Real-Time Execution of Trigger-Action Connection for Home Internet-of-Things

Kai Dong[y][z], Yakun Zhang[x], Yuchen Zhao[x], Daoming Li[y], Zhen Ling[y] *, Wenjia Wu[y], and Xiaorui Zhu[{]
[y]School of Computer Science and Engineering, Southeast University, P.R. China
[x]School of Cyber Science and Engineering, Southeast University, P.R. China
[z]State Key Laboratory for Novel Software Technology, Nanjing University, P.R. China
Email:fdk, zyk, zyc, lidaoming0219, zhenling, wjwug@seu.edu.cn
[{] School of Information Engineering, Nanjing Xiaozhuang University, P.R. China
Email:xr_zhu@outlook.com

Abstract—IFTTT is a programming framework for Applets (i.e., user customized policies with a "trigger-action" syntax), and is the most popular Home Internet-of-Things (H-IoT) platform. The execution of an Applet prompted by a device operation suffers from a long delay, since IFTTT has to periodically reads the states of the device to determine whether the trigger is satisfied, with an interval of up to 5min for professionals and 60min for normal users. Although IFTTT sets up a flexible polling interval based on the past several times an Applet has run, the delay is still around 2min even for frequently executed Applets. This paper proposes a novel trigger notification mechanism "RTX-IFTTT" to implement real-time execution of Applets. The mechanism does not require any changes to the current IFTTT framework or the H-IoT devices, but only requires an H-IoT edge node (e.g., router) to identify the device events (e.g., turning on/off) and notify IFTTT to perform the action of an Applet when an identified event is the trigger of that Applet. The experimental results show that the averaged Applet execution delay for RTX-IFTTT is only about 2sec.

Index Terms—H-IoT, IFTTT, Applet, real-time execution

I. Introduction

IFTTT is a popular service integration platform which provides a convenient way to connect the Home Internet-of-Things (H-IoT) devices (e.g., Fitbit, Philips Hue) and web services (e.g., Gmail, Dropbox) [1]. A user can establish and customize Applets to create connections among devices and services by describing the triggers and actions, with the "IF this THEN that" syntax [2].

Each Applet suffers from a variable execution delay after the trigger event happens. The reason is that IFTTT uses a polling architecture to request a list of recent events. According to IFTTT documentation [3], the polling interval is up to 60min for normal users, and 5min for professionals. This delay also attracts the attention of the academia, e.g., [4] shows that the averaged delay is roughly 2min and can be up to 15min. However, there is no practical way to address the problem. On the one hand, an intuitive signaling architecture is impractical since it requires changes to the H-IoT devices. On the other hand, a polling architecture is born with a polling interval. It is supposed that IFTTT can never get rid of this delay, but only make some slight optimization to reduce it, e.g., by decreasing the polling interval at the cost of heavier traffic overhead.

We propose a novel trigger-notification mechanism named RTX-IFTTT which really gets rid of the polling interval to minimize the Applet execution delay. This mechanism offloads the task of monitoring the trigger events from the IFTTT server side to the edge node (e.g., a router). With RTX-IFTTT, the execution of an IFTTT Applet no longer relies on the polling architecture. Instead, the edge node is responsible for identifying the trigger events and notifying IFTTT of the events in real-time. It follows a two-step approach.

In the first step, the edge node should identify the trigger events with extremely high precision and recall rate. We propose a fine-grained event identification method based on traffic analysis. It has already been verified by existing researches that the traffic generated by an IoT device can be used to infer an IoT event [5][6][7][8][9][10][11]. However, RTX-IFTTT requires a much higher recall level. Suppose a trigger event, the identification (or inference) recall rate of which is 90%. It is really dangerous in an attack scenario, but is inadequate if an Applet can only be executed with this probability. In RTX-IFTTT, we divide a trigger event into fine-grained sub-events, and fingerprint sub-events to achieve nearly perfect identification precision and recall rate.

In the second step, the edge node must notify IFTTT of the trigger events. We propose a real-time Applet execution method based on two interfaces. The first one is a user interface named Check Now. The alternative interface is the Webhook, i.e., a callback interface. After the edge node identifies a trigger event, it either sends a "check now" request to the IFTTT, or makes an HTTP request to the URL configured for the Webhook. In either situation, IFTTT can be signaled to do something. Some additional tasks related to Applet processing is also performed by the edge node, to ensure the behavior of IFTTT conforms to the correct semantics of that Applet.

The advantage of RTX-IFTTT is three fold. Firstly and most importantly, it greatly reduces the Applet execution delay from roughly 2min to 2sec. Secondly, it enlarges IFTTT's ecosystem, since it is able to identify trigger events which are not supported by IFTTT. Lastly, it enables IoT connections across platforms/ecosystems which support Webhooks, e.g.,

