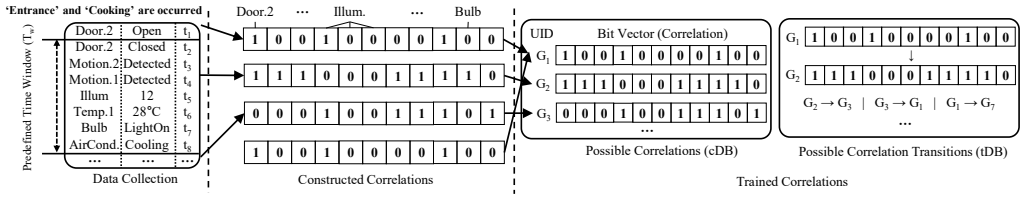
This work implements the PCoExtractor, compiler-assisted correlation extraction scheme on top of the LLVM C++ compiler infrastructure [22]. To evaluate the PCoExtractor, this work implements 4 IoT real-world services and collects data from the smart laboratory with 12 users for more than 40 days. Compared to existing schemes, PCoExtractor detects and identifies 40.06% more failures consisting of both fail-stop and non-fail-stop failures with concurrent activities. Moreover, the PCoExtractor reduces 80.3% of detection and identification time with only 181.7 μs delay on average.

The contributions of this paper are:

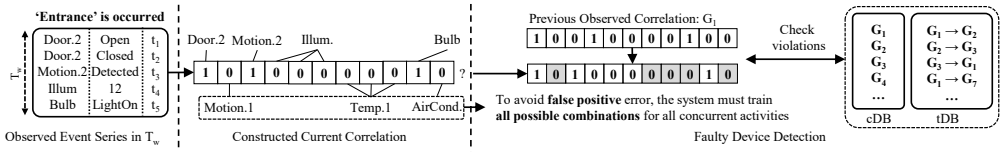
- the fine-grained correlation representation that reflects various reaction types of IoT devices and program contexts,
- the PCoExtractor compiler that automatically extracts contexts and correlations defined in IoT services,
- the PCoExtractor runtime that excludes non-correlated devices from a correlation, and precisely and efficiently detects and identifies a faulty device in real-world environments with multiple concurrent user activities, and
- in-depth evaluation with real-world IoT services and data collected with 12 people for more than 40 days.

2 MOTIVATION

This section describes existing faulty device detection schemes and their limitations with a simple IoT environment example, as shown in Figure 2a. Figure 2b shows time series of observed events in the smart home with two users, A and B. User A performs a user activity `cooking` in the kitchen, and black sticks and arrows in Figure 2b illustrate events triggered by cooking. User B enters the house, and gray sticks and arrows in Figure 2b show events triggered by the entrance.



(a) Correlation Training Phase



(b) Faulty Device Detection Phase

Fig. 3. Overall process of existing faulty device schemes. G_x in Figure means unique ID for each trained correlation.

```
1  /* *** Occupancy Checker Example *** */
2  subscribe(door.contact,doorHandler)
3  subscribe(motion.detected,presenceHandler)
4  subscribe(vibration.detected,presenceHandler)
5
6  def doorHandler(evt) {
7    if(evt.value == open) {
8      state.uncond = false
9      state.isDoorClosed = false
10   } else {
11     state.isDoorClosed = true
12     runIn(20,checkNotOccupied)
13 } }
14
15 def presenceHandler(evt) {
16   /* evtHandler for positive value transition
17      of motion & vibration sensors */
18   if(state.isDoorClosed && !state.uncond) {
19     if(state.isOccupied == false) {
20       state.uncond = true
21       state.isOccupied = true
22       turnOnHVAC()
23 } } }
24
25 def checkNotOccupied(){
26   if(!state.uncond && state.isDoorClosed) {
27     if(state.isOccupied) {
28       state.isOccupied = false
29       turnOffHVAC()
30 } } }
```

Fig. 4. Example codes of Occupancy Checker that computes isOccupied context in Figure 1.

2.1 Correlation-based Faulty Device Detection

To guarantee the reliable execution of IoT services, existing work [6, 19, 20, 25, 45–47] proposes correlation-based faulty device detection schemes. Figure 3a shows how the existing schemes extract

correlations by training a series of events. First, they collect a series of raw event data within a predefined time window (T_w). Then, they construct a correlation as a bit vector, each bit of which represents an event occurrence for each device. For example, the first bit of the bit vector represents the event occurrence of the door sensor (Door.2). If there is an event from the door sensor during T_w, the schemes set the first bit as 1. After constructing the correlation bit vectors, the existing schemes assign a unique id for each correlation (G_x), train meaningful correlations with the constructed correlations, and record meaningful correlations with their unique id into a possible correlation database (cDB). Moreover, they train the transition between the correlations over time with a possible transition database (tDB).

At runtime, with the two generated databases, the existing schemes detect a faulty device (Figure 3b. First, they construct a correlation with a bit vector from observed events during T_w. To detect a faulty device, they check whether the current constructed correlation exists in the possible correlation database (cDB). Since the possible correlation database contains observed correlations from training data, if there is no same correlation in the cDB, existing schemes regard one of the correlation bits and its corresponding device faulty. After checking the cDB, they check whether a transition from the previous to the current correlations is valid with the possible correlation transition database (tDB). If the existing schemes detect a fault during the detection phase, they try to identify which device is faulty.

2.2 Limitations of Existing Schemes

Although the existing faulty device detection schemes [6, 19, 20, 25, 45–47] extract correlations among IoT devices and detect a faulty device that violates the trained correlations, they are not suitable for real-world IoT environments with multiple concurrent activities due to following four reasons.

Coarse-grained time unit: The existing schemes cannot distinguish multiple activities in the same time window due to the coarse-grained time unit. Figure 3a shows how the existing schemes construct the correlation with training data. The schemes observe events from the devices during T_w and generate a bit vector that represents a correlation. However, the generated bit vector embeds two activities that have a causal relation. For example, 'LightOn' of the bulb depends on the events 'Motion.2', 'Door.2', and 'Illum' due to the smart light service. Although the 'LightOn' event should be reported after the three events, the existing schemes consider all the events occurred at the same time, so may miss a fault such as the bulb 'LightOn' before the 'Motion.2' event.

Limited correlation representation: Due to their limited correlation representation on events and their reactions, the existing schemes cannot differentiate different activities. For example, the existing schemes represent 'Opening a door', 'Leaving a door open', 'Closing a door', and 'Leaving a door closed' events with a bit value 1 in their correlation. Thus, the existing schemes do not distinguish different events on an IoT device like 'Door.2'. Even if the 'Door.2' has a fault and generates 'Closed' event instead of 'Open', the schemes cannot detect the fault of the 'Door.2' event due to its limited correlation representation.

Context oblivious correlation: The existing schemes construct their correlations only using raw event data without reflecting the program status such as contexts. Disregarding the contexts in correlation construction may cause extracting imprecise correlations because the same activity may cause different reactions depending on its contexts like room occupancy. For example, 'Motion1.Detected' is a valid event when `Room.isOccupied` is true, but 'Motion1.Detected' means a fault when `Room.isOccupied` is false.

