An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance

Xianghang Mi
Indiana University Bloomington
xmi@indiana.edu

Feng Qian
Indiana University Bloomington
fengqian@indiana.edu

Ying Zhang
Facebook Inc.
zhangying@fb.com

XiaoFeng Wang
Indiana University Bloomington
xw7@indiana.edu

ABSTRACT

IFTTT is a popular trigger-action programming platform whose applets can automate more than 400 services of IoT devices and web applications. We conduct an empirical study of IFTTT using a combined approach of analyzing data collected for 6 months and performing controlled experiments using a custom testbed. We profile the interactions among different entities, measure how applets are used by end users, and test the performance of applet execution. Overall we observe the fast growth of the IFTTT ecosystem and its increasing usage for automating IoT-related tasks, which correspond to 52% of all services and 16% of the applet usage. We also observe several performance inefficiencies and identify their causes.

CCS CONCEPTS

• **Networks → Home networks**; *Network performance evaluation*;
• **Computer systems organization → Embedded systems**;

KEYWORDS

IFTTT, IoT, Measurement

ACM Reference Format:
Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance. In *Proceedings of IMC '17*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3131365.3131369

1 INTRODUCTION

Consumer Internet of Things (IoT) devices such as smart home gadgets are becoming increasingly popular. For example, Amazon has sold more than 5 million Amazon Echo devices in just two years since their debut in late 2014 [1]. Among many challenges faced by today's consumer IoT systems, *policy management* plays a critical role in ensuring scalable, automated, secure, and resource-efficient interactions among devices. Take smart home as an example. In our measurement (§3), we observe more than 20 types of smart home devices such as light, security camera, thermostat, A/C, washing machine, sprinkler, doorbell, garage door, lock, refrigerator, and even smart egg tray [6]. These devices are highly heterogeneous in terms of their vendors, form factors, computation power, networking capabilities, and programming interfaces. It is non-trivial to manage each device separately, letting alone managing proper policies to coordinate them to accomplish complex tasks.

In this paper, we conduct an empirical characterization of IFTTT (IF This Then That [4]), a task automation platform for IoT and web services. Through IFTTT, end users can easily create policies that connect IoT devices or bridge IoT devices with web services such as *"add 'buying eggs' to my iPhone reminder when there are no more than 3 eggs in the fridge"*. We select IFTTT because it is the most popular one among a plethora of commercial task automation platforms [2, 10, 11, 13–15]. As of early 2017, IFTTT has more than 320,000 automation scripts (called "applets") offered by more than 400 service providers. The applets have been installed more than 20 million times. More importantly, unlike other task automation platforms that are mainly dedicated to web services, more than half of IFTTT services are IoT devices related, as to be measured in §3.2. This makes IFTTT a perfect platform to profile the interactions between web services and IoT devices. The success of IFTTT is attributed to several technical factors. First, it employs very simple trigger-action API (TAP [18, 23]); end users only need to specify a trigger and an action to construct an applet. Second, IFTTT supports both IoT and non-IoT services using a unified HTTP RESTful interface. Third, it uses crowdsourcing to enrich the applet library by allowing users to create applets and sharing them with other users.

Despite the popularity of IFTTT, there is a lack of systematic understanding of its ecosystem, usage, and performance. The goal of this paper is thus to perform in-depth studies of these important aspects. However, our measurement study faces several challenges. First, the IFTTT ecosystem involves multiple policy stakeholders (*e.g.,* end users, IoT devices, service providers, web applications, and the IFTTT engine itself) incurring complex interactions. Second, observing only from end users' perspective has limited visibility, and thus is challenging to gain insights into what happens "under the hood" for applet execution. Third, conducting long-term measurement against IFTTT faces several practical challenges such as automation.

To address the above challenges, we built a testbed to automatically monitor and profile the whole process of applet execution.

IMC '17, November 1–3, 2017, London, UK

ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131369

To further explore potential performance bottlenecks and to characterize interactions between different entities, we launched our own IFTTT service that controls our home-deployed IoT devices and third-party web services. The testbed along with our own IFTTT service enable us to interact with IFTTT from both end users' and service providers' perspectives, and to instrument the applet execution at multiple vantage points.

We then study the usage of IFTTT in the wild by collecting and analyzing its services and applets for six months. We found that 52% of services and 16% of applet usage are IoT-related. With more than 200 IoT services identified, IFTTT indeed provides a way to identify popular smart home and wearable devices on the market in a "centralized" manner. We further provide detailed taxonomies of services and their interactions, as well as characterize how users contribute to IFTTT's applet library.

