RT-IFTTT: Real-Time IoT Framework with Trigger Condition-aware Flexible Polling Intervals

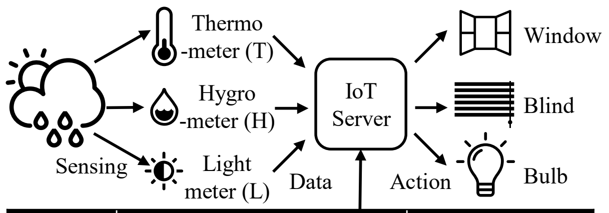
Seonyeong Heo Seungbin Song Jong Kim Hanjun Kim
POSTECH, Republic of Korea
{heosy, sbsong, jkim, hanjun}@postech.ac.kr

Abstract—**With a simple "If This Then That" syntax, IoT frameworks such as IFTTT and Microsoft Flow allow users to easily create custom applets integrating sensors and actuators. Users expect appropriate actions to be taken within a certain latency in response to sensor value changes while the sensors usually have limited battery power. Therefore, reading the sensor values at the right time point is crucial for the IoT frameworks to support real-time responses of the applets while saving battery lives of sensors. However, existing IoT frameworks periodically read the sensor data with fixed intervals without reflecting current sensor values and trigger conditions of applets, so the intervals are either too long to meet the real-time constraints, or too short wasting batteries of sensors. This work extends the existing IFTTT syntax for users to describe real-time constraints, and proposes the first real-time IoT framework with trigger condition-aware flexible polling intervals, called RT-IFTTT. RT-IFTTT analyzes current sensor values, trigger conditions and constraints of all the applets in the framework, and dynamically calculates the efficient polling intervals for each sensor. This work collects real-world sensing data from 10 physical sensors for 10 days, and shows that the RT-IFTTT framework with the proposed scheduling algorithm executes 100 to 400 applets according to user-defined real-time constraints with up to 64.12% less sensor polling counts compared to the framework with the fixed intervals.**

I. Introduction

In the Internet of Things (IoT) environments, networked sensors and actuators provide various services that assist daily lives of users, interacting with the physical world. To satisfy various user demands, recent IoT frameworks such as IFTTT [1] and Microsoft Flow [2] allow users to write their own custom service as an applet with a simple "If This Then That" syntax. Fig. 1 illustrates an example smart home IoT framework and its custom applets. In the example, the custom applets describe several operational rules such as ventilating a room if sunny, closing blinds if too bright, turning on/off a bulb if dark/bright, and closing a window if rainy. Given the applets, the IoT framework periodically reads current temperature, humidity, and illuminance values from connected sensors, and executes described actions in the applets if their trigger conditions are true. Here, some of the applets are time critical. For example, if the framework does not close the window on time when it is rainy, the inside of the house will get wet. If the applets include more time critical services like health monitoring and disaster response, their real-time responses become more important [3–6].

To support real-time responses of the applets, the IoT frameworks should read sensor data at the right time point.



Applets	Trigger Conditions	Actions
App_vent	T > 25 °C && L > 25,000 lux	Window → Open
App_blind	L > 50,000 lux	Blind → Close
App_day	L > 20,000 lux	Bulb → Off
App_night	L < 1,500 lux	Bulb → On
App_rainy	H > 75% && L < 1,000 lux	Window → Close

Fig. 1. Example Applets: Smart Home

The response time is from when the physical event occurs to when the actuators react. To react to the event on time, the IoT framework should first notice the occurrence of the event from the sensor data. However, IoT sensors generally have limited battery power, so the sensors measure the environments only at the scheduled time and hibernate after the measurement. If the interval between each measurement is too long, the IoT frameworks cannot take actions of the applets within the real-time constraints. On the other hand, if the interval is too short, each sensor unnecessarily wastes its battery power. As results, to maximize battery lifetimes of the sensors while supporting real-time responses, the IoT frameworks should read the sensor values with an appropriate interval that reflects real-time constraints.

However, existing IoT frameworks [1, 2, 7, 8] periodically read the sensor data with fixed intervals regardless of real-time constraints. Moreover, the frameworks do not provide any interface about real-time constraints, so there is no way for users to specify a deadline of each applet. For example, the IFTTT syntax consists of only trigger conditions and actions without real-time constraints, and the IFTTT framework polls all the connected sensors every 15 minutes [1]. The IFTTT framework provides real-time APIs that allow a sensor to notify the framework of any change, and reduces the response time of the applets. However, since the real-time APIs make the sensor send notifications for every change, the sensor sends unnecessary notifications that do not affect trigger conditions, wasting its battery power. Publish-and-subscribe messaging protocols [9, 10] that many IoT frameworks [2, 7, 8] adopt also require a sensor to publish its change without considering trigger conditions of target applets, so the sensor suffers from the same unnecessary notification problem like the real-time APIs of the IFTTT framework.

This work proposes RT-IFTTT, the first real-time IoT language and its framework that uses trigger condition-aware flexible sensor polling intervals. The RT-IFTTT language extends the existing IFTTT syntax, and allows users to specify real-time constraints for their applets. Its compiler analyzes the applets and extracts their trigger conditions, actions and deadlines. The RT-IFTTT framework consists of an applet manager and a sensor polling scheduler. The applet manager analyzes prerequisite relations among applets, and classifies an applet as an inactive applet if at least one of its prerequisites is not triggered. The sensor polling scheduler dynamically calculates evaluation and polling intervals for each trigger condition and related sensors, reflecting their current sensor values and corresponding trigger conditions.

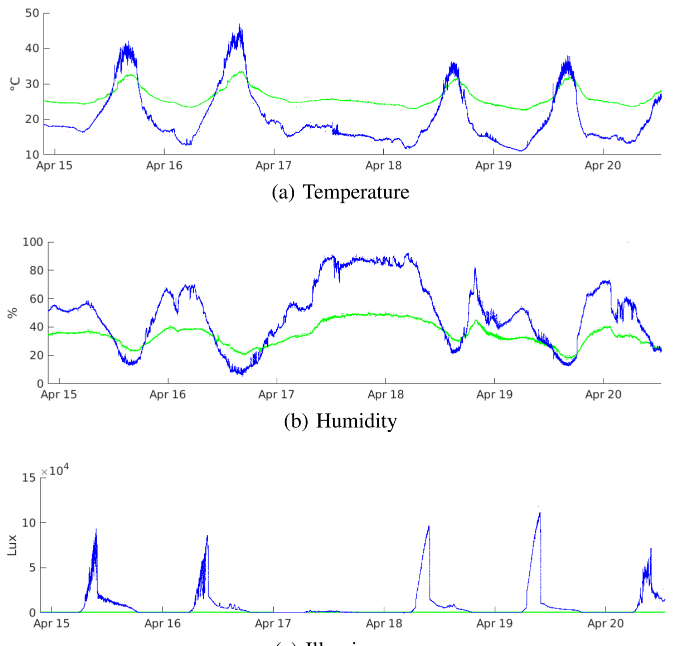
This work implements the RT-IFTTT framework and installs 10 physical sensors that consist of 2 sets of temperature, humidity, UV Index, ambient light, and pressure sensors. The RT-IFTTT framework collects sensing data from the sensors for 10 days. To deeply evaluate the proposed scheduler in the RT-IFTTT framework, this work also implements three other schedulers including fixed and random interval schedulers, and executes the schedulers with the collected data. The evaluation results show that the RT-IFTTT framework with the proposed scheduling algorithm executes 100 to 400 applets with up to 64.12% less sensor polling counts compared to the framework with the fixed intervals.