However, since the existing schemes do not reflect the contexts in their correlations, they cannot recognize the difference and cannot catch the fault. For example, Figure 4 explains how IoT services compute the context `isOccupied` for the room. The event handlers in Figure 4 check all events

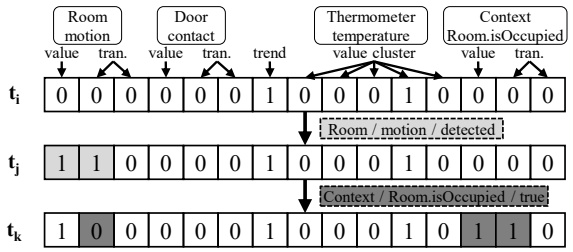


Fig. 5. Fine-grained event representation of Fig. 1 using bit vector.

from door, motion, and vibration sensors with subscribe function. If there is an event from motion or vibration sensors while the room is sealed, then there is at least one person in the room. Moreover, the context `isOccupied` is unconditionally true until opening the door (`state.uncond`). When a user opens the door, then the doorHandler function initializes the value of `state.uncond`. When the user closes the door, the doorHandler function sets the timer to check that the room is not occupied if there is no motion or vibration event for 20 seconds (at line 12). Although there is a state variable `state.isOccupied` that reflects the context `isOccupied`, existing schemes do not have any APIs, programming model, or compiler to extract the contexts in the source codes.

False-positive correlations from non-correlated devices: In real-world environments with concurrent activities, multiple users perform independent activities concurrently. To detect a faulty device in real-world environments, the existing schemes should train all possible combinations of user activities that cause various constructed correlations. Moreover, two correlations overlapped in various ways cause a large number of possible correlations. With limited training data, the existing schemes may not extract and train correlations sufficiently to detect a faulty device. For example, Figure 3b shows how existing schemes detect a faulty device. The existing schemes try to train not only correlations between actually correlated devices such as 'Door.2', 'Motion.2', 'Illum', and 'Bulb', but also non-correlated devices such as 'Illum' and 'Motion.1' in Figure 2a. Due to the gray bits in Figure 3b that represent the status of non-correlated devices, existing schemes need a huge amount of training data and may fail to detect a faulty device with false-positive errors.

3 FINE-GRAINED CORRELATION REPRESENTATION

To enhance the expressiveness of the correlation, this work newly defines a fine-grained correlation representation. The proposed correlation representation reflects fine-grained events from generic sensors, actuators, and contexts.

Binary sensor: A binary sensor is a sensor that measures a physical reaction that can be quantized to a Boolean value. For example, a motion sensor in Figure 1 senses a motion event and represents the value as motion-is-*detected* or motion-is-*not-detected*. Although the existing schemes only consider the occurrence of events with a bit, this work analyzes fine-grained events from a binary sensor and uses three bits that represent the value and its positive and negative transitions. The first bit represents the current status of the binary sensor. If a motion sensor senses the event *detected*, then the value is set with *detected* until the motion sensor senses *not-detected*. A positive transition means the transition of the sensor value from false to true. In the case of the motion sensor, if the value changes from *not-detected* to *detected*, the positive transition becomes true. Conversely, a negative transition represents the value changes from true to false.

Numeric sensor: A numeric sensor is a sensor that measures a specific degree with a numeric value. Existing work [25, 45–47] does not consider numeric sensor and other approaches [19, 20] only consider the occurrence of event. DICE [6] reflects three features of a numeric sensor such as

skewness, increasing or decreasing trends, and whether the value exceeds the mean value. However, none of them directly uses an actual value of the numeric sensor even though the actual value of the numeric sensor is strongly correlated with other events. For example, the temperature of the thermometer in Figure 1 is directly correlated with the context *isHot*. Thus, PCoExtractor reflects an actual value and its increasing or decreasing trend to indicate fine-grained events from a numeric sensor. PCoExtractor uses a clustering algorithm with training data to generate a flexible number of clusters and represents each cluster with a bit. If a value of a numeric sensor belongs to a specific cluster, then the bit that represents the cluster is set. Moreover, PCoExtractor uses additional two bits to represent the value that is smaller than the minimum value and larger than the maximum value. Also, PCoExtractor uses another bit to indicate an increasing or decreasing trend of value. If a numeric sensor generates a larger value than the previous event, then the bit is set.

Actuator: An actuator has various functionalities such as *cooling* and *heating*. Each functionality has different semantics and physical impacts. For example, an air conditioner in Figure 1 has two functionalities, *cooling* and *heating*. When the home is occupied and hot, then smart HVAC service executes *cooling*. Thus, *cooling* is correlated with the contexts such as *isOccupied* and *isHot*. Moreover, *cooling* lowers the temperature of the home. Therefore, *cooling* is also correlated with the decreasing trend of temperature of thermometer. However, *heating* is correlated with *isCold* and increasing trend of temperature instead of *isHot* and decreasing trend of temperature. Therefore, PCoExtractor assigns a bit for each function of actuator to reflect its different semantics and correlations.

Context: PCoExtractor categorizes a context as a binary sensor or numeric sensor. The contexts such as *isOccupied* and *isHot* are binary contexts and a context like *discomfort index* are numeric contexts. PCoExtractor assigns bits for a context using the same scheme with binary and numeric sensor.

As a result, PCoExtractor generates a bit vector for each event to represent fine-grained events from IoT devices. Figure 5 shows an example of fine-grained correlation representation. When the room motion sensor detects a motion at time t_j, two bits that represent value and positive value transition of desk motion sensor are set with 1. Then, smart HVAC service computes the occupancy of the room and changes the value and positive transition bits of context *Room.isOccupied* to 1. The positive transition bit of the room motion sensor is automatically set with 0.

4 PCOEXTRACTOR COMPILER

The PCoExtractor compiler analyzes IoT services and extracts contexts to support fine-grained correlation representation in Section 3. Moreover, the PCoExtractor compiler extracts statically defined correlations in IoT services to remove false-positive correlations.

4.1 Programming Model of PCoExtractor

Context-aware programming models play an important role in IoT environments because existing work [11, 15, 18, 31, 34, 49] emphasizes the importance of context-awareness in many fields such as security and fault-tolerance of IoT services. To support context-awareness, commercial platforms [1, 16, 17, 28, 35] and existing researches [15, 31, 32, 49] propose context-aware programming models. To implement the PCoExtractor compiler with context-aware programming model, this work adapts the programming model of Samsung SmartThings [35].

With an annotation, the PCoExtractor compiler allows programmers to implement a context-aware IoT service that consists of two types of applets such as *Service Logic* and *Context Generator*. The PCoExtractor compiler supports SmartThings [35] like event-driven programming models to implement both types of applets. First, the PCoExtractor compiler allows programmers to implement the main logic of the service with multiple event handlers that react to events from IoT devices and

```
1  /* *** Smart HVAC Example *** */
2  subscribe(Occupancy,occupancyHandler)
3  subscribe(isHot,hvacHandler)
4
5  /* Senses the changes of "Occupancy" variable and tries to control HVAC */
6  /* "controlHVAC" function checks the value of state.isHot and control the air conditioner*/
7  def occupancyHandler(evt){
8    if(evt.value)
9      controlHVAC()
10   ... }
11
12 /* Senses the changes of "isHot" variable from TempChecker (CtxGenerator)
13    and sets the global variable "state.isHot" */
14 def hvacHandler(evt) {
15   if(evt.value == "hot")
16     state.isHot = true
17   else if(evt.value == "cold")
18     ... }
19
20 /* *** Occupancy Checker Example *** */
21 CtxDecl(Occupancy,state.isOccupied)
22
23 subscribe(motion.detected,presenceHandler)
24 subscribe(vibration.detected,presenceHandler)
25
26 def presenceHandler(evt) {
27   if(state.isDoorClosed && !state.uncond) {
28     if(state.isOccupied == false) {
29       state.uncond = true
30       state.isOccupied = true
31 } } }
32
33 def checkNotOccupied(){
34   if(!state.uncond && state.isDoorClosed) {
35     if(state.isOccupied) {
36       state.isOccupied = false
37 } } }
```

Fig. 6. Example codes of smart HVAC service and Occupancy Checker implemented with PCoExtractor.

contexts. Figure 6 shows an example of smart HVAC service source codes. The service subscribes fine-grained events from IoT devices or contexts, as shown in lines 2 and 3 in Fig. 6. Moreover, `occupancyHandler` and `hvacHandler` functions define the main logic of the smart HVAC service. Second, the PCoExtractor compiler allows programmers to implement a special type of applet, *Context Generator*. A *Context Generator* applet declares and defines a context with multiple event handler functions that compute the context. For example, the programmers implement an occupancy checker applet as a *Context Generator* to compute the occupancy of a room. With an annotation, *CtxDecl*, programmers can declare and define a new context with a context id and a variable that contains a value of the context. For instance, the PCoExtractor compiler allows programmers to declare and define a new context with 'Occupancy' as context id and 'state.isOccupied' as a variable that contains the value of 'Occupancy'. The PCoExtractor compiler automatically inserts code to publish fine-grained events from contexts, and subscribers such as the smart HVAC service can subscribe to fine-grained events of contexts with the existing subscribe function.