We further go beyond passive measurements by conducting in-lab controlled experiments to understand the applet execution performance. Our results suggest that actions of many applets (including some "realtime" applets such as turning on the light using a smart switch) cannot be executed in real time when their triggers are activated. The delays are long (usually 1 to 2 minutes) with huge variance (up to 15 minutes). We found the delay is caused by IFTTT's long polling interval. We also study the scenarios when multiple applets execute sequentially and concurrently, and find their performance is often suboptimal. For example, chained applets can form explicit and implicit "infinite loops", causing resource waste or even damage of the physical devices.

Overall, this paper makes three major contributions: (1) developing a measurement testbed with self-implemented IFTTT service and using them to profile the IFTTT ecosystem (§2), (2) conducting an in-depth characterization of service and applet usage (§3), and (3) using the testbed to measure the IFTTT applet execution performance (§4). Based on our findings, we provide recommendations in §6. We summarize related work and conclude the paper in §5. All data and code in this project can be found at:

https://www.cs.indiana.edu/~fengqian/ifttt_measurement

2 UNDERSTANDING THE ECOSYSTEM

IFTTT is a trigger-action programming (TAP) platform [18, 23] that allows end users to create conditional rules in the form of "if A then B" where A is called a *trigger*, B is called an *action*, and the entire rule is called an *applet*. Triggers and actions are often provided by IoT vendors and web service providers. The applets are constructed by users via picking triggers and actions from (usually different) third-party *partner services* (or *services* for short)[1]. A service abstracts functionalities provided by web applications or IoT devices, and it usually provides multiple triggers and actions. For example, consider the following applet: *automatically turn your hue lights blue whenever it starts to rain*. In this applet, the trigger (raining) is from the weather service and the action (changing the hue light color) belongs to the service provided by Philips Hue [8], a smart LED lamp vendor. A service usually exposes multiple triggers and/or actions. Both the trigger and action may have *fields* (*i.e.,* parameters) that customize the applet, such as the light color.

[1]Historically, an applet and a service in IFTTT were called "recipe" and "channel", respectively. We do not use these old names.

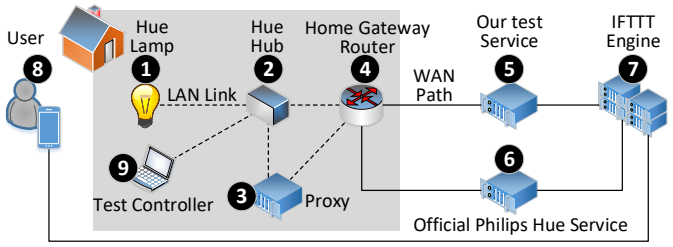

Figure 1: Our IFTTT testbed.

2.1 The Measurement Testbed

Multiple entities play roles in the IFTTT ecosystem: the centralized *IFTTT engine* executes the applet by contacting and coordinating the services; *partner services* (such as Philips Hue) respond to IFTTT's requests by testing the trigger condition or executing the action; *IoT devices and web applications* are controlled by the services to implement the policy; finally, *end users* can define the applets through IFTTT's mobile app or web interfaces.

The above players incur potentially complex interactions. To understand them, we set up a measurement testbed using IFTTT, commodity IoT devices, and commercial web apps. From the measurement perspective, a challenge here is that an end user is sitting at the "edge" of the ecosystem and henceforth does not have visibility of how partner services interact with the IFTTT engine. To overcome this limitation, we obtained a service provider testing account from IFTTT. By doing so, we essentially become a service provider partnering with IFTTT and can publish our own services. Our services support triggers and actions for both IoT devices and web applications. For the former, we purchased four popular off-the-shelf smart home devices: Philips Hue smart lights, WeMo Light Switch (programmatically controlling any home light), and Amazon Echo Dot (smart speaker connecting to Alexa, Amazon's personal assistant service), and Samsung SmartThings Hub (controlling various home appliances). Our testbed also supports several web applications such as Gmail and Google Drive.