The contributions of this paper are:

- The first real-time IoT framework with flexible polling intervals, called RT-IFTTT
- The RT-IFTTT syntax and compiler that allow users to write their own IoT applets with real-time constraints
- The applet manager that analyzes prerequisite relations among trigger conditions
- The flexible polling interval model and its scheduler that dynamically calculate efficient polling intervals for each sensor reflecting the current sensor values and trigger conditions.

II. Motivation

While providing real-time responses is important for many IoT services such as health monitoring, disaster response, and home automation, existing IoT frameworks [1, 2, 7, 8] do not effectively and efficiently support real-time execution of the services. First, the frameworks do not provide real-time interfaces for users, so users cannot specify real-time constraints for their applets. There exist some real-time interfaces like the IFTTT real-time APIs [1], but the interfaces are about forcing instance notification of sensing value changes, not describing real-time constraints for their applets. As a result, the frameworks schedule their applets without considering their deadlines. To fully support real-time execution of the applets, the IoT frameworks should provide an interface for users to define their own real-time constraints for each applet.



(a) Temperature

(b) Humidity

(c) Illuminance

Fig. 2. Sensor data from indoor and outdoor thermometers, hydrometers and lightmeters. Green and blue lines indicate indoor and outdoor sensing values respectively. Each sensor measures the environments every second for 5 days. April 17th is rainy.

Moreover, the frameworks [1, 2, 7, 8] collect sensor data without reflecting the current sensor values and trigger conditions of the applets. To support instant responses for the applets, the frameworks can request the connected sensors to notify their changes using the real-time APIs [1] and publish-and-subscribe messaging protocols [9, 10]. However, since the sensors do not know the trigger conditions of applets, the sensors keep sending the sensing values whenever changed although the changed values do not affect trigger conditions. For example, during the night time, outdoor thermometers keep sending their temperature changes like Fig. 2(a), but the temperatures do not affect any trigger condition of the applets in Fig. 1. To efficiently collect sensor data without unnecessary data communication, either the sensors should send their data that affect trigger conditions of the applets, or the IoT frameworks should poll the sensors reflecting the current sensor values and the trigger conditions. Since the IoT sensors are generally too lightweight to manage various trigger conditions, this work focuses on the efficient polling scheduling of IoT frameworks. Moreover, since capturing changes of discrete sensor values is relatively simple through notification, this work focuses on continuous sensor values like temperature, humidity, and illuminance.

Recent works [11–13] have proposed data acquisition scheduling algorithms from normally-off sensors to support real-time decision making in the IoT environments. To make a decision with valid sensing data, the proposed algorithms analyze structures of trigger conditions, and determine the

applet ::= IF conditions
THEN actions
WITH constraints
conditions ::= (conditions && conditions)
::= (conditions || conditions)
::= condition
condition ::= $S_i \bigtriangleup X_i$
$\bigtriangleup$ ::= $<, \leq, >, \geq$
actions ::= actions; action
::= action
action ::= A_i.operation()
constraints ::= Time $<$ D
::= Time $<$ D && MissRatio $<$ e

Fig. 3. RT-IFTTT Syntax. Here, X_i, D and e are constants

App_vent ::= IF $T > 25$ && $L >$ 25,000
THEN Window.open(); blind.open()
WITH Time $<$ 300 && MissRatio $<$ 0.05
App_blind ::= IF $L >$ 50,000
THEN blind.close()
WITH Time $<$ 120 && MissRatio $<$ 0.05
App_day ::= IF $L >$ 20,000
THEN bulb.off()
WITH Time $<$ 90
App_night ::= IF $L <$ 1,500
THEN bulb.on()
WITH Time $<$ 90 && MissRatio $<$ 0.01
App_rainy ::= IF $H > 75$ && $L <$ 1,000
THEN Window.close()
WITH Time $<$ 60 && MissRatio $<$ 0.01

Fig. 4. RT-IFTTT Applet Examples of the Smart Home Service in Fig. 1

order of sensor polling. Here, the algorithms assume that the freshness periods of each sensor data are given as a fixed value. However, as Fig. 2 illustrates, sensing values from different sensors have different characteristics even though they are the same type of sensors. For example, indoor sensors generate more stable temperature, humidity, and illuminance data than outdoor sensors, so indoor sensors can have longer freshness period. Moreover, the sensing values have repeated patterns, so predicting the range of sensor values after minutes or tens of minutes is possible. If the current sensing value is different enough from the trigger conditions, an IoT framework can increase the freshness period of the data. To maximize sensor polling intervals and minimize battery power consumption of sensors, this work dynamically calculates the freshness period of each sensor reflecting the sensing value history, the current sensing value, and its related trigger conditions.

III. Design of RT-IFTTT Language and Framework

To fully support real-time execution of applets while minimizing battery power consumption of sensors with an appropriate sensor polling interval, this work proposes RT-IFTTT that consists of the RT-IFTTT language and its framework.

A. RT-IFTTT Language

The RT-IFTTT language extends the existing IFTTT rules, and allows users to describe their applets with real-time constraints. Fig. 3 illustrates the overall syntax of the RT-IFTTT language. The RT-IFTTT syntax largely consists of three parts such as trigger `conditions`, `actions` and real-time `constraints`. Like the IFTTT syntax, the trigger `conditions` lead the `actions` in an applet. The `conditions` can recursively consist of && or || combinations of multiple `conditions`. Each `condition` consists of a sensing value, a target value and a relational operator. Here, since this paper focuses on finding an efficient polling intervals of sensors that measure continuous values, to simplify the interval finding problem, the RT-IFTTT syntax only allows integers or real numbers for the sensing and target values. Like the `conditions`, `actions` can consist of a series of `actions`, and each `action` describes an operation of an actuator.

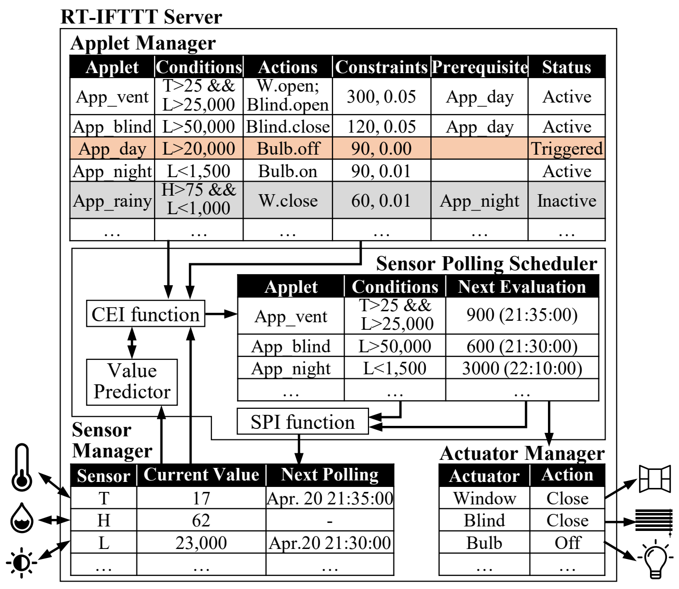
Unlike the IFTTT syntax, RT-IFTTT has real-time `constraints` in its syntax. The `constraints` describe the deadline of the applet as `D` and its maximum deadline miss ratio as `e`. If an applet should always meet its deadline, users can describe the `constraints` without its miss ratio. If an applet has its maximum miss ratio as `e`, the RT-IFTTT framework can aggressively defer its sensor polling time, saving battery power of connected sensors.