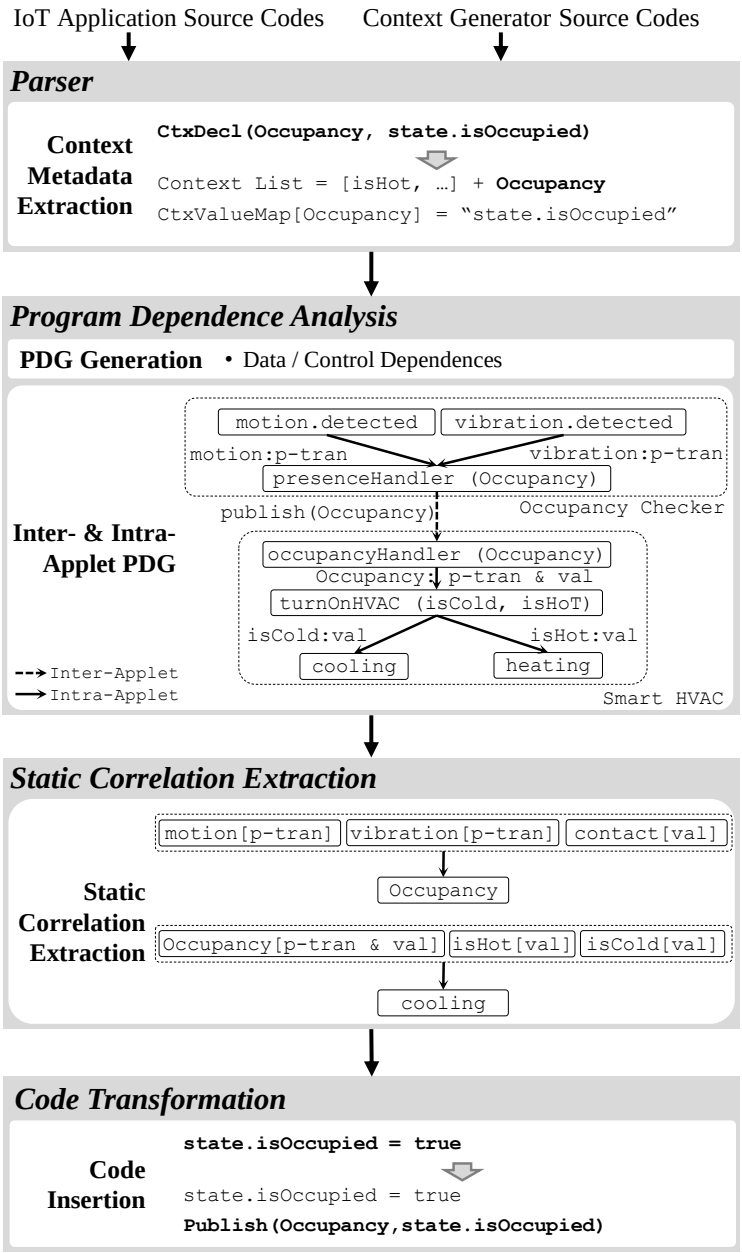


Fig. 7. Overall structure of the PCoExtractor compiler. In program dependence analysis, "A -> B" means that B depends on A.

4.2 Overall Structure

The PCoExtractor compiler consists of four modules such as a parser, program dependence analysis, static correlation extraction, and code transformation. The parser extracts context information from source codes and the static correlation extraction module extracts static correlation using the program

dependence analysis module. The code transformation module inserts communication codes to deliver contexts across the applets.

4.3 Context Extraction

To extract contexts from the source codes, the parser parses the annotation, *CtxDecl*. *CtxDecl* declares and defines a context with a string identifier and a variable that represents a value of the context. First, the parser generates a context list which is a list of context identifiers. Second, the parser builds a key-value map that has a context identifier as key and variable identifier as value. With the context list, the PCoExtractor runtime recognizes the contexts to reflect fine-grained events of the contexts. Moreover, the program dependence analysis module takes the context value map to generate a program dependence graph (PDG) with fine-grained events. The parser in Figure 7 shows an example of the parsing process with the 'Occupancy'.

4.4 Static Correlation Extraction

To extract statically defined correlations from the services, the PCoExtractor compiler has program dependence analysis and static correlation extraction modules.

For the first step, the program dependence analysis module generates a program dependence graph with intra-applet control and data dependence. Unlike usual PDG, the program dependence analysis module marks every vertex in PDG with fine-grained events of IoT devices or contexts. For example, 'state.isOccupied' depends on events from motion and vibration sensors because subscribe functions at lines 19 and 20 in Figure 6 subscribe 'detected' events from motion and vibration sensors. Thus, the positive transition event of 'Occupancy' triggered by line 26 in Figure 6 is correlated with positive transition events of motion and vibration sensors. If the subscribe function monitors general events like line 3 in Figure 6, then the program dependence analysis module treats that handler function depends on both positive and negative transition events. With this procedure, the program dependence analysis module generates multiple intra-applet PDG with fine-grained events as shown in Figure 7.

Second, the program dependence analysis module analyzes inter-applet dependence caused by contexts and generates inter-applet PDG. The program dependence analysis module checks a context value map from the parser and searches the instructions that define the values of the contexts. If there is any write instruction for the value of the context in a target applet, then the program dependence analysis module sets a dependence between the target applet and an applet that subscribes the context. For example, the occupancy checker in Figure 6 modifies the value of 'Occupancy' with an instruction at line 26 and smart HVAC service subscribes the events of 'Occupancy' with the subscribe function call at line 3. Then, the program dependence analysis module sets a dependence between an instruction at line 26 and 'occupancyHandler' function in smart HVAC service. Finally, the program dependence analysis modules generate inter- and intra-PDG as shown in Figure 7.

For the final step, the static correlation extraction module extracts correlations between fine-grained events using generated PDG. With marked fine-grained events, the static correlation extraction module recursively visits child nodes of target event and accumulates marked fine-grained events to generate new static correlations. For example, 'cooling' event depends on the value of the 'isHot'. To execute 'cooling' event, smart HVAC service must execute 'turnOnHVAC' and 'occupancyHandler'. Since the execution of 'turnOnHVAC' depends on the value and positive transition of the 'Occupancy' context, 'cooling' is correlated with them either. Finally, the static correlation extraction module extracts all static correlations among fine-grained events.