For each of the above smart devices and web apps, our service leverages its API to get and set its states. We illustrate this using Philips Hue as an example. As shown in Figure 1, we run our service on a server ❺ in our lab. The actual Hue lamp ❶ and its hub (*i.e.,* controller, ❷) are located at an author's home. For security, most home deployed devices only accept access from a 3rd-party host in the same LAN so we deployed in the home LAN a local proxy ❸ which acts as a bridge for communication between our service server and local devices. Our local proxy communicates with the devices through different protocols such as the Hue RESTful Web API [9] for the Hue hub and UPnP (Universal Plug and Play) [**?**] for the Wemo Switch. We design a custom protocol between the local proxy ❸ and our service server ❺ both of which we have control. Then, The communication path between the lamp and our service is thus Hue Lamp ❶–Hue Hub ❷–Local Proxy ❸–Gateway Router ❹–Our Service Server ❺. Note that for the official Hue service ❻, it can directly talk to the hub using a proprietary protocol so the path is Hue Lamp ❶– Hue Hub ❷– Gateway Router ❹– Hue Service ❻. Our service server ❺ and the IFTTT engine ❼ communicate using the IFTTT's web-based protocol [5]. The services for other smart home devices are developed in a similar manner. For web apps, our

service directly talks with Google using its App API [3]. Our overall implementation efforts for the testbed involve 1620 LoCs in PHP and 2900 LoCs in Python. The Test Controller ❾ automates the controlled experiments to be described in §4.

2.2 Profiling Interactions Among Entities

We leveraged the testbed and self-implemented IFTTT service to profile the interactions among the entities within the IFTTT ecosystem. The high-level approach is to monitor the message exchanges at our vantage points (*e.g.,* Our service server ❺ and the local proxy ❸ in Figure 1). It seems that the interactions can be obtained from the IFTTT documentation. However, we emphasize that an experimental approach is necessary for several reasons: it helps verify the actual system behavior and detect any deviation or unexpected behaviors; it provides details not revealed by the spec; it also quantifies the system performance and pinpoints inefficiencies as to be detailed in §4 and §6. We next describe our observations.

• To publish our service, our service server ❺ exposes to IFTTT ❼ a base URL such as **`https://api.myservice.com`** and other options such authentication configurations. Each trigger or action has a unique URL under the base URL, such as **`https://api.myservice.com/ifttt/actions/turn_on_light`**. IFTTT will generate for the service a key, which will be embedded in future message exchanges between our service server ❺ and IFTTT ❼ for authentication.

• To construct an applet, the user ❽ directly visits IFTTT ❼ using web or smartphone app, and selects the trigger/action services, the trigger/action, and their fields. Many triggers/actions need to authenticate the user. This is done using the OAuth2 framework [7]. The user will be directed to the authentication page that is usually hosted by service providers and asked for her credentials. An access token will be generated and cached at IFTTT ❼ to make future applet execution fully automated.

• In the online applet execution phase, IFTTT ❼ periodically polls the trigger service (*e.g.,* Our service server ❺). The polling query is encapsulated into an HTTPS POST message (with the access token, the service key, and a random request ID) sent to the trigger URL. The trigger service will then determine if the trigger condition is met by either active polling or having the target device/app push trigger events (our testbed uses the push approach for IoT devices and the polling approach for web apps). If the trigger is activated, the trigger service will notify IFTTT ❼ (passively through responding the poll from IFTTT ❼), which will, in turn, contact the action URL. Finally, the action service will execute the action.

3 UNDERSTANDING IFTTT USAGE

To gain a holistic view of IFTTT usage, we crawl its services, triggers, actions, and applets. We describe the data collection methodology in §3.1 and our findings in §3.2.

3.1 Data Collection Methodology

To begin with, we parse the IFTTT partner service index page to get a list of all services. Then through reverse engineering the URLs of applets' pages, we observe that the URLs can be systematically retrieved by enumerating a six-digit applet ID. Using this method, we managed to fetch more than 300K published applets. For each applet, we retrieved the following information from its page: applet name, description, trigger, trigger service (the service that the trigger belongs to), action name, action service, and add count. Add count is the number of this applet being installed by users. It quantifies the popularity of an applet and is similar to the installation count of a mobile app.

Table 1: Breakdown of IFTTT partner services.