Fig. 4 shows example applets written in the RT-IFTTT language for the Smart Home service in Fig. 1. In the example, there exist three sensors such as a thermometer, a hydrometer and a light meter, and three actuators such as a window, a blind and a bulb. The 5 example applets execute 5 home automation services such as ventilating a room, closing a blind, turning on and off a bulb and closing a window if the sensing values satisfy their trigger conditions. For example, if its temperature and illuminance values are higher than 25 and 25,000, the `App_vent` applet ventilates the room by opening its window and blind within 300 seconds. The `App_day` applet defines its constraints without a miss ratio, indicating a firm deadline. The RT-IFTTT compiler analyzes the applets and extracts their trigger conditions, actions and real-time constraints like the first 4 columns of the applet manager table in Fig. 5.

B. RT-IFTTT Framework

The RT-IFTTT framework executes given RT-IFTTT applets respecting their real-time constraints. Fig. 5 illustrates the overall structure of the RT-IFTTT framework. The framework consists of an applet manager, a sensor polling scheduler, a sensor manager and an actuator manager.

Applet manager analyzes prerequisite relations among applets and manages current states of applets. If an applet `A` always triggers its action before another applet `B`, `A` is a prerequisite of `B`. Since the applet cannot be triggered before its prerequisites, if at least one of the prerequisites is not yet triggered, the RT-IFTTT framework can exclude the applet from the scheduling candidates, and reduce the scheduling overheads. For the example of Fig. 5, `App_night` is a prerequisite of `App_rainy` because of their comparison operators and target values for `L`. Since `App_night` is not yet triggered, the applet manager classifies `App_rainy` as an inactive applet, and excludes `App_rainy` from the



Applet	Conditions	Actions	Constraints	Prerequisite	Status
App_vent	T>25 && L>25,000	W.open; Blind.open	300, 0.05	App_day	Active
App_blind	L>50,000	Blind.close	120, 0.05	App_day	Active
App_day	L>20,000	Bulb.off	90, 0.00		Triggered
App_night	L<1,500	Bulb.on	90, 0.01		Active
App_rainy	H>75 && L<1,000	W.close	60, 0.01	App_night	Inactive
…	…	…	…	…	…

Applet	Conditions	Next Evaluation
App_vent	T>25 && L>25,000	900 (21:35:00)
App_blind	L>50,000	600 (21:30:00)
App_night	L<1,500	3000 (22:10:00)
…	…	…

Sensor	Current Value	Next Polling
T	17	Apr. 20 21:35:00
H	62	-
L	23,000	Apr.20 21:30:00
…	…	…

Actuator	Action
Window	Close
Blind	Close
Bulb	Off
…	…

Fig. 5. Overall Structure of the RT-IFTTT Framework

scheduling candidates. Here, `App_vent` and `App_blind` are active because their prerequisite, `App_day` is already triggered.

Sensor polling scheduler dynamically calculates evaluation and polling intervals for each trigger condition and their sensors. The sensor polling scheduler consists of a sensor value predictor, a `CEI` (Condition Evaluation Interval) function, and a `SPI` (Sensor Polling Interval) function. The sensor value predictor predicts the range of sensor values after a given time period with previous sensing data, and provides the probability of value gradients over time gradients ($p_i(\Delta s|\Delta t)$). Interacting with the sensor value predictor, the `CEI` function calculates evaluation intervals of each trigger condition. From the evaluation intervals of trigger conditions, the `SPI` function calculates polling intervals of each sensor. Whenever the sensor manager updates a sensing value, the sensor polling scheduler evaluates all trigger conditions that use the sensing value, and also invokes the `CEI` function to update the evaluation intervals of trigger conditions. Then, the sensor polling scheduler invokes the `SPI` function to update polling intervals of all the sensors that are used in the trigger conditions. Section IV will describe details about how the sensor polling scheduler achieves efficient evaluation and polling intervals for trigger conditions and sensors.

Sensor manager manages connectivity of sensors and their sensing data. As Hasenfratz et al. [14] mentioned, the sensor manager acquires sensing data with pulling. At the scheduled polling time, the sensor manager requests sensing data from its corresponding sensor and receives the data. The sensor manager also keep the history of sensing data for the sensor polling scheduler to predict sensing values.

Actuator manager manages connectivity of actuators and sends their operations to actuators if an applet satisfies its trigger conditions.

C. Sensor Value Prediction

Based on an observation result that the sensing values have repeated patterns, this work designs a maximum normalized sensor value gradient model (MNSVG model) that calculates a probability of a maximum normalized sensor value gradient after a given time gradient ($p_i(\Delta s|\Delta t)$) using the history of sensing data. In the current sensor polling scheduler implementation, the `CEI` function uses the MNSVG model to analyze the possible range of a sensor value after a given time period and to calculate evaluation intervals of trigger conditions. Here, since the `CEI` function uses the prediction model as a black box, the RT-IFTTT framework can use other sensor value prediction models [15–18] instead of the MNSVG model.

The sensor value predictor generates a MNSVG model for a sensor, s_i, in three steps. First, the predictor calculates maximum normalized sensor value gradients over time gradients at a certain time point, t_0, by

$$\Delta s_i(\Delta t = t - t_0) = max_{t_0 \le \tau \le t} \frac{|s_i(\tau) - s_i(t_0)|}{|s_i(t_0)|}$$

Second, the predictor generates a multiset, $S_{i,\Delta t}$, that represents a distribution of maximum normalized sensor value gradients over different time points from t_s to t_f for a certain time gradient, Δt.

$$S_{i,\Delta t} = \{\Delta s_i(\Delta t_j = (t_j + \Delta t) - t_j) | t_s \le t_j \le t_f\}$$

Finally, the predictor generates a probability function about a maximum normalized sensor value gradient after a given time gradient, $p_i(\Delta s|\Delta t)$.

$$p_{s_i}(\Delta s|\Delta t) = \frac{|\{s | s = \Delta s, s \in S_{i,\Delta t}\}|}{|S_{i,\Delta t}|}$$

The probability function will be used for the `CEI` function to analyze the possible ranges of a sensor value over different time gradients.

IV. Problem Statement and Solution

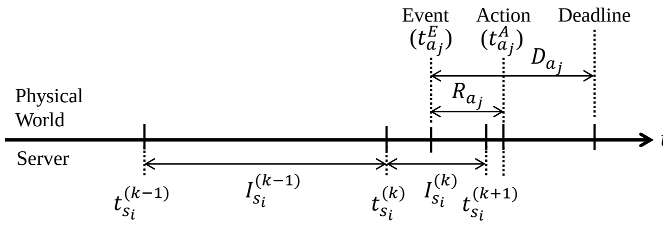
The sensor polling scheduler in the RT-IFTTT framework dynamically calculates evaluation and polling intervals for each trigger condition and their sensors. This section formally defines a problem about the goal of the sensor polling scheduler (Section IV-A), and provides a solution with a proof (Section IV-B).

A. Problem Statement

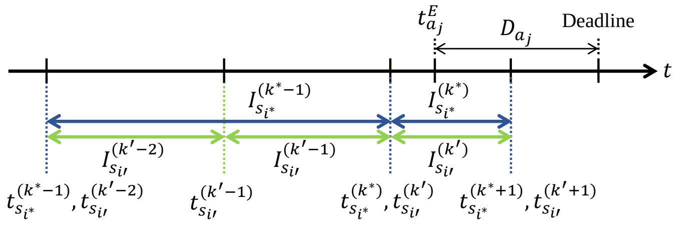
To support real-time execution of an applet, the RT-IFTTT framework should execute various tasks such as polling sensors, evaluating trigger conditions and taking actions before its deadline. Figure 6(a) illustrates the tasks for a single-sensor applet at a timeline. For an applet a_j with a relative deadline D_{a_j}, the framework polls the sensor and updates its value s_i at time points t_{s_i} with flexible intervals I_{s_i}. Then, the framework evaluates trigger conditions and takes actions at time point $t^A_{a_j}$ if an event occurs at $t^E_{a_j}$. Table I summarizes the notations used in this section.