* Corresponding author: Prof. Zhen Ling of Southeast University, China

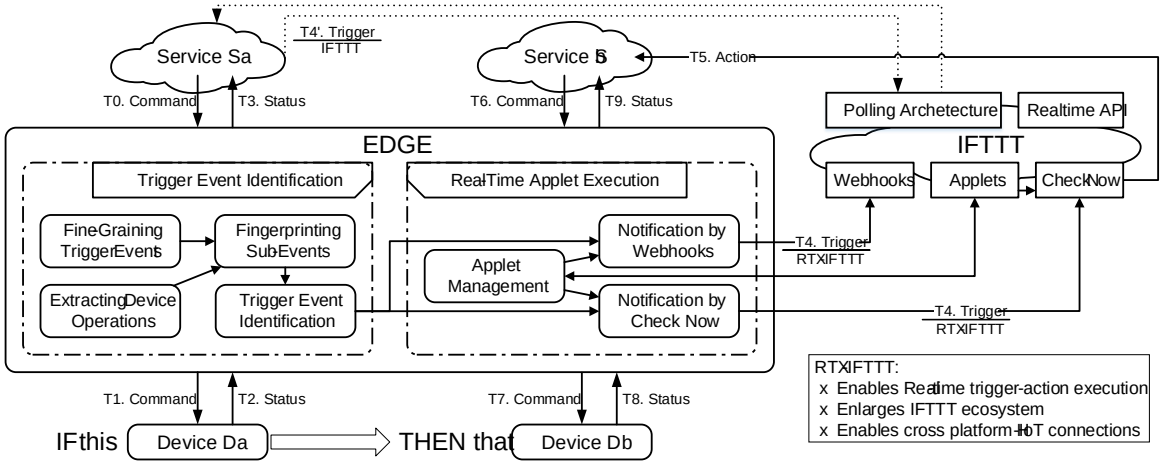


Fig. 1. RTX-IFTTT overview

IFTTT, SmartThings [12], HomeKit [13], Zapier [14], Home Assistant [15].

To summarize, this paper makes the following contributions:

- We propose an edge-based trigger noti cation mechanism named RTX-IFTTT to implement real-time execution of Applets. To the best of our knowledge, this is the rst mechanism which is able to reduce the Applet execution delay to seconds of time.
- We propose a ne-grained trigger event identi cation method. By ngerprinting sub-events instead of the whole trigger event, that event can be identi ed with nearly perfect precision and recall rate.
- We propose a real-time Applet execution method by employing either Check Now or Webhooks. With these interfaces, RTX-IFTTT does not require any changes to the IFTTT service or the H-IoT devices.
- Based on RTX-IFTTT, we introduce a new way to not only enlarge a single H-IoT ecosystem (IFTTT), but also connect devices and services across various ecosystems.

The rest of this paper is organized as follows. Sec. II describes the Applet execution delay in current IFTTT platform. Sec. III proposes a trigger event noti cation mechanism RTX-IFTTT and Sec. IV provides some detailed analysis. Sec. V evaluates RTX-IFTTT and Sec. VI gives a brief survey on related techniques. Sec. VII concludes the paper.

II. Problem

IFTTT enables "trigger-action" connections only between services. When a user connects his H-IoT device to the IFTTT ecosystem, what IFTTT actually communicates with is the vendor's service rather than the device itself. The mechanism behind the connection is the API endpoint, which is a Uniform Resource Identi er (URI) at the service's domain where IFTTT will GET updates (for triggers) or POST data (for actions).

By default, IFTTT uses a polling architecture to GET the updates. The polling interval is 60min for normal users and 5min for professionals [3], and the execution delay for each H-IoT Applet is various and ranges from 2min to 15min [4]. In recent years, IFTTT uses some really clever methods to reduce the delay by tuning the polling interval. However, the averaged delay is still roughly 2min (as detailed in Sec. V).

Along with the polling architecture, IFTTT also provides the Realtime API. This API has already been used by many web services (for triggers). An Applet involving such a trigger can be executed near-instantly.

Unfortunately, many services (especially H-IoT services) do not implement the Realtime API. We use Selenium [16], an automatic testing tool to crawl all the services and events including triggers and actions. By January 1st 2021, IFTTT's ecosystem consists of 681 services and over 2,600 events. Among them are 335 H-IoT services and 1,447 H-IoT trigger events. Most Applets prompted by H-IoT trigger events rely on the polling architecture instead of the Realtime API. One possible reason is that, if all H-IoT trigger services utilize this API, the incurred instantaneous workload may be too high [4], since IoT workload is known to be highly bursty [17].

III. Methodology

In this section, we propose a trigger-noti cation mechanism. We name it RTX-IFTTT, since it enables real-time execution of "IF-this-THEN-that" form of connection between H-IoT services/events, not only for IFTTT platform, but also for other popular platforms (as discussed later in Sec. IV-C).

A. Mechanism Overview

The idea behind RTX-IFTTT is to use a "signaling" architecture instead of the "polling" one, by of oading the task of monitoring triggers from IFTTT to the edge. The edge follows a two-step approach to implement real-time execution of Applets: it rst identi es a trigger event, then noti es IFTTT of that trigger to ensure real-time execution of the Applet. The trigger event identi cation is mainly based on traf c analysis and ngerprinting device events (status changes, e.g., turning on/off). The edge maintains features (ngerprints) of all device events. It monitors the transmitted packets, and identi es the device events and the corresponding triggers, and noti es IFTTT of the triggers. The real-time execution of an

2

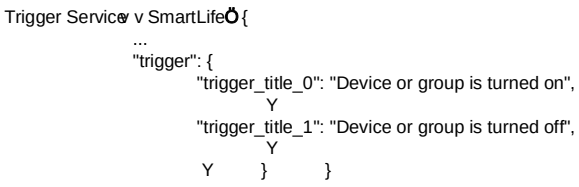


Fig. 2. An example trigger service of Smart Life

Applet is guaranteed by either requesting IFTTT to perform an immediate check on the target Applet with the Check Now interface, or by notifying the Webhook of a speci c connection constructed in advance (i.e., another Applet) which has the same action of the target Applet. In what follows, we detail implementation of these two steps.