4.5 Finalization

To finalize the compilation and generate executable binaries, the code transformation module inserts communication codes for context and compiles the source codes into the binaries. The code

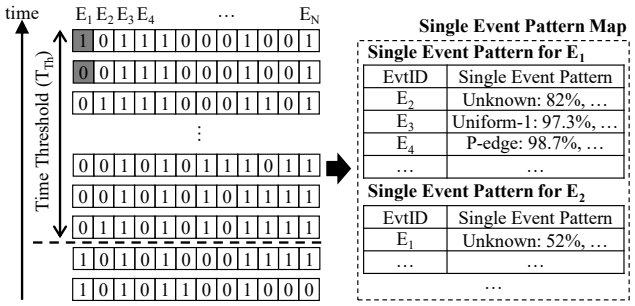


Fig. 8. Training process for correlations among fine-grained events.

transformation module finds 'write' instructions for the value of the contexts and inserts call instructions of `publish` function with the arguments such as context identifier and value of the contexts. For instance, the code transformation module inserts `publish` function that sends the value of 'Occupancy' at runtime.

5 PCOEXTRACTOR RUNTIME

The PCoExtractor runtime consists of training and detection phases. In training phase, the PCoExtractor runtime extracts dynamically defined correlations (Section 5.1.1) and trains correlations and their transitions for each fine-grained event (Section 5.1.2). In detection phase, the PCoExtractor runtime detects probable faulty devices that violate trained correlations and transitions (Section 5.2.1) and identifies an actual faulty device with intersecting probable faulty devices (Section 5.2.2).

5.1 Training Phase

The training phase consists of three processes, disregarding non-correlated events, correlation training, and transition training. The PCoExtractor runtime uses trained correlations among fine-grained events to ignore meaningless correlations among irrelevant events.

5.1.1 Disregarding non-correlated Events. With fine-grained correlation representation in Figure 5, the PCoExtractor runtime represents current status of IoT environment with a bit vector. Each bit of the bit vector means a fine-grained event which is defined in Section 3. Thus, value of each event representation can be zero or one. Therefore, this work categorizes each event's patterns into five patterns such as *Occurred*, *P-edge*, *N-edge*, *Uniform-1*, and *Uniform-0*. *Occurred* pattern means that there is an unique bit with different value from others. *P-edge* represents the value of an event is changed once from zero to one. *N-edge* has opposite meaning to *P-edge*. *Uniform-1* indicates all bits of an event in predefined time threshold is one and *Uniform-0* has opposite meaning of that. If the pattern is not one of proposed five patterns, then the PCoExtractor runtime marks it *Unknown*.

The PCoExtractor runtime checks other events' patterns when the value of the target event bit is changed. For example, Figure 8 shows a simple example of the correlation training process. When the value of event E_1 is changed from zero to one, the PCoExtractor runtime checks patterns of other events (from E_2 to E_N) in the predefined time threshold (T_{Th}). Then, the PCoExtractor runtime generates a single event pattern map by checking the whole training data. If the occurrence rate of a specific pattern exceeds the threshold, then the PCoExtractor runtime treats two events are correlated.

5.1.2 Correlation and Transition Training. After extracting correlated events, the PCoExtractor runtime trains correlations and their transitions. The correlation database in Figure 9 contains a list of correlated fine-grained events for each event using static and dynamic correlations from the

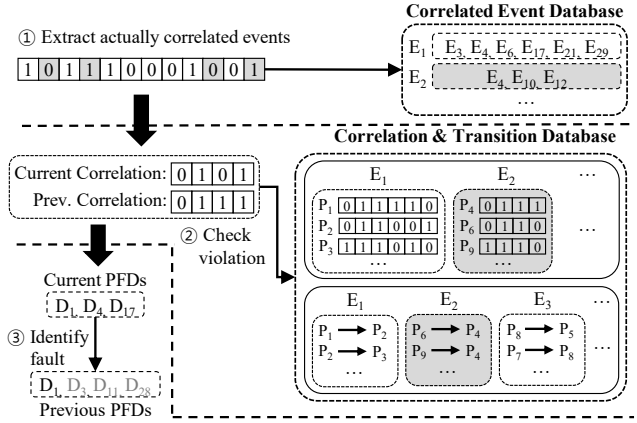


Fig. 9. Failure detection and identification process of the PCoExtractor runtime. E_x, P_x, and D_x mean unique IDs of the fine-grained event, event pattern, and device. PFD means probable faulty devices.

PCoExtractor compiler and correlation training. At the correlation training phase, the PCoExtractor runtime builds a new bit vector that contains bits of only correlated events from a bit vector in a predefined time threshold. Then, the PCoExtractor runtime gives a correlation identifier for each unique correlation and saves the correlation in the correlation database as shown in Figure 9. Moreover, the PCoExtractor runtime trains the transitions among the correlations and saves all possible transitions in the transition database.

Correlation and transition training methods of the PCoExtractor runtime do not need activity labeling or user interventions that degrade the applicability and usability of the system. Also, the PCoExtractor correlation and transition training are suitable for real-world IoT environments with concurrent activities because the PCoExtractor runtime trains correlations among only correlated fine-grained events to remove ambiguity from concurrent activities.

5.2 Detection Phase

In detection phase, the PCoExtractor runtime detects and identifies a faulty device in IoT environments with trained correlations and transitions. Figure 9 shows overall process of faulty device detection and identification.

5.2.1 Faulty Device Detection. To detect a faulty device, the PCoExtractor runtime uses correlation and transition database and checks the current status of IoT environment is unfamiliar. When a new event occurs, the PCoExtractor runtime updates the bit vector that contains whole bits of fine-grained events. Then, the PCoExtractor runtime extracts a precise bit vector with only correlated fine-grained events using a correlated event database (The arrow at the top of the Figure 9). For next, the PCoExtractor runtime checks a current correlation (bit vector) exists in the correlation database (The arrow of step 2 in the Figure 9). If the correlation exists, then the PCoExtractor runtime checks the transition. If the correlation does not exist in the database, then the PCoExtractor runtime reports device fault and starts the identification process. Checking transition is similar to checking correlation. If there is a trained transition from the previous pattern to the current pattern, then the PCoExtractor runtime finishes the detection process. If there is no matched transition, the PCoExtractor runtime reports a device failure and proceeds the identification process.

5.2.2 Failure Identification. To identify a faulty device, the PCoExtractor runtime considers two cases. If a correlation violation is detected in the detection process, the PCoExtractor runtime compares the current correlation with all trained patterns. Then, the PCoExtractor runtime checks the difference between the correlations and figures out the correlations that the differences of bits are caused by a single device. As a result, the PCoExtractor runtime finds a set of IoT devices as a set of probable faulty devices. If a transition violation is detected in the detection process, the PCoExtractor runtime checks all possible transitions from the previous pattern. To specify a faulty device, the PCoExtractor runtime checks all possible current correlations if there is no device fault and transitions between previous and possible current correlations. Then, if the PCoExtractor runtime finds the transition in the transition database, the PCoExtractor runtime adds another probable faulty device that makes difference in the bit vector.

For each real-time event, the PCoExtractor runtime generates a set of probable faulty devices. To reduce the number of candidates, the PCoExtractor runtime counts the number of occurrences of a specific device in probable faulty device set. If the count exceeds the threshold, the PCoExtractor runtime reports a faulty device to users.

6 EVALUATION

This work implements PCoExtractor on top of the LLVM compiler infrastructure [22]. To evaluate PCoExtractor, this work designs and implements 4 real-world IoT services with 12 contexts and collects the data from 68 devices. To prove that PCoExtractor extracts correlations, detects a device failure, and identifies an exact faulty device precisely and efficiently, this work measures accuracy, detection and identification time, and runtime delay of detection and identification processes. For the evaluation, this work uses a desktop server (Intel Core i7-6700, 16GB) with Ubuntu 18.04.