TABLE I
NOTATION FOR THE PROBLEM

Notation	Description	Notation	Description
N	The total number of sensors	M	The total number of active applets
D_{a_j}	Relative deadline of applet a_j	R_{a_j}	Response time of applet a_j
I_{s_i}	Polling interval of sensor s_i	I_{a_j}	Evaluation interval of applet a_j
$t^E_{a_j}$	Time point when the event of a_j occurs	$t^A_{a_j}$	Time point when the action for a_j is taken
t_{s_i}	Time point when s_i is polled	$p_{s_i}(\Delta s\|\Delta t)$	Probability of value gradients over time gradients for s_i



(a) Applet with Single Sensor



(b) Applet with Mutiple Sensors

Fig. 6. Timeline of an Event Occurrence and its Detection

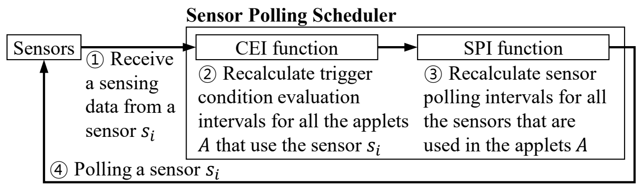

Sensor Polling Scheduler
Sensors
① Receive a sensing data from a sensor s_i
CEI function
② Recalculate trigger condition evaluation intervals for all the applets A that use the sensor s_i
SPI function
③ Recalculate sensor polling intervals for all the sensors that are used in the applets A
④ Polling a sensor s_i

Fig. 7. Brief Process of Sensor Polling Scheduler

For the framework to meet the deadline of the applet, its response time R_{a_j} should be shorter than the deadline D_{a_j}.

$$R_{a_j} = t^A_{a_j} - t^E_{a_j} \le D_{a_j}$$

However, the framework hardly know the exact time point of the event occurrence $t^E_{a_j}$. Since the event did not occur at the previous polling, the framework conservatively considers the previous polling time point $t^{(k)}_{s_i}$ as the event occurrence time point $t^E_{a_j}$, and calculate the conservative response time $R^C_{a_j}$ by

$$R_{a_j} \le R^C_{a_j} = t^A_{a_j} - t^{(k)}_{s_i} \le D_{a_j}$$

To simplify the problem and focus on finding the efficient sensor polling intervals, this work assumes that the time required to complete the action after sensor polling ($t^A_{a_j} - t^{(k+1)}_{s_i}$) is fixed and already reflected into the deadline. Therefore, to support real-time execution, the framework should keep the kth polling interval $I^{(k)}_{s_i}$ less than the deadline D_{a_j}, while maximizing the other polling intervals to reduce battery power consumption of the sensor.

$$R^C_{a_j} \approx I^{(k)}_{s_i} = t^{(k+1)}_{s_i} - t^{(k)}_{s_i} \le D_{a_j}$$

Since the RT-IFTTT language supports trigger conditions with multiple sensors, an applet a_j can use multiple sensing values such as s_{i^*} and $s_{i'}$. Moreover, a sensor can be used in multiple applets. Figure 6(b) illustrates the tasks of a multi-sensor applet at a timeline. To correctly find the occurrence of an event, the framework should evaluate all the conditions while all the sensing data are fresh, so the framework polls all the sensors in an applet condition together. However, a sensor $s_{i'}$ may be used in another applet, and polled at a time point $t^{(k'-1)}_{s_{i'}}$ before the scheduled one $t^{(k')}_{s_{i'}}$. Then, the framework recalculates the next polling point $t^{(k^*)}_{s_{i^*}}, t^{(k')}_{s_{i'}}$ of sensors $s_{i^*}, s_{i'}$ in the applet with the updated sensor value.

Then, this work formulates a problem as follows.

Problem Statement. The problem is to find maximum polling intervals $I_{s_i} (i = 1, ..., N)$ for each sensor value s_i that satisfy the following condition.

Condition. For every applet a_j ($j = 1, ..., M$), there exists a sensor polling of a senor value s_{i^*} that is used in a_j and satisfies

$$I^{(k)}_{s_{i^*}} = t^{(k+1)}_{s_{i^*}} - t^{(k)}_{s_{i^*}} \le D_{a_j}$$

where $t^{(k)}_{s_{i^*}} < t^E_{a_j} < t^{(k+1)}_{s_{i^*}}$.

B. *Algorithms*

The basic principle of the algorithms in the sensor polling scheduler is increasing sensor polling intervals if an event will not occur with high possibility, and decreasing the intervals when the event seems to occur soon. In other words, the algorithms reflect the probabilities of trigger conditions in an applet into the sensor polling intervals. For the example applets in Fig. 4, if the current temperature is around 0°C, the algorithms do not poll the temperature data every several minutes because the probability that the temperature becomes higher than 25°C in an hour is very low.

In the algorithms, the next polling time of a sensor value t_{s_i} is determined by

$$t_{s_i} \leftarrow t_{now} + min(I^m_{s_i}, I^a_{s_i})$$

where $I^m_{s_i}$ is the next sampling interval required to keep the sensor value prediction model valid, and $I^a_{s_i}$ is the next

Algorithm 1: Sensor Polling Scheduler

```
Input: $A = \{a_1, a_2, ...a_M\}$: Set of active applets
       $S = \{s_1, s_2, ...s_N\}$: Set of sensor values
       $s_i$ : Updated sensor value
Output: $T_S = \{t_{s_1}, t_{s_2}, ..., t_{s_N}\}$: Set of next polling time
1  $I \leftarrow \phi$ ;
2  foreach $a_j \in A$ do
3      if $s_i \in a_j.Conditions$ then
4          $I_{a_j} \leftarrow$ CEI ($a_j.Conditions$, $a_j.D$, $a_j.e$);
5          $I \leftarrow I \cup \{I_{a_j}\}$;
6      end
7  end
8  foreach $s_i \in S$ do
9      $I^a_{s_i} \leftarrow$ SPI ($s_i$, $A$, $I$);
10     if $I^a_{s_i} == Int_{max}$ then
11         continue;
12     end
13     $t_{s_i} \leftarrow t_{now} + \min(I^m_{s_i}, I^a_{s_i})$;
14 end
```

evaluation interval of an applet that uses the sensor value s_i.

Each sensor value s_i has its maximal sampling interval $I^m_{s_i}$ to build its sensor value prediction model. This work assume that a fixed value is given for $I^m_{s_i}$, and the value is much larger than relative deadlines of the applets. That is,

$$D_j \ll I^m_{s_i}$$

for every applet $a_j (j = 1, .., M)$.

Algorithm 1 describes the overall algorithm of the sensor polling scheduler that finds the efficient sensor polling intervals. Whenever a sensor value s_i is newly updated, the RT-IFTTT framework invokes the algorithm because the updated value may affect the probabilities of applet conditions and their intervals. The algorithm largely consists of two steps. First, the algorithm checks if an applet a_j uses the updated sensor value s_i. If so, the algorithm calls the CEI function that calculates the evaluation interval of the applet a_j reflecting the updated value (Algorithm 2). Second, the algorithm calls the SPI function that reflects the updated evaluation intervals to all the sensors (Algorithm 3). If the sensor is not used in any updated applet, the algorithm does not update its next polling schedule. Fig. 7 briefly illustrates the cyclic process of the sensor polling scheduler.