B. Trigger Event Identi cation

RTX-IFTTT is able to automatically extract features of a trigger event and identi es that trigger. It has already been veri ed by existing researches that various features of traf c can be used by an adversary to infer an event of an H-IoT device [5][6][7][8][9][10][11]. The inference recall rate ranges from roughly 70% to 100% depending on various events, devices, noise handling technologies, and machine learning models generated in the training phase.

The main challenge for RTX-IFTTT deals with identi cation recall rate. Although the recall rate achieved by existing techniques is really dangerous for performing an inference attack, it is far from adequate for identifying a trigger event, since this rate determines the probability of successfully prompting the action of an Applet. Furthermore, the trigger event identi cation in RTX-IFTTT is deployed in large-scale and performed automatically, inevitably at the cost of precision and recall rate. To address this challenge, RTX-IFTTT divides a trigger event to sub-events, and identi es every sub-event to precisely identify the original trigger event. In what follows, we detail the work ow related to trigger event identi cation in RTX-IFTTT. Some analysis on our improvement on identi cation recall rate is provided in Sec. IV-A

1) Fine-Graining Trigger Events: In real H-IoT environments, the traf c generated with a same trigger event is heterogeneous. An H-IoT trigger event describes one speci c device status, however this status can be resulted from any one of many different operations (e.g., manual/APP/IFTTT operation). A device can be either remotely controlled by a service (e.g., user controls the device from an APP like SmartThings, or from an IoT platform like IFTTT), or locally controlled by a nearby user (e.g., user presses a button on the device or on the infra-red controller), to respond to different operations but result in a same status (i.e., a same event). Due to this reason, one trigger event corresponds to many different features in traf c generated with distinct operations.

For each operation of a same trigger event, usually two sub-events can be distinguished. Each sub-event corresponds to a hybrid of up-streaming and down-streaming traf c. The rst sub-event is the controlling command sent from the vendor's service to the H-IoT device. If an operation is remotely

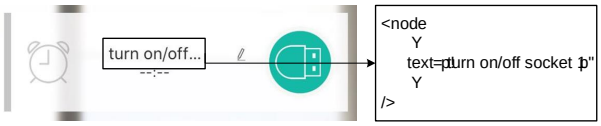


Fig. 3. A layout and corresponding XML le in Smart Life

controlled, the service will send a message about the operation to the device, and then the device will probably send some feedback. If an operation is locally controlled, there is no such traf c. The second sub-event is the status change sent from the device to the service. Whether remotely controlled or locally controlled, the device should de nitely respond to the operation and change its status, and report this change to the service. Then the service will con rm the status change. We rely on the router to identify the sub-events of a trigger, since all the traf c is forwarded by the router.

For most cases, we can obtain the features of a status change sub-event by performing a manual operation. After that, the features of the controlling command sub-events can also be obtained by performing other operations. When no manual operation is available, the features of the status change sub-event can also be obtained by performing different operations (i.e., different controlling command) which lead to a same device state (i.e., possibly same status change).

The identi cation recall rate is greatly improved by dividing a trigger event to sub-events. Some analysis is provided in Sec. IV-A, which is con rmed by our experiments in Sec. V-B.

2) Extracting Device Events: The events of an H-IoT device can be extracted from IFTTT Applets [7][18][19] and the UI of an APP for that device [20][21][22], by using Natural Language Processing (NLP) techniques.

For IFTTT Applet, every event (trigger or action) has a title eld to specify its functionality. Take a trigger service in Smart Life as an example (as shown in Fig. 2), the contents in the title eld of the rst trigger event is “Device or group is turned on”, where “Device” and “group” speci es the subject, and “is turned on” speci es the triggering condition. RTX-IFTTT uses Selenium [16] for crawling the description in title for IFTTT Applets, and uses NLTK [23] for parts-of-speech tagging and dependency relation parsing [24], and uses WordNet [25] for interlinking different expressions of a same operation, to nally extract device events supported by IFTTT.

For the UI of an APP, each device event correlates with a control in some layout. We use UI Automator [26] and Android Debug Bridge (ADB) [27] to obtain the UI hierarchy XML le, which contains the information of all the controls within a layout. An example layout and the corresponding XML le is as shown in Fig. 3. The device event can be identi ed by the String value in the text eld in the XML le.

3) Fingerprinting Sub-Events: There are three steps in ngerprinting sub-events, i.e., traf c collection, noise lter-ing and ngerprint generation. For traf c collection, RTX-IFTTT collects all routed traf c by using Tcpdump [28] and Wireshark [29]. For noise ltering, RTX-IFTTT lters the beacon packets, re-transmission packets, unrelated packets, and other noise packets. For ngerprint generation, RTX-

3

(a) Check Now (b) Webhook

Fig. 4. Interfaces used for noti cation.

IFTTT uses the MAC addresses to distinguish devices, and uses the packet lengths and the transmission directions to compute the ngerprint F of event as follows.

$$F = \arg\min_{s_i \in S} \frac{1}{|S|} \sum_{\forall s_j \in S} dist(s_i, s_j): \quad (1)$$

Where s_i represents the ith sequence of packets for event , S represents all the sequences collected for , $dist(s_i, s_j)$ represents the Levenshtein Distance [30] between s_i and s_j. With RTX-IFTTT, we have already constructed ngerprints for 27 kinds of H-IoT devices from 16 vendors. Part of ngerprints are listed in Table II, and all the devices are listed in Table III.