6.1 Experimental Setup

This section describes four public datasets from Kasteren dataset [40] and CASAS dataset [9, 43] that are also used by existing work [6, 19, 20, 47]. Moreover, this work deploys 68 devices in a laboratory where 12 people work, implements and runs 4 IoT services with 12 contexts on the laboratory, and generates two datasets with different combinations of the devices to validate PCoExtractor using more realistic data containing concurrent activities.

6.1.1 Public Dataset. HouseA, HouseB, and HouseC datasets from Kasteren dataset [40] and HH102 dataset from CASAS dataset [9, 43] contain the data collected in smart home. The datasets such as HouseA, HouseB, and HouseC collect data from only binary sensors such as contact, motion, pressure, and switch sensors, and show simple patterns of events due to the limited number of sensors and binary sensor types. HH102 dataset includes data from multiple binary sensors, numeric sensors, and actuators and generates various patterns and their transitions. However, the public datasets are collected from testbeds with a single user who performs only one activity at a certain time. Moreover, the public datasets do not contain static correlations written in IoT services.

6.1.2 Smart Lab Dataset. Although the Kasteren and CASAS datasets contain various activities and a large amount of sensor data, the datasets still lack realistic features such as concurrent activities from multiple users and the IoT services running on the testbed. Therefore, this work sets up a testbed on a laboratory where 12 people work (referred as Smart Lab dataset). As shown in Figure 10, this work deploys three kinds of multi-purpose sensors and air purifier. The sensors collect 11 types of data such as motion, temperature, and etc. This work treats a sensing event of a deployed device as an individual IoT device because each sensor in a device can be failed independently. The major difference between the public datasets and Smart Lab dataset is that multiple users may perform

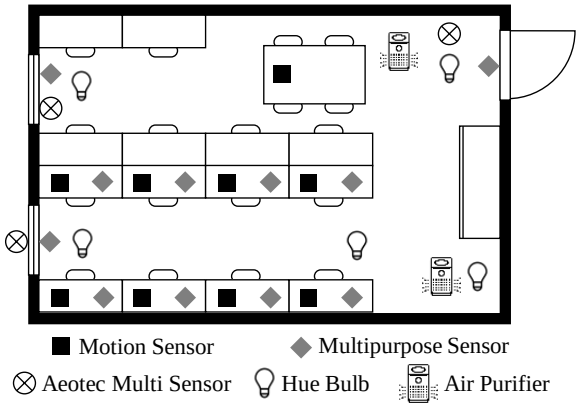


Fig. 10. Floor plan of smart laboratory testbed

diverse activities at the same time, making the fault detectors hard to extract the correlations of the events.

The Smart Lab dataset consists of two sub-datasets. The first dataset referred as *Lab-large* contains all the deployed devices. With *Lab-large* dataset, this work can validate that PCoExtractor extracts precise correlations efficiently in spite of a large number of devices. The second dataset referred as *Lab-small* includes half of the deployed motion and multipurpose sensors, and uses all of Aeotec multi-sensors, Hue bulbs, and air purifiers. Using *Lab-small* dataset, this work can prove that PCoExtractor works well with lower correlation degree that indicates how many correlations exists among the devices.

To make a realistic testbed, this work analyzes 287 SmartThings services [8, 29] and designs services and contexts based on the commonly used services. Then, this work implements and runs 4 IoT services such as smart HVAC, smart light, smart surveillance, and smart alarm with 12 contexts on the testbed.

Smart HVAC service checks that the laboratory is occupied and monitors temperature and humidity of both inside and outside of the laboratory to alert users if temperature and humidity are not in a normal range. Moreover, smart HVAC service monitors air quality, odor level, dust level, and fine dust level of the laboratory and activates or deactivates the air purifiers.

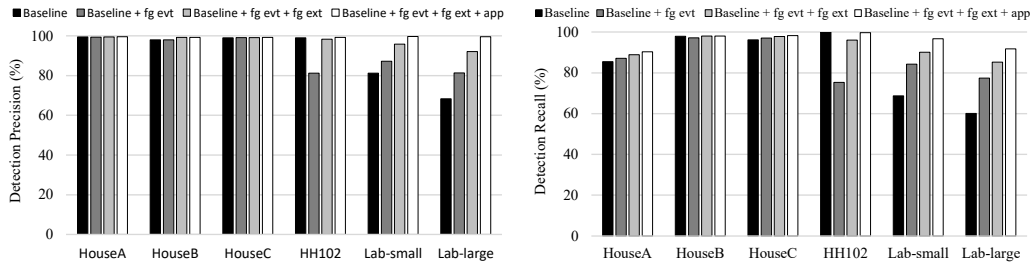
Smart light service monitors the illuminance of both inside and outside of the laboratory and turns on or off the Hue bulbs to maintain proper illuminance. Smart light service controls the bulbs when only the laboratory is occupied.

Smart surveillance service checks if the laboratory is sealed when all users leave the laboratory and alerts a user if the windows are open or the door is unlocked. Moreover, a smart surveillance service reports an intrusion if someone tries to enter the laboratory in a predefined time.

Smart alarm service reads schedules from web calendar and alerts to a user before the schedule. If the user is not in the seat, the smart alarm service sends a text message or push alarm based on the importance of a schedule. If the user is in the seat, the smart alarm service blinks the Hue bulbs to notify the user.

6.2 Evaluation Methods

This work chooses DICE [6] as a baseline because only DICE supports all types of IoT devices such as binary sensor, numeric sensor, and actuator. For the evaluation, this work implements DICE with C++ language same as PCoExtractor.



(a) Detection Precision (b) Detection Recall

Fig. 11. Precision and recall of device failure detection of PCoExtractor. *fg evt* means fine-grained correlation representation, *fg ext* means fine-grained correlation extraction, and *app* means service semantic-aware correlation extraction.

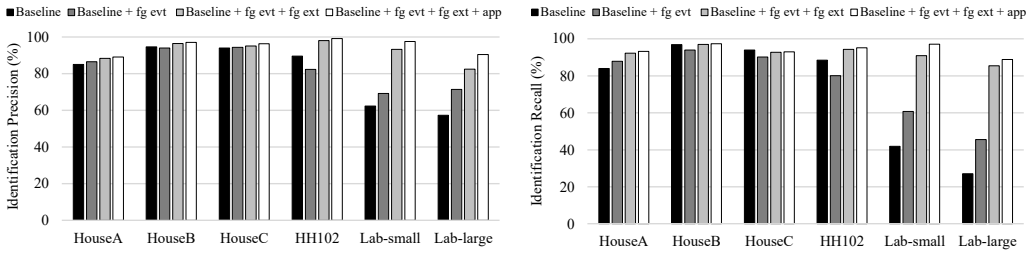
This work performs a training phase for 300 hours for public datasets and 500 hours for Lab dataset. In the evaluation phase, this work generates random faults on the rest of the datasets; the fault generator divides the provided datasets into 100 segments in time series, and randomly generates five kinds of faults (fail-stop, outlier, spike, stuck-at, high-noise). To validate each contribution's effectiveness, this work incrementally applies fine-grained correlation representation, fine-grained correlation extraction, and service semantic-awareness (static correlation and context extraction) to baseline [6]. Then, the second method uses fine-grained event representation (Section 3). Next, the third method applies fine-grained event representation with fine-grained correlation extraction that finds meaningful correlations on all event occurrence (Section 5). Finally, the fourth method is PCoExtractor that adapts all the contributions, including extracted static contexts (Section 4). This work calculates precision and recall of detection and identification accuracy. This work also measures detection and identification time, and computation delay of four versions.