Since a trigger condition of an applet can be a combination of sub-conditions with logical operators such as && and ||, the algorithm should consider logical structures of each condition. To manage the trigger conditions in a systematic way, the RT-IFTTT framework introduces a new data structure, called condition tree. A condition tree C is a binary representation of a condition of an applet, defined as

$$C = \textbf{Internal}(C^{left}, lop, C^{right}) \\ \mid \textbf{Leaf}(s, cop, x)$$

where $lop \in \{\&\&, ||\}$ and $cop \in \{<, \leq, \geq, >\}$. For example,

Algorithm 2: CEI(C,D,e)

```
Input: $C$: A condition tree
       $D$: Relative deadline
       $e$: A miss ratio
Output: $I_C$: Evaluation interval for the condition $C$
1  if $C ==$ Leaf$(s_i, cop, x_i)$ then
2      Find a maximum $\Delta t^*$ that satisfies
       $p_{s_i}(\Delta s < \frac{|x_i - s_i|}{|s_i|} \mid \Delta t^*) \geq 1 - e$;
3      if $\Delta t^*$ is not found || $\Delta t^* < D$ then
4          $I_C \leftarrow D$;
5      else
6          $I_C \leftarrow \Delta t^*$;
7      end
8  else if $C ==$ Internal$(C^{left}, lop, C^{right})$ then
9      if $lop == \&\&$ then
10         if $eval(C^{left}) == true$ then
11             $I_C \leftarrow$ CEI ($C^{right}$,$D$,$e$);
12         else if $eval(C^{right}) == true$ then
13             $I_C \leftarrow$ CEI ($C^{left}$,$D$,$e$);
14         else
15             $I_C \leftarrow$ max (CEI ($C^{left}$,$D$,$e$),
16                                 CEI ($C^{right}$,$D$,$e$));
17         end
18     else if $lop == ||$ then
19         $I_C \leftarrow$ min (CEI ($C^{left}$,$D$,$1 - \sqrt{1-e}$),
20                              CEI ($C^{right}$,$D$,$1 - \sqrt{1-e}$));
21     end
22 end
23 return $I_C$;
```

a condition tree for App_vent in Figure 4 is defined as

$$C_{App\_vent} = \textbf{Internal}(\textbf{Leaf}(T, >, 25), \&\&, \\ \textbf{Leaf}(L, >, 25000)).$$

Algorithm 2 describes the CEI function that recursively traverses a condition tree of an applet, and calculates an appropriate interval for the condition tree. The condition C is either a leaf node or an internal node. If C is a leaf node, the condition C has only one sensor value s_i and its comparison target value x_i. If the difference between s_i and x_i is large enough, the CEI function can increase its evaluation interval until the predicted sensor value changes more than the difference. Thus, the CEI function finds the maximum evaluation interval that satisfies $p_{s_i}(\Delta s < \frac{|x_i - s_i|}{|s_i|} \mid \Delta t^*)$ larger than $1 - e$. Here, $p_{s_i}(\Delta s < \frac{|x_i - s_i|}{|s_i|} \mid \Delta t^*)$ means the probability that s_i does not reach x_i after a given time interval Δt^*. The probability should be larger than user-defined success ratio $(1 - e)$. If the CEI function cannot find any appropriate interval or the interval is smaller than the relative deadline D, the function just returns the deadline D, which is the worst-case interval.

If C is an internal node, CEI first recursively calculates the condition evaluation intervals of both subtrees (C^{left} and

C^{right}). Then, if two subtrees are linked by && operator, the CEI function returns the maximum interval between two intervals because the condition C will be true only when both subtrees are true. Here, if one of the sub-condition is already true, the CEI function uses the interval of the other sub-condition. If two sub-conditions are true , that is, C is $true$, then the applet is already triggered and the applet manager excludes this applet from the sensor polling scheduler. If two subtrees are linked with || operator, it takes the minimum interval between the two intervals that CEI calculates with an adjusted miss ratio ($1-\sqrt{1-e}$), because the condition C will be true if at least one of the subtrees is true.

Algorithm 3: SPI(s_i, A, I)

```
  Input: $s_i$: Target sensor
         $A = \{a_1, a_2, ...a_M\}$: Set of active applets
         $I = \{..., I_{a_j}, ...\}$: Set of updated evaluation
         intervals
  Output: $I^a_{s_i}$:The next polling interval of $s_i$
1 $I^a_{s_i} \leftarrow Int_{max}$;
2 foreach $I_{a_j} \in I$ do
3     if $s_i \in a_j.Conditions$ then
4         $I^a_{s_i} \leftarrow$ min ($I^a_{s_i}$, $I_{a_j}$);
5     end
6 end
7 return $I^a_{s_i}$;
```

Algorithm 3 describes the SPI function that finds the minimum polling interval of a sensor s_i among its relevant evaluation intervals. Since the evaluation intervals are the maximum intervals that each applet can meet its constraints, the polling intervals of the related sensors cannot exceed the evaluation intervals. Thus, the SPI function finds the minimum interval for a senor among the maximum evaluation intervals of its relevant applets. Here, if a sensor s_i is not used in any of applets with updated evaluation intervals, the SPI function sets the polling interval of s_i as Int_{max}, so the sensor polling scheduler leaves its next polling schedule unchanged.

Theorem 1 (Deadline Guarantee)**.** *Given a misspeculation ratio e, the algorithm finds $I_{s_i} (i = 1, ..., N)$ that meets the following condition.*

For every applet a_j ($j = 1, ..., M$), there exists a sensor polling of a sensor value s_{i^*} that is used in a_j and satisfies

$$p(I^{(k)}_{s_{i^*}} = t^{(k+1)}_{s_{i^*}} - t^{(k)}_{s_{i^*}} > D_{a_j}) \leq e$$

where $t^{(k)}_{s_{i^*}} < t^E_{a_j} < t^{(k+1)}_{s_{i^*}}$.

Proof. We will prove by induction that for any trigger condition C of an arbitrary applet a_j, there exists s_{i^*} that is used in C and satisfies

$$p(I^{(k)}_{s_{i^*}} = t^{(k+1)}_{s_{i^*}} - t^{(k)}_{s_{i^*}} > D_{a_j}) \leq e \qquad (1)$$

where $t^{(k)}_{s_{i^*}} < t^E_{a_j} < t^{(k+1)}_{s_{i^*}}$.
Let $|C|$ be the number of sub-conditions in C.
Base Case: When $|C|$ = 1, $C = \textbf{Leaf}(s_{i^*}, cop, x)$

- *Case $I^{(k)}_{s_{i^*}} < D_{a_j}$:*
  This case is not possible because of lines 3 and 4 in Algorithm 2.
- *Case $I^{(k)}_{s_{i^*}} = D_{a_j}$:*

$$p(I^{(k)}_{s_{i^*}} > D_{a_j}) = 0$$

- *Case $I^{(k)}_{s_{i^*}} > D_{a_j}$:*
  We will prove this by contradiction. Let's assume that

$$p(I^{(k)}_{s_{i^*}} = t^{(k+1)}_{s_{i^*}} - t^{(k)}_{s_{i^*}} > D_{a_j}) > e$$

In order for an event to occur within a time period $I^{(k)}_{s_{i^*}}$, Δs_{i^*} should be larger than $\frac{|x_{i^*} - s_{i^*}|}{|s_{i^*}|}$.

$$p_{s_{i^*}}(\Delta s_{i^*} \geq \frac{|x_{i^*} - s_{i^*}|}{|s_{i^*}|} \mid I^{(k)}_{s_{i^*}}) > e$$

However, from line 2 of Algorithm 2,

$$p_{s_{i^*}}(\Delta s_{i^*} < \frac{|x_{i^*} - s_{i^*}|}{|s_{i^*}|} \mid I^{(k)}_{s_{i^*}}) \geq 1 - e$$

$$p_{s_{i^*}}(\Delta s_{i^*} \geq \frac{|x_{i^*} - s_{i^*}|}{|s_{i^*}|} \mid I^{(k)}_{s_{i^*}}) < e$$

This is contradictory to the assumption. Therefore,

$$p(I^{(k)}_{s_{i^*}} = t^{(k+1)}_{s_{i^*}} - t^{(k)}_{s_{i^*}} > D_{a_j}) \leq e$$

Induction Case:
*Induction Hypothesis:*If the statement (1) is true for C with $|C| \leq n$, the statement (1) will be true for C which $|C|$ is $n+1$.
For $C' = \textbf{Internal}(C'^{left}, lop, C'^{right})$ which $|C'|$ is $n+1$, C'^{left} and C'^{right} satisfy the statement (1) because $|C'^{left}| \leq n$ and $|C'^{right}| \leq n$.