4) Identifying Trigger Events: RTX-IFTTT rst identi es sub-events, then determines whether the trigger event has happened. To identify a sub-event in real-time, RTX-IFTTT keeps monitoring the traf c by using the Scapy.Sniff library, and compares the traf c to all the ngerprints. If there exists one ngerprint that matches the traf c, then the corresponding sub-event with that ngerprint is identi ed. Based on identi cation of sub-events, RTX-IFTTT establishes an incremental and autonomous event identi cation method, which achieves near perfect precision and recall rate, as detailed in Sec. IV-A and Sec. IV-B. After the edge successfully identi es a trigger event, it then asks IFTTT to perform the action of the Applet.

C. Real-Time Applet Execution

It is non-trivial for RTX-IFTTT to ensure real-time and correct execution of an Applet. The router is unable to perform the action of that Applet by itself, unless it makes some change to IFTTT, or the H-IoT devices, or the vendors' services. To address this challenge, RTX-IFTTT introduces a novel method in which RTX-IFTTT noti es IFTTT of a trigger, and ensures IFTTT will respond to that trigger immediately. RTX-IFTTT relies on either of the two common interfaces, Check Now and Webhooks. Both interfaces are supported not only by IFTTT but also the majority of other H-IoT platforms.

1) Noti cation by Check Now: The rst method is to call the Check Now interface (as shown in Fig. 4(a)), so that IFTTT will check for the trigger by itself immediately. On calling the interface, RTX-IFTTT should address the concurrency problems originated from IFTTT. There is a race condition when IFTTT executes related Applets, especially when the Applets are prompted within a short period of time. IFTTT maintains the latest event it has seen for each trigger service. Each time it GETs updates from the service, the service returns a list of (up to 50) recent events. The action prompted by the rst trigger event is executed together with a cluster of subsequent actions. These actions are performed concurrently, therefore are in a race condition.

Sequence of Trigger Events (WeMo Plug #1)
on!off!on!off!on!off

+

IFTTT Applets
IF WeMo Plug #1 on THEN WeMo Plug #2 on
IF WeMo Plug #1 off THEN WeMo Plug #2 off

#

Sequence of Actions (WeMo Plug #2)	Final State	Frequency	
on!off!on!off!on!off	Correct	2/25	12/25
on!on!on!off!off!off		8/25	
off!on!off!on!on!off		1/25	
on!on!off!on!off!off		1/25	
off!off!off!on!on!on	Incorrect	10/25	13/25
off!on!off!on!off!on		2/25	
on!on!off!off!off!on		1/25	

Fig. 5. Multiple actions in a race condition

Suppose two related Applets, “If WeMo Plug #1 is activated (or deactivate), turn on (or off) WeMo Plug #2”. If WeMo Plug #1 is activated and then deactivated within a short period of time, the actions of WeMo Plug #2 are in a mess. We further suppose a trigger sequence “on!off!on!off!on!off” and perform it 25 times, to obtain the possible sequences of actions as illustrated in Fig. 5. Within all the 25 action sequences, only 2 sequences satis es the “on-off” consistency (i.e., each on/off action corresponds to one on/off trigger sequentially). Moreover, it is possible that WeMo Plug #1 is nally off and WeMo Plug #2 is nally on. We believe this deviates from the user's real intention behind the Applets. To make the situation even worse, IFTTT will never turn off WeMo Plug #2 (e.g., after checking the consistency of the nal states of WeMo Plug #1 and #2), unless the WeMo Plug #1 is turned on/off again. This is determined by the underlying implementation of the polling architecture of IFTTT. Within each polling, IFTTT is only noti ed of changes of data GET from the URI at the trigger service. If the data of the trigger service (of WeMo Plug #1) is not changed, IFTTT will not POST anything to the action service (of WeMo Plug #2).

In RTX-IFTTT, the edge is conscious of the trigger sequence, therefore it guarantees that the last action corresponds to the last trigger to ensure the correctness of the nal states of all H-IoT devices. If necessary, the edge is also able to guarantee that every action is prompted the correct number of times in correct order, by blocking a noti cation to IFTTT until the previous actions are performed.

2) Noti cation by Webhooks: A more general method is to rely on the Webhooks which are user customized HTTP callbacks (as shown in Fig. 4(b)). Most platforms including IFTTT provide this interface for users and developers. RTX-IFTTT speci es a Webhook in advance by con guring a URL for each possible action, and constructs a Webhook-action connection. Multiple Applets with a same action share a same Webhook. When a trigger of an Applet is identi ed, RTX-IFTTT determines which action to be performed, and makes an HTTP request to the URL con gured for the corresponding Webhook. Then IFTTT performs that action immediately. For IFTTT, a Webhook-action connection is constructed as

4

TABLE I
THE FINGERPRINTS OF A TRIGGER EVENT IS COMPOSED OF FINGERPRINTS OF SUB-EVENTS. CC INDICATES THE controlling command SUB-EVENT, AND SC INDICATES THE status change SUB-EVENT. THE RECALL RATE IS SHOWN IN THE TABLE, AND THE PRECISION RATE IS ALWAYS 100%.

Trigger Event	Operations	Fingerprints	Recall #1	CC Fingerprints	Recall #2	SC Fingerprints	Recall #3
WeMo Smart Plug switch on/off	Manual	322"33#	92.00%	/	/	322"33#	92.00%
	APP	351#33"774"33#	86.00%	351#33"	100.00%	774"33#	86.00%
	Timer/Count down	330#33"322"33#	100.00%	330#33"	100.00%	322"33#	100.00%
	IFTTT Applet	363#33"774"33#	90.00%	363#33"	100.00%	774"33#	90.00%

follows. A Webhook action connection is in essence an Applet with a special trigger service, i.e., Webhook. The trigger event is IFTTT "receives a web request", and a name to the event needs to be specified. Then the Maker server of IFTTT will automatically configure a web URL which is a regular expression: "https://maker.ifttt.com/trigger/{name}/with/key/{key}", where name is the name of the trigger event specified by RTX-IFTTT, and key is the secret key assigned to a user by IFTTT which can be obtained from the Maker server.