6.3 Detection and Identification Accuracy

This section evaluates accuracy of detection and identification of PCoExtractor. To prove each contribution's effectiveness, this work measures precision and recall of detection and identification in four versions. Figure 11 shows precision and recall of failure detection.

6.3.1 Detection Accuracy. **Single user and sequential activity case**: For HouseA, HouseB, and HouseC, all versions show high precision due to the simplicity of the dataset. However, the recall of HouseA is slightly lower than HouseB and HouseC because HouseA contains few correlations, and correlation-based approaches may not detect some failures caused by a device with no or few correlations. Precision and recall of HH102 show interesting results. Except for the second version, all versions detect almost every failure, but the second version shows accuracy degradation with at a large scale. Since HH102 contains a lot of sensors and fine-grained event type expands the bit vector greatly, there is a huge number of possible patterns and transitions for the second version. Thus, the second version may need additional training data to train all patterns and transitions. For single user and sequential activity cases, both the baseline scheme and PCoExtractor achieve high precision and recall.

Multiple user and concurrent activity case: To evaluate the accuracy of detection and identification with concurrent activities, this work uses *Lab-small* and *Lab-large* datasets. Detection and recall of four versions with *Lab-small* and *Lab-large* datasets in Figure 11a and Figure 11b show



(a) Identification Precision (b) Identification Recall

Fig. 12. Precision and recall of device failure identification of PCoExtractor. *fg evt* means fine-grained correlation representation, *fg ext* means fine-grained correlation extraction, and *app* means service semantic-aware correlation extraction.

noticeable differences among four versions. These differences prove that each contribution actually affects the accuracy of failure detection. As a result, PCoExtractor detects 40.06% more failures on average than the baseline. Thus, PCoExtractor detects a faulty device more accurately in real-world IoT environments with concurrent activities and each contribution of this paper improves detection accuracy.

6.3.2 Identification Accuracy. **Single user and sequential activity case**: In Figure 12, precision and recall of identification show similar aspects with precision and recall of detection for public datasets. However, precision and recall of identification are lower than detection because the identification needs a higher correlation degree and larger training data to remove ambiguous candidates from probable faulty devices.

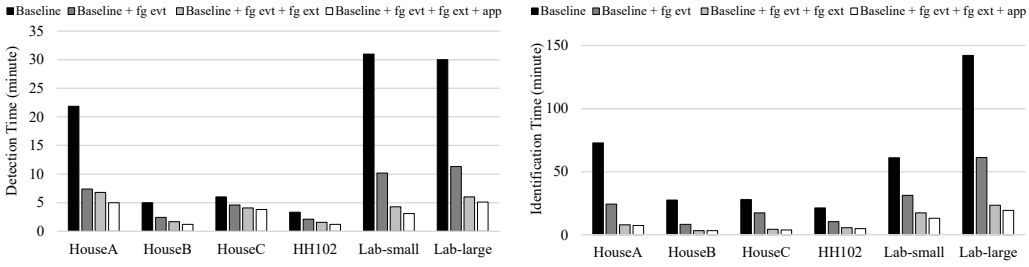
Multiple users and concurrent activity case: Figure 12 shows the results of precision and recall of identification. For two Lab datasets, PCoExtractor identifies 108.3% more device failures than the baseline. Moreover, each contribution improves the precision and recall of baseline noticeable. Since the identification phase needs more precise correlations to specify a faulty device, PCoExtractor successfully extracts precise correlations to identify a faulty device.

6.4 Detection and Identification Time

This work measures detection time as a time gap between actual device failure and the time that the failure detector recognizes the failure, and identification time as a time gap from device failure to identification of the faulty device. In Figure 13, PCoExtractor reduces the detection and identification time extremely. PCoExtractor needs only 24.5% and 14.9% of detection and identification time compared to baseline. In most cases, fine-grained event type reduces detection and identification time greatly because the fine-grained correlation definition enables precise comparison between the correlations.

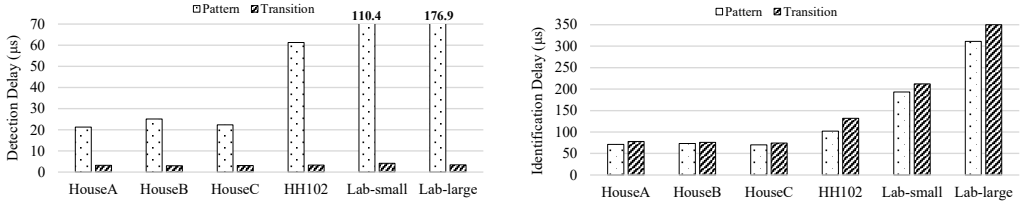
6.5 Detection and Identification Delay

This work measures the computation delay of failure detection. Although the PCoExtractor achieves higher precision and recall for failure detection and identification, the PCoExtractor also achieves reduction of detection and identification delay due to removing non-correlated events. As described in Section 5.2.1, failure detection consists of two phases: correlation check and transition check. This work measures the execution time of fault detector on every event and takes the average of



(a) Detection Time (b) Identification Time

Fig. 13. Detection and identification time of PCoExtractor. *fg evt* means fine-grained correlation representation, *fg ext* means fine-grained correlation extraction, and *app* means service semantic-aware correlation extraction.



(a) Detection Delay (b) Identification Delay

Fig. 14. Detection and identification delay. *Pattern* means the delay caused by correlation check and *Transition* means the delay caused by transition check.

the calculation time. As shown in Figure 14, PCoExtractor dramatically reduces runtime delay by reducing the length of bit vectors. The training phase (Section 5.1.2) of this work select only correlated events from the bit vectors. There is no need to compare every correlation that might be non-correlated to the current bit vector. Consequently, the search space of correlation and transition check shrinks, reducing runtime delay of failure detection.

7 RELATED WORK

Security Analysis of IoT Devices and Platforms: There have been extensive security analyses on both IoT devices [14, 21, 42] and platforms [10, 50].

IoT devices may fail due to malicious attacks [21, 42] or their own faults [14]. Kumar et al. [21] performs large-scale empirical analysis on 83M IoT devices around the world. As a result, a significant fraction of devices is still using weak credentials with insecure FTP/Telnet protocols and vulnerable to known attacks. Compromised IoT devices can not only end on their own failures but can damage other devices (MadIoT [36]) or be exploited for massive DDoS attacks (Mirai botnet [2]). Hnat et al. [14] observed over 350 sensors across 20 homes for 8 months and reported various failures such as process failures, link loss, and sensor failures. Nevertheless, device vendors are small or medium-sized businesses, so they have little incentive to take firm security infrastructures such as monitoring agents and security hardware [42].

Not only IoT devices but also IoT platform itself has its own vulnerabilities [10, 50]. For example, an attacker can remotely replace a victim's real device with a phantom device. This attack may cause the phantom device to send the wrong data, thus deceiving the victim user. This will lead to service failures and put the victim users at risk. Through security analysis of the widely-used Samsung SmartThings IoT platform [35], Fernandes et al. [10] address the overprivileged problem caused by its coarse-grained permission model based on functionally-grouped capabilities. Without any special privileges, an IoT service on the platform can read all events of the device if it is granted with at least one capability. The SmartThings platform also does not verify the event source, so an attacker can spoof physical device events by easily identifying the device identifier.

Therefore, the device failure occurs due to the device's own fault, external attack, or the vulnerable platform, which can lead to service failures and further endanger the user.

Device Failure Management in IoT: There have been various solutions [3, 5, 6, 12, 19, 20, 24–27, 33, 45–47] on how to manage a device failure for preventing a service failure.