- *Case loq = &&:*

$$p^l(I^{(k_l)}_{s_l} = t^{(k_l+1)}_{s_l} - t^{(k_l)}_{s_l} > D_{a_j}) \leq e$$

$$p^r(I^{(k_r)}_{s_r} = t^{(k_r+1)}_{s_r} - t^{(k_r)}_{s_r} > D_{a_j}) \leq e$$

There exist four cases.

1) $eval(C'^{left}) = true \ \wedge \ eval(C'^{right}) = true$
Since C is $true$, this applet is already triggered and the applet manager excludes this applet from the sensor polling scheduler.
2) $eval(C'^{left}) = true \ \wedge \ eval(C'^{right}) \neq true$
From line 11 of Algorithm 2,

$$I_{C'} = I^{(k_r)}_{s_r} = t^{(k_r+1)}_{s_r} - t^{(k_r)}_{s_r}$$

$$p(I_{C'} > D_{a_j}) = p(I^{(k_r)}_{s_r} > D_{a_j})$$

Since $eval(C'^{left})$ can be $false$ at $t^{(k_r+1)}_{s_r}$, C' will less likely occur than C'^{right}.

$$p(I^{(k_r)}_{s_r} > D_{a_j}) \leq p^r(I^{(k_r)}_{s_r} > D_{a_j}) \leq e$$

Thus,

$$p(I_{C'} > D_{a_j}) \leq e$$

3) $eval(C'^{left}) \neq true \ \wedge \ eval(C'^{right}) = true$
We can complete this case using the same argument as we used for the case 2).
4) $eval(C'^{left}) \neq true \ \wedge \ eval(C'^{right}) \neq true$
From line 15 of Algorithm 2,

$$I_{C'} = max(I_{s_l}^{(k_l)}, I_{s_r}^{(k_r)})$$

Without loss of generality, assume $I_{s_l}^{(k_l)} \geq I_{s_r}^{(k_r)}$

$$I_{C'} = I_{s_l}^{(k_l)} = t_{s_l}^{(k_l+1)} - t_{s_l}^{(k_l)}$$

$$p(I_{C'} > D_{a_j}) = p(I_{s_l}^{(k_l)} > D_{a_j})$$

Since $eval(C'^{right})$ can be $false$ at $t_{s_l}^{(k_l+1)}$, C' will less likely occur than C'^{left}.

$$p(I_{s_l}^{(k_l)} > D_{a_j}) \leq p^l(I_{s_l}^{(k_l)} > D_{a_j}) \leq e$$

Thus,

$$p(I_{C'} > D_{a_j}) \leq e$$

– *Case $loq = ||$*:

$$p^l(I_{s_l}^{(k_l)} = t_{s_l}^{(k_l+1)} - t_{s_l}^{(k_l)} > D_{a_j}) \leq 1 - \sqrt{1-e}$$
$$p^r(I_{s_r}^{(k_r)} = t_{s_r}^{(k_r+1)} - t_{s_r}^{(k_r)} > D_{a_j}) \leq 1 - \sqrt{1-e}$$

From line 19 of Algorithm 2,

$$I_{C'} = min(I_{s_l}^{(k_l)}, I_{s_r}^{(k_r)})$$

Without loss of generality, assume $I_{s_l}^{(k_l)} \leq I_{s_r}^{(k_r)}$

$$I_{C'} = I_{s_l}^{(k_l)}$$

Since $I_{s_l}^{(k_l)} \leq I_{s_r}^{(k_r)}$,

$$p^r(I_{s_l}^{(k_l)} > D_{a_j}) \leq p^r(I_{s_r}^{(k_r)} > D_{a_j}) \leq 1 - \sqrt{1-e}$$

$$\begin{aligned}
p(I_{s_l}^{(k_l)} > D_{a_j}) = \ & p^l(I_{s_l}^{(k_l)} > D_{a_j}) + p^r(I_{s_l}^{(k_l)} > D_{a_j}) \\
& - p^l(I_{s_l}^{(k_l)} > D_{a_j}) * p^r(I_{s_l}^{(k_l)} > D_{a_j}) \\
= \ & 1 - (1 - p^l(I_{s_l}^{(k_l)} > D_{a_j})) * (1 - p^r(I_{s_l}^{(k_l)} > D_{a_j})) \\
\leq \ & 1 - (1 - p^l(I_{s_l}^{(k_l)} > D_{a_j})) * (1 - p^r(I_{s_r}^{(k_r)} > D_{a_j})) \\
\leq \ & 1 - \{1 - (1 - \sqrt{1-e})\} * \{1 - (1 - \sqrt{1-e})\} \\
= \ & e
\end{aligned}$$

Since the induction hypothesis holds for both logical operators, the statement (1) holds for C' which $|C'|$ is $n+1$.

Since both the base and the induction cases are true, by mathematical induction, the statement (1) holds for all the trigger conditions C of an arbitrary applet a_j.

□

V. Evaluation

A. Methodology

To show that RT-IFTTT effectively schedules data polling, this work simulates the RT-IFTTT system and compares its polling scheduling algorithm with the following three baseline algorithms in the evaluation.

- *Optimistic Fixed Interval (Fix-Opt)*: A server polls data from all the sensors every 15 minutes. The 15-minute interval is chosen with reference to the commodity IoT frameworks [1]. Since the algorithm sends requests for data sporadically, the algorithm is highly likely to miss an event with a short relative deadline.
- *Conservative Fixed Interval (Fix-Con)*: A server polls data from all the sensors every Δt_{con}, where Δt_{con} is the minimum relative deadline of all the applets. The algorithm rarely misses deadline, but requires lots of communication.
- *Random Interval (Random)*: A server polls data from all the sensors with random intervals. The intervals are randomly selected between the minimum relative deadline and 15 minutes.

The above three scheduling algorithms check all the applet conditions at every polling time and trigger appropriate actions if an event occurrence is detected. The major difference between our algorithm in the RT-IFTTT framework and the baseline algorithms is that our algorithm dynamically determines polling intervals considering the probabilities of trigger conditions.

To simulate the four algorithms with realistic data, this work installs 10 physical sensors that consist of 2 sets of temperature, humidity, UV Index, ambient light, and pressure sensors, and then collects data from the sensors for 10 days. This work uses the data in the simulation after removing noises of the collected data using a low-pass filter. To differentiate evaluation data from training data, this work divides the collected data into two parts. The RT-IFTTT framework generates the sensor value prediction model using one part of the data, and evaluates the polling scheduling algorithm using the other part of the data.

Using the value prediction model, the RT-IFTTT framework calculates probabilities of sensor value gradients over specific time gradient as described in Section III-C. Note that the performance of the value prediction model hugely affects the performance of our algorithm, and the RT-IFTTT framework can use other sensor value prediction models instead of our value prediction model.