3) Applet Management: RTX-IFTTT must ensure the behavior of IFTTT conforms to the correct semantics of that Applet. For notification by Check Now, the router simply sends a request to IFTTT. For notification by Webhooks, the router establishes a new Webhook action connection in advance, where the action in the connection is the same action in the target Applet. When RTX-IFTTT notifies the Webhook, it also disables the original Applet in IFTTT to ensure that action is prompted only once.

D. Workflow of RTX-IFTTT

The router maintains fingerprints of all possible trigger events and sub-events, and monitors routed traffic as illustrated in Fig. 1. ⟨T0, T1⟩ and ⟨T6, T7⟩ indicate the controlling commands sent from the vendor's service to the H-IoT device, along with some optional feedback from the device to the service. ⟨T2, T3⟩ and ⟨T8, T9⟩ indicate the status changes sent from the device to the service, along with the acknowledgement from the service to the device. T4 indicates the traffic generated by the edge in RTX-IFTTT, which is in comparison with that generated in IFTTT (indicated by T4').

The workflow of RTX-IFTTT[1] is as follows. When a trigger event happens, the router identifies that trigger from traffic (T0 T3). Then the router notifies IFTTT of that trigger in real-time (T4). Therefore, IFTTT does not need to poll for that trigger (T4'). After being notified, IFTTT POSTs data to the action service (T5), to perform the action (T6 T9). The workflow of RTX-IFTTT is quite different from that of the vanilla IFTTT. The traffic marked as T4' (dotted arrows) is generated by IFTTT for polling the trigger service and by the service to notify IFTTT of that trigger. In contrast, RTX-IFTTT uses a signaling architecture implemented on the edge to replace the polling one.

IV. ANALYSIS

In this section, we provide some analysis on RTX-IFTTT. We provide the reason that fine-grained identification achieves higher recall rate in comparison with the traditional coarse-grained identification. In the meanwhile, we investigate the reason that real traffic generated with a trigger-event is different with its fingerprints. We also make some comparison between notification by Check Now and that by Webhooks. The prior is faster and tolerates identification errors, while the latter can be used to enable connections across platforms.

A. Identifying Fine-Grained Sub-Events

Existing inference techniques suffers from an inadequate recall rate, when applying to trigger identification in real H-IoT environments. This is because a same trigger event can be the result of different operations, while each operation can be divided into sub-events (controlling command and status change), and each sub-event can generate different traffic patterns. Even if the traffic of a same trigger event is collected thousands of times, no one can guarantee a perfect recall rate. Table I illustrates the recall rate in identifying an example trigger event " WeMo Smart Plug switch on/off". The recall rate (Recall #1) is inadequate since there are too many (potential) fingerprints for this trigger event.

By dividing a trigger event to sub-events, we obtain the following findings. The recall rate (Recall #2) for identifying the controlling command sub-event is always 100%, however the recall rate (Recall #3) for identifying the status change sub-event is often inadequate. If a controlling command sub-event is identified, while the corresponding status change sub-event is not, then the trigger probably happens. RTX-IFTTT decides whether the trigger event has happened as follows. It supposes this trigger happens, and notifies IFTTT of this trigger by using the Check Now interface. If the action is prompted by IFTTT, then this trigger has really happened.

The fine-grained sub-event identification performance is provided in Table II. Take WeMo Smart Plug (the 1^{st} device) as an example. If it is operated by an IFTTT Applet (the 4^{th} operation of the device), the recall rate for identifying the controlling command sub-event is 100% and that for identifying status change is 90%. This implies that, with the i.i.d. assumption, the traditional coarse-grained identification achieves a recall rate of 100% 90% = 90%, while RTX-IFTTT can in theory achieve a recall rate of (1 100%) (1 90%) = 100%. This is confirmed by our experiments where the recall rate for identifying this trigger event is perfect.

B. Identifying Trigger Events in Real H-IoT Environments

Although one can identify a trigger event based on traffic analysis in a laboratory environment, it is still challenging to achieve adequate precision rate and recall rate in the real H-IoT environments. This is because the real traffic generated