To detect a faulty device in the IoT, previous work [5, 6, 12, 19, 20, 24–26, 33, 37, 45–48] propose failure detection schemes based on various correlations of events, sensors and actuators. Against an attacker who is able to spoof events, Peeves [5] learns correlations between events on collected data set without attacks, and classifies whether events are normal at runtime. FailureSense [25] correlates electrical appliances with associated sensor events and learn the regular patterns of sensing events and appliance activation events, then monitors a significant deviation from the regularity at runtime. CLEAN [45–47] clusters sensors based on the similarity of sensor events and detects sensor failure by detecting outlier. DICE [6] represents a state as a set of sensing values at a time and defines correlated sensors as a group of sensors with the same states for a certain period. DICE also checks probabilities of transitions from one state to the other states to detect sensor abnormalities. 6thSense [33] defines correlation as sensors associated with activity that occurs while using a smartphone. Other works [37, 48] find an anomaly from IoT environment by analyzing continuous sensor data with correlation coefficient. They extract correlations from continuous values of numeric sensors and find anomalies that violate the trained correlations.

Some work [19, 20] uses service-level contexts by analyzing IoT services. SMART [19] uses application semantics to detect and recover sensor failures. Although SMART considers service-level failures to train the failure detector, SMART does not use correlations among sensors. Compared to SMART, this work extract correlations among sensors by looking at services. Idea [20] detects redundant sensors in an Activity of Daily Living (ADL; context in this work) and excludes failing sensors to detect the context. Although the previous schemes use ADL rules that contain correlated sensor groups, the previous methods are not adequate to distinguish concurrent and multi-user activities which are not actually correlated. Conversely, this work eliminates the ambiguity of the correlations by discarding inessential events.

Once device failure is detected, fault-tolerant systems such as IoTRepair [27] and Rivulet [3] recover the failure or keep the service working normally. With the failure handling library such as replication, retry, restart, checkpoint, and rollback, IoTRepair [27] can autonomously handle various device failures. Rivulet [3] is a fault-tolerant distributed system that utilizes redundant smart consumer appliances to execute an IoT service reliably by replicating the events across appliances. This work can be integrated with the IoTRepair or Rivulet to provide a more reliable IoT service with the proposed highly accurate and efficient failure detection scheme.

Activity Recognition in IoT: IoT frameworks use contexts to identify how users currently behave and provide IoT services based on the information. To extract the users' contexts, recent publications [4, 7, 23, 30, 38, 39, 44] have proposed the systems that recognize human activities for multi-user environments. Some approaches [4, 30] propose unsupervised recognition methods to identify complex ADL by deriving semantic reasoning between sensors and activities. Other

approaches [7, 38, 44] extend unsupervised learning by using cloud-edge-based IoT frameworks that collect feedbacks from multiple distinct environments and refine correlations and semantics of sensors and activities. Supervised or semi-supervised learning methods [23, 39] are the alternatives of the context extraction. This work currently uses contexts from analysis of static IoT services. By adopting the proposed machine learning-based approaches, this work can improve the precision of contexts.

Correlation-based Failure Detection: Two existing works [13, 41] propose correlation-based anomaly detection methods for specific systems such as adaptive system and Web applications in cloud computing. Although their whole methods are not directly applicable for IoT environments, their core concepts that compute correlation coefficient may improve PCoExtractor. AdaptGuard [13] extracts causality assumptions between variables in adaptive systems, calculates correlation coefficient among them, and detects an assumption violation that harms stability of the system. Tao et el. [41] proposes an automatic fault diagnosis method for Web applications in cloud computing. The work models the correlations between workloads and metrics related to the application performance and resource utilization and detects anomalies by recognizing abrupt change of correlation coefficients.

8 CONCLUSION

This work proposes a new compiler-assisted correlation extraction scheme, named PCoExtractor. PCoExtractor newly defines a fine-grained correlation representation with the context to support precise correlation extraction with concurrent activities. To easily reflect high-level information in IoT services, the PCoExtractor compiler automatically extracts contexts and statically defined correlations from the services. The PCoExtractor runtime traces reactions of sensors, actuators, and contexts during a series of multiple concurrent activities without any time window, excludes non-correlated devices that inconsistently react for the same activities, and construct fine-grained correlations. Finally, the runtime detects a faulty device that violates trained correlations and their transitions. PCoExtractor detects and identifies 40.06% more faults for 4 IoT services with concurrent activities of 12 users. Moreover, PCoExtractor reduces 80.3% of detection and identification time with only 181.7 μs delay.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. We also thank the CoreLab and HPC Lab members for their support and feedback during this work. This work is supported by IITP-2020-0-01847, IITP-2020-0-01361 and IITP-2021-0-00853 through the Institute of Information and Communication Technology Planning and Evaluation (IITP) funded by the Ministry of Science and ICT. This work is also supported by Samsung Electronics. (Corresponding author: Hanjun Kim)

REFERENCES

[1] AllJoyn 2018. *AllJoyn*. Retrieved April 20, 2020 from https://github.com/alljoyn/alljoyn.github.com/wiki

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110.

[3] Masoud Saeida Ardekani, Rayman Preet Singh, Nitin Agrawal, Douglas B. Terry, and Riza O. Suminto. 2017. Rivulet: A Fault-tolerant Platform for Smart-home Applications. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (Las Vegas, Nevada) *(Middleware '17)*. ACM, New York, NY, USA, 41–54. https://doi.org/10.1145/3135974.3135988

[4] G. Azkune and A. Almeida. 2018. A Scalable Hybrid Activity Recognition Approach for Intelligent Environments. *IEEE Access* 6 (2018), 41745–41759. https://doi.org/10.1109/ACCESS.2018.2861004

[5] Simon Birnbach, Simon Eberz, and Ivan Martinovic. 2019. Peeves: Physical Event Verification in Smart Homes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1455–1467. https://doi.org/10.1145/3319535.3354254

[6] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim. 2018. Detecting and Identifying Faulty IoT Devices in Smart Home with Context Extraction. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 610–621. https://doi.org/10.1109/DSN.2018.00068

[7] G. Civitarese, C. Bettini, T. Sztyler, D. Riboni, and H. Stuckenschmidt. 2018. NECTAR: Knowledge-based Collaborative Active Learning for Activity Recognition. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–10. https://doi.org/10.1109/PERCOM.2018.8444590

[8] CommunitySmartapps 2021. SmartThings Smartapps from Community. Retrieved August 1, 2020 from https://community.smartthings.com

[9] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan. 2013. CASAS: A Smart Home in a Box. *Computer* 46, 7 (July 2013), 62–69. https://doi.org/10.1109/MC.2012.328

[10] E. Fernandes, J. Jung, and A. Prakash. 2016. Security Analysis of Emerging Smart Home Applications. In *2016 IEEE Symposium on Security and Privacy (SP)*. 636–654. https://doi.org/10.1109/SP.2016.44

[11] N. Ghosh, S. Chandra, V. Sachidananda, and Y. Elovici. 2019. SoftAuthZ: A Context-Aware, Behavior-Based Authorization Framework for Home IoT. *IEEE Internet of Things Journal* 6, 6 (Dec 2019), 10773–10785. https://doi.org/10.1109/JIOT.2019.2941767

[12] Shuo Guo, Ziguo Zhong, and Tian He. 2009. FIND: Faulty Node Detection for Wireless Sensor Networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (Berkeley, California) *(SenSys '09)*. ACM, New York, NY, USA, 253–266.

[13] Jin Heo and Tarek Abdelzaher. 2009. Adaptguard: Guarding adaptive systems from instability. 77–86. https://doi.org/10.1145/1555228.1555256

[14] Timothy W. Hnat, Vijay Srinivasan, Jiakang Lu, Tamim I. Sookoor, Raymond Dawson, John Stankovic, and Kamin Whitehouse. 2011. The Hitchhiker's Guide to Successful Residential Sensing Deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems* (Seattle, Washington) *(SenSys '11)*. Association for Computing Machinery, New York, NY, USA, 232–245.