B. Single-applet Results

Figure 8 compares the sensor polling timings of RT-IFTTT with three baseline algorithms (Fix-Opt, Fix-Con, and Random) for one applet. The Fix-Opt and Fix-Con algorithms periodically poll a sensor every 15 minutes and 30 seconds respectively. The random interval algorithm polls the sensor with random intervals. As a result, the Fix-Opt and random baseline algorithms miss the event deadline. The Fix-Con algorithm meets the deadline, but it polls the sensor frequently even when the sensor value is far from the threshold value of the event, causing more battery consumption of the sensor. Unlike the three baseline algorithms, RT-IFTTT dynamically changes its polling intervals reflecting the current sensor value and the threshold value. When the sensor value reaches the threshold value of the event, RT-IFTTT reduces the sensor

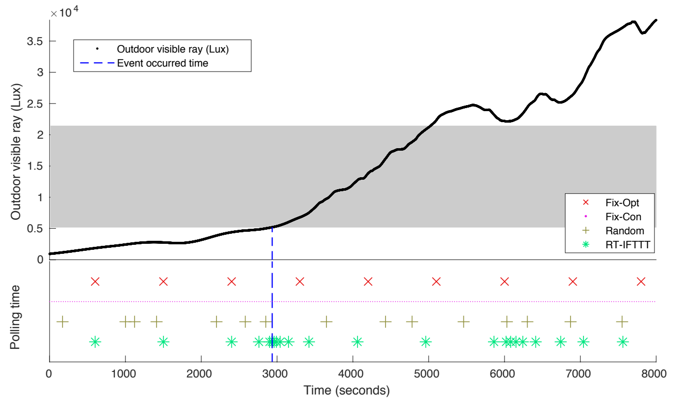


Fig. 8. Comparison of sensor pollings of different algorithms for one applet

polling intervals at near 3000 and 6000 seconds. If the sensor value is far from the threshold, RT-IFTTT increases the polling intervals. Here, at 6000 seconds, RT-IFTTT frequently polls the sensor because the sensor value reaches the threshold value although the event does not occur. Since RT-IFTTT dynamically changes the polling intervals according to the current sensor value and the threshold value, the RT-IFTTT framework can reduce the number of polling requests while keeping its deadline miss ratio small.

C. Multi-applet Results

This work compares the data scheduling algorithm of RT-IFTTT with three baseline algorithms (Fix-Opt, Fix-Con, and Random) described in Section V-A in terms of communication counts, miss ratio, and response time for multiple applets. The simulation uses randomly generated 100, 200, 300 and 400 applets with two sub-conditions based on the data set. The condition values in the applets are randomly selected within a certain range, considering the maximum and minimum values of each data set. For example, the value of the humidity sensor is absolutely between 0% and 100%, so the simulator sets comparison target values of the humidity sensor as any integer value between 0 and 100. Also, relative deadlines for each applet are chosen from a finite set $Ds = \{30, 60, 300, 600\}$. The simulation uses a constant miss ratio $e = 0.10$ for all the applets. Due to the randomness in generating applets, trigger conditions of some applets can be always true.

The simulator also calculates the ground truth values about the actual event occurrences. The ground truth includes the specific times at which events have occurred. To prevent from triggering an event multiple times in a short period due to the fluctuating sensor values, the simulator utilizes a threshold interval, which restricts re-trigger of events within the threshold interval. In other words, if the time gap between two events is too short, the simulator considers the two events as one event. With this true event history, this work evaluates the polling scheduling algorithm of RT-IFTTT and the three baseline algorithms.

1) Communication analysis: The communication count between sensors and a server is one of the crucial metrics in evaluating IoT frameworks because the communication count significantly affects the battery lifetime of the sensors. Figure 9(a) shows the communication counts of the four algorithms. Each communication count indicates the total number of data polling requests between a server and sensors. Since Fix-Opt and Fix-Con use a fixed polling period, their communication counts do not change over the different numbers of the applets. However, the data scheduling algorithm of RT-IFTTT utilizes flexible polling intervals. Since the intervals are influenced by the applet conditions, the algorithm shows different communication counts over the different numbers of the applets. Moreover, RT-IFTTT frequently polls sensors to meet the deadlines, the communication counts of RT-IFTTT are larger than the counts of the random polling. Our polling scheduling algorithm reduces the number of polling requests by 39.47% on average and up to 64.12% for 100 applets compared to the Fix-Con algorithm.

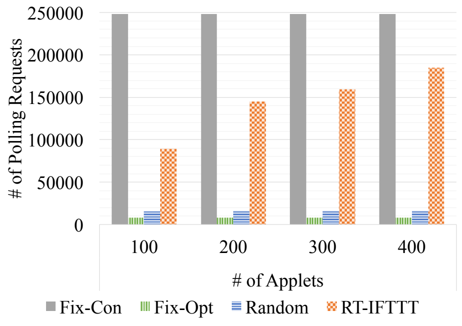
2) Miss ratio analysis: Miss ratio is defined as the number of correctly detected events over the total number of events and Figure 9(b) shows the result of evaluation on miss ratio. The Fix-Con algorithm obtains near 0 miss ratio. The algorithm sometimes misses an event occurrence because sensor data may slightly fluctuate near the decision boundary. As expected, the Fix-Opt algorithm hardly catches events for the applets of which relative deadlines are shorter than their polling periods. The random algorithm cannot also catch the events because it sets its polling intervals regardless of the relative deadlines. Therefore, the Fix-Opt and random algorithms show high miss ratios, which are around 65.6% and 55.5%.

The actual miss ratio of our data scheduling algorithm is 7.62% on average, outperforming the Fix-Opt and random algorithms by 8.96 times and 7.58 times respectively. Though the evaluation results fully satisfy the miss ratio condition, the current algorithm may not meet the predefined miss ratio if sensor values are dramatically changing. This is because the proposed probabilistic model for the value prediction described in Section III-C assumes that sensor values periodically and slowly change in a regular pattern. To predict irregularly changing sensor values, programmers can use a more powerful probabilistic model instead of the MNSVG model in the RT-IFTTT framework.

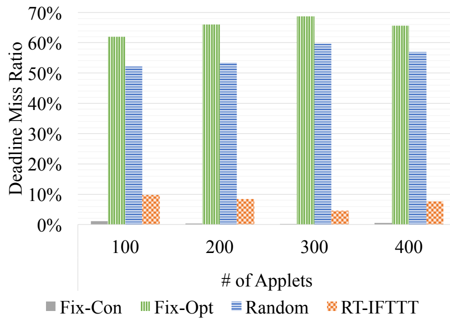
3) Response time analysis: This work calculates response times for each detected event by subtracting the event triggered time and the actual event occurrence time, and takes an average of them. Figure 9(c) shows the average response time of each algorithm. The results show that RT-IFTTT has 2.63 times faster response time than Fix-Opt, but 4.04 times slower response time than Fix-Con.

VI. Related Work

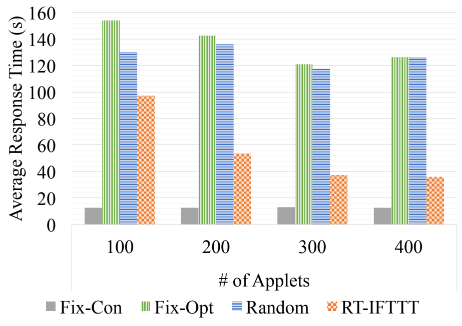
Real-time scheduling for IoT environments: Scheduling each task to meet its deadline has been a widely-discussed issue in real-time systems [11–13, 19–24]. Kang et al. [19] propose a real-time database model which manages deadline miss ratio and sensor data freshness, and Kang [20] improves the model to achieve lower power consumption. Chen [21]



(a) Communication Counts



(b) Deadline Miss Ratio



(c) Average Response Time

Fig. 9. Evaluation results for multiple applets. Here, target miss ratio of RT-IFTTT is 0.10.

considers sporadic tasks on real-time embedded systems, and explores task set synthesis scheduling while satisfying time constraints. Gopalakrishnan [22] suggests an optimal scheduler that determines the polling period by the existence of the previous event, and balances both data freshness and energy costs. Unlike the previous work, this work targets IoT frameworks where a server manages a number of applets with different event trigger conditions and checks the trigger conditions repetitively, and the characteristics of the system pose different challenges.