[1]A demo is available at https://github.com/nis-seu/RTX-IFTTT-demo

5

TABLE II
FINGERPRINTS AND IDENTIFICATION FOR TRIGGER EVENTS OF 5 SELECTED DEVICES

(Vendor) Device Trigger Events	Operations	Sub-Events	Fingerprints	Sub-Event Identification			Trigger Event Identification		
				Precision	Recall	F1 Score	Precision	Recall	F1 Score
WeMo Smart Plug Switch on/off	Manual	SC	322",33#	100.00%	92.00%	95.83%	100.00%	92.00%	95.83%
	APP	CC	351#,33"	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	774" 33#	100.00%	86.00%	92.47%			
	Timer/ Countdown	CC	330#,33"	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	322",33#	100.00%	100.00%	100.00%			
	IFTTT Applet	CC	363#,33"	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	774" 33#	100.00%	90.00%	94.74%			
MiJia Smart Switch 2 Switch on/off	Manual	SC	169"185"89#89#	100.00%	81.00%	89.50%	100.00%	81.00%	89.50%
	APP	CC	169#169"	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	185"137"89#89#	100.00%	87.00%	93.05%			
	Timer/ Countdown	CC	217#105"	98.52%	100.00%	99.25%	98.52%	99.50%	99.01%
		SC	169"185"89#89#	100.00%	68.50%	81.31%			
Smart Life Smart Strips Switch on/off	Manual	SC	255"4#	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
	APP	CC	188#	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	255"4#	100.00%	100.00%	100.00%			
	Timer/ IFTTT Applet	CC	296#	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
		SC	255"4#	100.00%	100.00%	100.00%			
SmartThings Switch Switch on/off	Manual	SC	433"47# 434"47# 435"47#	100.00%	96.00%	97.96%	100.00%	96.00%	97.96%
	APP/ Timer/ Countdown/ IFTTT Applet	CC	127#47" 128#47" 255#47" 256#47" 257#47"	98.46%	96.00%	97.21%	98.52%	99.75%	99.13%
		SC	433"47# 434"47# 435"47#	100.00%	93.50%	96.64%			
Yeelight LED Bulb 1 Switch on/off	APP	CC	121#89"	99.01%	100.00%	99.50%	99.01%	100.00%	99.50%
		SC	121"89#	100.00%	99.00%	99.50%			
	Timer	CC	153#89"	100.00%	100.00%	100.00%	98.04%	100.00%	99.01%
		SC	121"89#	100.00%	97.00%	98.48%			
	IFTTT Applet	CC	105#89"	100.00%	96.00%	97.96%	99.01%	100.00%	99.50%
		SC	121"89#	100.00%	100.00%	100.00%			

between a device and the vendor service can be changeable and is not always ideal.

We conduct a small experiment (as illustrated in Fig. 6) and dive into the details of the traffic a little bit, to obtain some insight into the reason why fingerprinting events perform poorly in real H-IoT environments. We only focus on two operations of a same device, i.e., switch on/off WeMo Smart Plug manually or via APP, and we suppose we have obtained the fingerprints of this trigger event (and the corresponding three sub-events including status change for manual operation and controlling command and status change for APP operation). In the experiment, we turn on/off the plug via APP and then within 1 second turn off/on the plug manually. We record the traffic, reduce the noise, and try to identify the events/sub-events. The process is repeated 100 times. It is quite interesting that the ideal traffic for identifying the trigger event is observed only 8 times. This implies that traffic generated with concurrent events of a same device is mixed up.

We observe some possible patterns of the mixed up traffic. 1) Multiple feedbacks: Multiple feedbacks can be generated with concurrent events of a device. 2) Random order: Concurrent events and corresponding packets can be in indeterminate orders. 3) Repetitive events: Some of the concurrent events can be performed more times than expected. 4) Missed events: Some events can be missed. 5) Coalesced packets: packets generated with distinct events can be coalesced to form a new packet. 6) Changed packets: Feedback packets generated with concurrent events can be indeterminate. Figure 6 illustrates how likely each possible pattern happens. There can be more complicated patterns when we consider more events/sub-events. Fortunately, we can still identify fine-grained sub-events with most of these patterns (except coalesced packets) with adequate precision and recall rate.

It should be noted that, increasing the recall rate by identifying fine-grained sub-events instead of the whole trigger event, is in theory at the expense of precision rate. This is because the information entropy of the fingerprints for a sub-event is smaller than that for a trigger event. Moreover, the precision rate of identifying a trigger event can be lower than each of its sub-event. For traffic with multiple feedbacks, multiple status change sub-events might be mistakenly identified. This is confirmed by our experiment as illustrated in Table II. For Yeelight LED Bulb 1 (the 5^{th} device), if it is operated by IFTTT Applet (the 3^{rd} operation of the device), the precision rate of identifying sub-events is 100% while that of identifying the whole trigger event drops to 99.01%.

RTX-IFTTT is designed to increase recall rate at the expense of precision rate due to two reasons. Firstly, the increment in recall rate is significant while the decrement in precision rate is always negligible. Secondly, notification by Check Now tolerates identification errors but not misses. The final trigger event identification performance is provided in Table II.

C. Check Now Vs. Webhooks

On identifying a trigger event, RTX-IFTTT immediately notifies IFTTT by using either the Check Now interface or

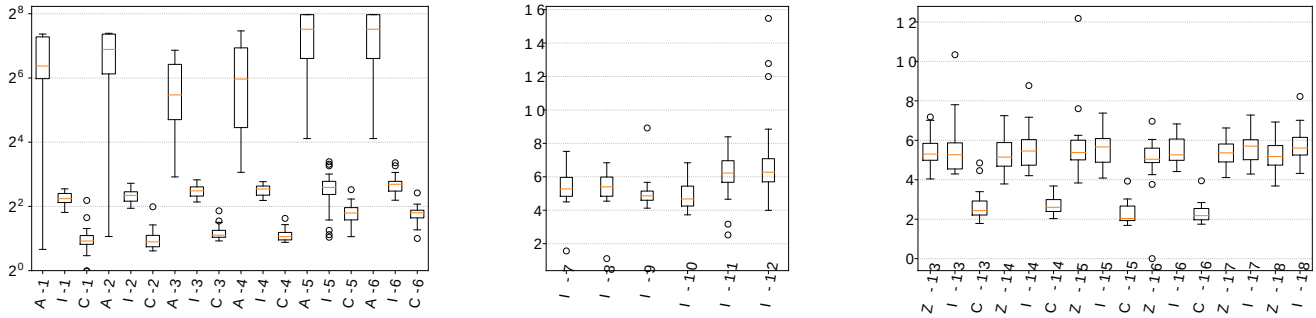
6

Event	Operations	Fingerprints
WeMo Smart Plug switch on/off	Manual	322"33
	APP	351#33"774"33

+

Switch WeMo Smart Plug on/off via APP, then switch it on/off manually within seconds.