[15] Casen Hunger, Lluis Vilanova, Charalampos Papamanthou, Yoav Etsion, and Mohit Tiwari. 2018. DATS - Data Containers for Web Applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. ACM, New York, NY, USA, 722–736. https://doi.org/10.1145/3173162.3173213

[16] IFTTT 2021. IFTTT. Retrieved May 10, 2019 from https://ifttt.com/

[17] IoTivity 2021. IoTivity. Retrieved May 10, 2019 from https://iotivity.org

[18] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *21st Network and Distributed Security Symposium*.

[19] Krasimira Kapitanova, Enamul Hoque, John A. Stankovic, Kamin Whitehouse, and Sang H. Son. 2012. Being SMART About Failures: Assessing Repairs in SMART Homes. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (Pittsburgh, Pennsylvania) *(UbiComp '12)*. ACM, New York, NY, USA, 51–60. https://doi.org/10.1145/2370216.2370225

[20] Palanivel A. Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. 2016. Idea: A System for Efficient Failure Management in Smart IoT Environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services* (Singapore, Singapore) *(MobiSys '16)*. ACM, New York, NY, USA, 43–56. https://doi.org/10.1145/2906388.2906406

[21] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All Things Considered: An Analysis of IoT Devices on Home Networks. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1169–1185.

[22] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization* (Palo Alto, California) *(CGO '04)*. IEEE Computer Society, Washington, DC, USA, 75–86. http://dl.acm.org/citation.cfm?id=977395.977673

[23] J. Liono, F. D. Salim, N. van Berkel, V. Kostakos, and A. K. Qin. 2019. Improving Experience Sampling with Multi-view User-driven Annotation Prediction. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–11. https://doi.org/10.1109/PERCOM.2019.8767394

[24] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *2017 IEEE 37th International Conference on Distributed*

Computing Systems (ICDCS). 2177–2184. https://doi.org/10.1109/ICDCS.2017.283

[25] S. Munir and J. A. Stankovic. 2014. FailureSense: Detecting Sensor Failure Using Electrical Appliances in the Home. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*. 73–81. https://doi.org/10.1109/MASS.2014.16

[26] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. Sadeghi. 2019. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 756–767. https://doi.org/10.1109/ICDCS.2019.00080

[27] M. Norris, B. Celik, P. Venkatesh, S. Zhao, P. McDaniel, A. Sivasubramaniam, and G. Tan. 2020. IoTRepair: Systematically Addressing Device Faults in Commodity IoT. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 142–148.

[28] openHAB 2021. openHAB. Retrieved August 1, 2020 from https://www.openhab.org

[29] PublicSmartapps 2021. SmartThings Public Smartapps. Retrieved August 1, 2020 from https://github.com/SmartThingsCommunity/SmartThingsPublic

[30] Daniele Riboni, Timo Sztyler, Gabriele Civitarese, and Heiner Stuckenschmidt. 2016. Unsupervised Recognition of Interleaved Activities of Daily Living through Ontological and Probabilistic Reasoning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Heidelberg, Germany) *(UbiComp '16)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2971648.2971691

[31] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2018. Situational Access Control in the Internet of Things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. ACM, New York, NY, USA, 1056–1073. https://doi.org/10.1145/3243734.3243817

[32] Chenguang Shen, Rayman Preet Singh, Amar Phanishayee, Aman Kansal, and Ratul Mahajan. 2016. Beam: Ending Monolithic Applications for Connected Devices. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, Denver, CO, 143–157. https://www.usenix.org/conference/atc16/technical-sessions/presentation/shen

[33] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 2017. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 397–414. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sikder

[34] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. 2019. Aegis: A Context-Aware Security Framework for Smart Home Systems. In *Proceedings of the 35th Annual Computer Security Applications Conference* (San Juan, Puerto Rico) *(ACSAC '19)*. Association for Computing Machinery, New York, NY, USA, 28–41. https://doi.org/10.1145/3359789.3359840

[35] SmartThings 2020. Samsung SmartThings. Retrieved May 10, 2019 from http://www.smartthings.com

[36] Saleh Soltan, Prateek Mittal, and H. Vincent Poor. 2018. BlackIoT: IoT Botnet of High Wattage Devices Can Disrupt the Power Grid. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 15–32.

[37] Shen Su, Yanbin Sun, Xiangsong Gao, Jing Qiu, and Zhihong Tian. 2019. A Correlation-Change Based Feature Selection Method for IoT Equipment Anomaly Detection. *Applied Sciences* 9, 3 (2019). https://doi.org/10.3390/app9030437

[38] X. Su, P. Li, J. Riekki, X. Liu, J. Kiljander, J. Soininen, C. Prehofer, H. Flores, and Y. Li. 2018. Distribution of Semantic Reasoning on the Edge of Internet of Things. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–9. https://doi.org/10.1109/PERCOM.2018.8444596

[39] Tao Gu, Zhanqing Wu, Xianping Tao, H. K. Pung, and Jian Lu. 2009. epSICAR: An Emerging Patterns based approach to sequential, interleaved and Concurrent Activity Recognition. In *2009 IEEE International Conference on Pervasive Computing and Communications*. 1–9. https://doi.org/10.1109/PERCOM.2009.4912776

[40] T. L. M. van Kasteren, G. Englebienne, and B. J. A. Kröse. 2011. *Human Activity Recognition from Wireless Sensor Network Data: Benchmark and Software*. Atlantis Press, Paris, 165–186. https://doi.org/10.2991/978-94-91216-05-3_8

[41] Tao Wang, Wenbo Zhang, Jun Wei, and Hua Zhong. 2015. Fault detection for cloud computing systems with correlation analysis. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 652–658. https://doi.org/10.1109/INM.2015.7140351

[42] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. 2019. Looking from the Mirror: Evaluating IoT Device Security through Mobile Companion Apps. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1151–1167.

[43] WSU CASAS Datasets 2020. WSU CASAS Datasets. Retrieved August 1, 2020 from http://casas.wsu.edu/datasets/

[44] J. Ye. 2018. SLearn: Shared learning human activity labels across multiple datasets. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 1–10. https://doi.org/10.1109/PERCOM.2018.8444594

[45] J. Ye, G. Stevenson, and S. Dobson. 2015. Fault detection for binary sensors in smart home environments. In *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 20–28. https://doi.org/10.1109/PERCOM.2015.7146505

[46] J. Ye, G. Stevenson, and S. Dobson. 2015. Using temporal correlation and time series to detect missing activity-driven sensor events. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. 44–49. https://doi.org/10.1109/PERCOMW.2015.7133991

[47] Juan Ye, Graeme Stevenson, and Simon Dobson. 2016. Detecting abnormal events on binary sensors in smart home environments. *Pervasive and Mobile Computing* 33 (2016), 32 – 49. https://doi.org/10.1016/j.pmcj.2016.06.012

[48] Pushe Zhao, Masaru Kurihara, Junichi Tanaka, Tojiro Noda, Shigeyoshi Chikuma, and Tadashi Suzuki. 2017. Advanced correlation-based anomaly detection method for predictive maintenance. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 78–83. https://doi.org/10.1109/ICPHM.2017.7998309

[49] S. Zhou, K. Lin, J. Na, C. Chuang, and C. Shih. 2015. Supporting Service Adaptation in Fault Tolerant Internet of Things. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*. 65–72. https://doi.org/10.1109/SOCA.2015.38

[50] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. 2019. Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1133–1150.