Recent works [11–13] have proposed data acquisition scheduling algorithms for normally-off sensors to support real-time decision making in the IoT environments. Kim et al. [12] propose an algorithm that finds the optimal order of acquisition of data items considering both decision validity (data freshness) and schedulability constraints. However, their work pays little attention to response time on actual event occurrence, while this work tries to reduce the response time by dynamically adjusting polling intervals.

Nath [24] introduces an acquisitional context engine (ACE), a middleware for energy-efficient acquisition of context information (attributes) from sensors. Based on the key observation that context attributes are correlated each other, ACE tries to reduce sensing cost by inferring one context attribute from other context attributes. This work focuses on avoiding acquisition of a requested value with inference caching and speculative sensing, not on scheduling when to acquire data.

Sensor data prediction: Sensor data prediction plays a crucial role in the RT-IFTTT framework to correctly calculate an appropriate sensor polling interval. There exist some studies [15–18] that use autoregressive (AR) model or its derivation. Especially Raza et al. [18] propose derivative-based prediction that samples least amount of data to predict. Wei et al. [25] combine grey model and Kalman Filter to ensure high prediction accuracy. Deshpande et al. [26] use an approximation model with correlated data and scheduled queries with the model to meet the deadline. The prediction models can be used in the RT-IFTTT framework instead of the MNSVG model because the `CEI` function uses the prediction model as a black box. Here, some work additionally needs a synchronization of a prediction model between a sensor and a server [17], or requires an assumption that a sensor pushes data and its prediction model [25].

Publish-and-subscribe or request-and-receive: Many IoT frameworks [2, 7, 8] assume a publish-and-subscribe message passing model [9, 10], but Hasenfratz et al. [14] point out that the publish-and-subscribe message passing model causes unnecessary energy consumption, and the request-and-receive model is more energy efficient. However, the request-and-receive model may increase the latency of data acquisition. While exploiting the energy efficient data acquisition strength, this work solves the latency problem by introducing trigger condition-aware flexible polling intervals. By predicting sensor values in the near future, this work dynamically increases/decreases sensor polling intervals when trigger conditions become unlikely/likely triggered.

VII. Conclusion

This work proposes RT-IFTTT, the first real-time IoT language and its framework that uses trigger condition-aware flexible sensor polling intervals. With the RT-IFTTT language, users can develop their applets with real-time constraints. Given applets, the RT-IFTTT framework finds an efficient sensor polling interval for each sensor, reflecting current sensor values, their related trigger conditions, and real-time constraints, with a sensor value prediction model, a condition evaluation interval function, and a sensor polling interval function. This work evaluates the RT-IFTTT framework with sensing data that 10 physical sensors collect for 10 days. With the collected data, the evaluation results show that the RT-IFTTT framework with the proposed scheduling algorithm executes 100 to 400 applets according to user-defined real-time constraints with up to 64.12% less sensor polling counts compared to the framework with the fixed intervals.

Acknowledgment

We thank the anonymous reviewers for their insightful comments and suggestions. This work is supported by Samsung Electronics, Samsung Research Funding Center under

Project Number SRFC-TB1403-04 and the Korean Government (MIST) under the ”ICT Consilience Creative Program” (IITP-2017-R0346-16-1007).

References

[1] “IFTTT,” https://ifttt.com.

[2] “Microsoft Flow,” https://flow.microsoft.com.

[3] M. M. Rathore, A. Ahmad, and A. Paul, “The internet of things based medical emergency management using hadoop ecosystem,” in *2015 IEEE SENSORS*, 2015.

[4] P. Castillejo, J. F. Martinez, J. Rodriguez-Molina, and A. Cuerva, “Integration of wearable devices in a wireless sensor network for an e-health application,” *IEEE Wireless Communications*, 2013.

[5] D. G. Simons-Morton, D. C. Goff, S. Osganian, R. J. Goldberg, J. M. Raczynski, J. R. Finnegan, J. Zapka, M. S. Eisenberg, M. A. Proschan, H. A. Feldman, J. R. Hedges, and R. V. Luepker, “Rapid early action for coronary treatment: Rationale, design, and baseline characteristics,” *Academic Emergency Medicine*, 1998.

[6] “Omnivex-emergency notification service,” https://www.omnivex.com/solutions/applications/emergency-notifications.

[7] “SmartThings,” https://www.smartthings.com.

[8] “AWS IoT - Amazon Web Services,” https://aws.amazon.com/iot.

[9] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-S–a publish/subscribe protocol for wireless sensor networks,” in *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops*, 2008.

[10] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” 2014.

[11] S. Hu, S. Yao, H. Jin, Y. Zhao, Y. Hu, X. Liu, N. Naghibolhosseini, S. Li, A. Kapoor, W. Dron, L. Su, A. Bar-Noy, P. Szekely, R. Govindan, R. Hobbs, and T. F. Abdelzaher, “Data acquisition for real-time decision-making under freshness constraints,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2015.

[12] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, and R. Hobbs, “Sporadic decision-centric data scheduling with normally-off sensors,” in *Proceedings of the IEEE Real-Time Systems Symposium*, 2016.

[13] J. E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, R. Hobbs, and W. Dron, “On maximizing quality of information for the internet of things: A real-time scheduling perspective (invited paper),” in *Proceedings of the IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016.

[14] D. Hasenfratz, A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele, “If you have time, save energy with pull,” in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, 2010.

[15] T. B. Matos, A. Brayner, and J. E. B. Maia, “Towards in-network data prediction in wireless sensor networks,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010.

[16] H. Jiang, S. Jin, and C. Wang, “Prediction or not? an energy-efficient framework for clustering-based data collection in wireless sensor networks,” *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[17] Y.-A. Le Borgne, S. Santini, and G. Bontempi, “Adaptive model selection for time series prediction in wireless sensor networks,” *Signal Process*, 2007.

[18] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, “Practical data prediction for real-world wireless sensor networks,” *IEEE Transactions on Knowledge and Data Engineering*, 2015.

[19] K. D. Kang, S. H. Son, and J. A. Stankovic, “Managing deadline miss ratio and sensor data freshness in real-time databases,” *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[20] K. D. Kang, “Reducing deadline misses and power consumption in real-time databases,” in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016.

[21] J. J. Chen, “Task set synthesis with cost minimization for sporadic real-time tasks,” in *2013 IEEE 34th Real-Time Systems Symposium*, 2013.

[22] S. Gopalakrishnan, “Optimal schedules for sensor network queries,” in *2010 31st IEEE Real-Time Systems Symposium*, 2010.

[23] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, “Towards optimal sleep scheduling in sensor networks for rare-event detection,” in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, 2005.

[24] S. Nath, “Ace: Exploiting correlation for energy-efficient and continuous context sensing,” in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys ’12. New York, NY, USA: ACM, 2012, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307640

[25] G. Wei, Y. Ling, B. Guo, B. Xiao, and A. V. Vasilakos, “Prediction-based data aggregation in wireless sensor networks: Combining grey model and kalman filter,” *Computer Communications*, 2011.

[26] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, “Model-driven data acquisition in sensor networks,” in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, 2004.