#



(a) Applets with IFTTT Triggers and Actions (b) Applets with Non-IFTTT Triggers (c) Cross-Platform (IFTTT and Zapier) Connections

Fig. 8. Runtime performance of single-platform Applets and cross-platform Applets in IFTTT. RTX-IFTTT. Pre x A- indicates that Applet is executed directly by IFTTT, C- indicates that RTX-IFTTT noti es IFTTT by Check Now, I- indicates that RTX-IFTTT noti es IFTTT by Webhooks, Z- indicates that RTX-IFTTT noti es Zapier by Webhooks. The number indicates the serial number of an applet in Table IV. RTX-IFTTT greatly reduces the execution delay from roughly 2min to 2sec by Check Now or 5sec by Webhooks, and it enables connections across platforms.

TABLE IV
APPLETS (CONNECTIONS) USED IN EXPERIMENTS IN FIG. 8

#	Triggers	Actions
1	Smart Life Smart Strip is on	Turn on WeMo Smart Plug
2	Smart Life Smart Strip is off	Turn off WeMo Smart Plug
3	WeMo Smart Plug is on	Turn on Smart Life Smart Strip
4	WeMo Smart Plug is off	Turn off Smart Life Smart Strip
5	Smart Life Smart Strip is on	Turn on Yeelight Bulb 1
6	Smart Life Smart Strip is off	Turn off Yeelight Bulb 1
7	MiJia Smart Plug is on	Turn on Smart Life Smart Strip
8	MiJia Smart Plug is off	Turn off Smart Life Smart Strip
9	MiJia Smart Plug is on	Turn on WeMo Smart Plug
10	MiJia Smart Plug is off	Turn off WeMo Smart Plug
11	MiJia Smart Plug is on	Turn on Yeelight Bulb 1
12	MiJia Smart Plug is off	Turn off Yeelight Bulb 1
13	Smart Life Smart Strip is on	Add row to Google Sheets
14	Smart Life Smart Strip is off	Add row to Google Sheets
15	WeMo Smart Plug is on	Add row to Google Sheets
16	WeMo Smart Plug is off	Add row to Google Sheets
17	MiJia Smart Plug is on	Add row to Google Sheets
18	MiJia Smart Plug is off	Add row to Google Sheets

and then match it with the ngerprints of this device. For most devices, we consider the trigger event be "switch on/off". The ngerprints for switching on and that for switching off a device are always the same. Due to this reason, RTX-IFTTT maintains a local variable for each device to save the current state of that device. In the meanwhile, RTX-IFTTT discovers for each device whether it is online/of ine according to the cyclic packets (e.g., ping/pong and heartbeat). If the device is supposed to be of ine for some time, the state of the device is updated with the noti cation by Check Now.

Each operation is at rst performed 20 times, and the generated packet sequences are collected to generate the ngerprint(s) (calculated by Equation 1). The operation is then performed additional 100 times for identifying the sub-events. All packets generated in the latter 100 experiments are collected sequentially for identi cation, so the identi ed number of a certain sub-event can be greater/smaller than 100 in case of errors/misses. Then the trigger events are identi ed according to method described in Sec.III-B and IV-A. The n-gerprints and identi cation performance of trigger events/sub-events for 5 selected devices are provided in Table II.

In an H-IoT environment, devices are often supposed to be operated remotely via APPs or even automatically via Applets. For sub-event identi cation, the precision rate is near perfect (is always greater than 98.5%). However the recall rate is not at all adequate (sometimes drops to 68.5%). For identi cation of the whole trigger events, the precision rate drops a little bit in comparison with that of sub-events, but is still near perfect (is always greater than than 98%). The recall rate is signi cantly increased and near perfect (is always greater than 99.5%). These results validate the identi cation performance of RTX-IFTTT when devices are not operated manually.

Results for other devices. The identi cation performance for other devices is also near perfect. We make the following conclusions. 1) For normal H-IoT devices, if they are not operated manually, the precision and recall rate are both near perfect. For example, event identi cation for Qing Mi Smart Strip (turning on/off 327 times) and Yeelight Bulb 1S (turning on/off 327 times) both achieve 99.08% precision rate, 100.00% recall rate, and 99.54% F1-score. 2) For WiFi enabled sensors, the precision and recall rate are both near perfect. For example, event identi cation for Smart Life PIR Motion (updating data 50 times) achieves 100.00% precision rate, recall rate, and F1-score. 3) Even for hub/gateway which connects multiple wireless sensors (ZigBee or Z-Wave enabled), the precision and recall rates based on the integrated traf c are still near perfect. For example, event identi cation for MiJia multi-purpose gateway (updating data from motion sensor, door sensor or temperature/humidity sensor 689 times) achieves 98.99% precision rate, 99.27% recall rate, and 99.13% F1-score. For sensors, the edge can only identify events of updating data, but cannot identify trigger events which are mainly based on speci c values of sensor data. RTX-IFTTT must use the Check Now interface and rely on IFTTT platform to determine whether the trigger event is satis ed.

C. Runtime Performance of Single-Platform Applets

In this experiment, we compared the runtime performance of IFTTT Applets executed by IFTTT and that by RTX-IFTTT. The Applets are listed in Row 1 to 6, Table IV. Each Applet is executed directly by IFTTT 40 times, and then by RTX-IFTTT with noti cation by Check Now 40 times and then by

8

Webhooks 40 times. The results are as illustrated in Fig. 8(a), the Applet execution delay by IFTTT ranges from 1m 56sec. RTX-IFTTT greatly reduces the average execution delay from roughly 2min to 2sec by Check Now or 5sec by Webhooks.

Results for other Applets. The runtime performance for other devices/Applets is quite similar to that illustrated in Fig. 8(a). The average delay for IFTTT is always around 2min, and that for RTX-IFTTT ranges from 2sec to 6sec. The only exception deals with Ring video doorbell, when the trigger event is "new ring detected". Applets with this trigger event are executed by IFTTT extremely fast (the average delay is 2sec), faster than that by RTX-IFTTT. One possible reason for this exception is that the vendor of this device implements the Realtime API for its trigger service.

D. Runtime Performance of Cross-Platform Connections

We conduct experiments to validate that RTX-IFTTT enlarges IFTTT's ecosystem by considering connections of non-IFTTT triggers to IFTTT actions. We choose MiJia Smart Plug which is not supported by IFTTT to generate trigger events. We consider 6 trigger-Webhooks connections as listed in Row 7 to 12, Table IV, and run each Applet 40 times. The runtime performance is as illustrated in Fig. 8(b). The average execution delay is only about 5sec.

We also conduct experiments to validate that RTX-IFTTT enables cross-platform connections. In this experiments, we choose two platforms IFTTT and Zapier. We consider "Add row to Google Sheets" as the action of each connection, and establish Webhooks for this action in both IFTTT and Zapier. We construct Applets (or connections) as listed in Row 13 to 18, Table IV. Each Applet is executed by RTX-IFTTT with notification by IFTTT Webhooks 40 times, then by Zapier Webhooks 40 times, and by IFTTT Check Now 40 times if this Applet can be established in IFTTT platform. The runtime performance is illustrated in Fig. 8(c). The average execution delay of cross-platform connections in RTX-IFTTT is about 5sec for both IFTTT Webhooks and Zapier Webhooks, and that for IFTTT Check Now is about 2sec.

VI. Related Work

This section briefly surveys related techniques.

A. Device Action Inference

There are already many researches on device action inference based on traffic analysis in H-IoT environment. Mollers

REFERENCES

[1] IFTTT, "IFTTT Website," [online], https://ifttt.com, Accessed JAN. 2022.

[2] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, "Trigger-Action Programming in the Wild: An Analysis of 200, 000 IFTTT Recipes," in Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2016, pp. 3227–3231.

[3] IFTTT, "IFTTT Documentation," [online], https://platform.ifttt.com/docs/, Accessed JAN. 2022.

[4] X. Mi, F. Qian, Y. Zhang, and X. Wang, "An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance," in Proceedings of the Internet Measurement Conference, IMC 2017, pp. 398–404.

[5] F. Möllers, S. Seitz, A. Hellmann, and C. Sorge, "Short Paper: Extrapolation and Prediction of User Behaviour from Wireless Home Automation Communication," in 7th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec 2014, pp. 195–200.

[6] B. Copos, K. Levitt, M. Bishop, and J. Rowe, "Is Anybody Home? Inferring Activity From Smart Home Network Traf c," in IEEE Security and Privacy Workshops, SP Workshops 2016, pp. 245–251.

[7] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit: Monitoring Smart Home Apps from Encrypted Traf c," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, pp. 1074–1088.

[8] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior Transparency and Control for Smart Home IoT Devices," in 12th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec 2019, pp. 128–138.

[9] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-Level Signatures for Smart Home Devices," in 27th Annual Network and Distributed System Security Symposium, NDSS 2020.

[10] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-A-Boo: I See Your Smart Home Activities, Even Encrypted!" in 13th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WeSec 2020, pp. 207–218.

[11] B. Charyyev and M. H. Gunes, "IoT Event Classi cation Based on Network Traf c," in 39th IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2020, pp. 854–859.

[12] Samsung, "SmartThings," [online], https://www.smartthings.com, Accessed JAN. 2022.

[13] Apple, "HomeKit," [online], https://www.apple.com/ios/home/, Accessed JAN. 2022.

[14] Zapier, "Zapier website," [online], https://zapier.com/, Accessed JAN. 2022.

[15] HomeAssistant, "Home assistant website," [online], https://www.home-assistant.io/, Accessed JAN. 2022.

[16] Selenium, "SeleniumHQ Website," [online], http://www.seleniumhq.org/, Accessed JAN. 2022.

[17] M. Z. Sha q, L. Ji, A. X. Liu, J. Pang, and J. Wang, "A First Look at Cellular Machine-to-Machine Traf c: Large Scale Measurement and Characterization," in ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2012, pp. 65–76.

[18] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "SmartAuth: User-Centered Authorization for the Internet of Things," in Proceedings of the 26th USENIX Security Symposium, USENIX Security 2017, pp. 361–378.

[19] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards Automating Risk Assessment of Mobile Applications," in Proceedings of the 22th USENIX Security Symposium, USENIX Security 2013, pp. 527–542.

[20] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps," in [18]pp.e(u)14.993v7 (P)15.00erurit(s,503.99979 it()-378.99804 (Mobi4(Y)121249979 (Apps,)69.99919 (t()-378.99883 (in)]TJ /T1_1 7.97 d)