Service Category	% Services	Trigger AC %	Action AC %
1. Smarthome devices (*e.g.,* Light, thermostat, camera, Amazon Echo)	37.7%	6.4%	7.9%
2. Smarthome hub / integration solution (*e.g.,* Samsung SmartThings)	9.3%	0.8%	1.0%
3. Wearables (*e.g.,* smartwatch, band)	2.7%	1.6%	1.0%
4. Connected cars (*e.g.,* BMW Labs)	2.0%	0.5%	0.1%
5. Smartphones (*e.g.,* battery, NFC)	3.7%	11.0%	13.8%
6. Cloud storage (*e.g.,* Google Drive)	2.5%	0.6%	13.6%
7. Online service and content providers (*e.g.,* weather, NYTimes)	8.8%	20.0%	1.9%
8. RSS feeds, online recommendation	2.2%	9.8%	0.1%
9. personal data & schedule manager (*e.g.,* note taking, reminder)	10.3%	11.2%	27.4%
10. Social networking, blogging, photo/video sharing (*e.g.,* Facebook)	5.6%	17.7%	17.3%
11. SMS, instant messaging, team collaboration, VoIP (*e.g.,* Skype)	4.7%	0.8%	3.1%
12. Time and location	1.2%	14.1%	0.0%
13. Email	1.0%	4.4%	12.8%
14. Other	8.3%	1.3%	0.2%

Table 2: Compare our IFTTT dataset with [28].

Aspect	Our Dataset	The Dataset of [28]
# Applets	320K	224K
# Channels	408	220
# Triggers	1490	768
# Actions	957	368
# Adoptions	24 millions	12 millions
# Applet Contributors	135K	106K
# Snapshots	25, one each week	1
Duration	Nov 2016 to Apr 2017	Sep 2015

We implemented the above data collection tool. Every week from November 2016 to April 2017, we used the tool to take a "snapshot" of the IFTTT ecosystem by performing the aforementioned crawling. About 200 GB data was collected during the six-month period (∼12GB each snapshot). Note our data only contains the applets that are publicly shared, as opposed to users' private applets. Previous IFTTT measurements [27, 28] also share this limitation. Note that the crawling methodology is conceptually similar to the one used by [28], but we captured a much larger dataset for a longer period as shown in Table 2.

3.2 Data Characterization

We now analyze the data to reveal the up-to-date landscape of IFTTT usage. First we notice that across the weekly snapshots collected over the six-month period, services and applets kept growing steadily. Compared to 11/24/2016, on 4/1/2017, the number of services, triggers, actions, and applet add count increase by 11%, 31%, 27%, and 19%, respectively, indicating the IFTTT platform is gaining popularity. Next, without loss of generality, we characterize a particular snapshot collected on 3/25/2017 where the number of services, triggers, actions, applets, and total add counts are 408, 1490, 957, 320K, and 23M respectively. We also notice the *significant increase* of the applet size compared to prior studies: 67K in 6/2013 [27], 224K in 9/2015 [28], and ∼320K in our dataset.

Service Semantics. For each service, we examine its service description, trigger list, action list, and its external website if needed. We then classify the service into one of the 13 categories listed

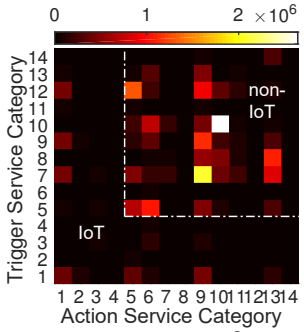

Figure 2: Heat map of interactions

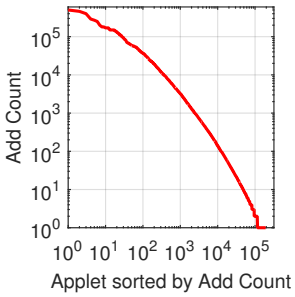

Figure 3: Add count per applet.

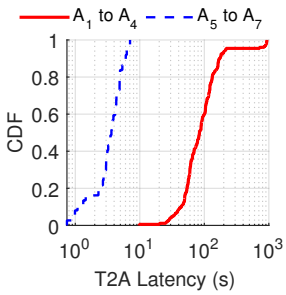

Figure 4: T2A latency for two applet groups.

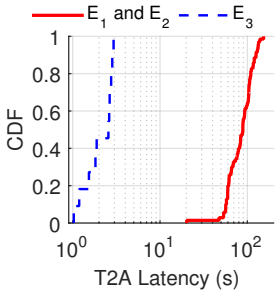

Figure 5: T2A latency under three scenarios for A_2.

in Table 1 based on our domain knowledge. Given the number of services is moderate (~400), the classification was done manually to ensure its accuracy. In Table 1, Service Category 1 to 4 relate to IoT devices, including controlling specific smart home devices, general smart home hubs/controllers, wearable devices, and connected cars. Within Category 1, we observed more than 20 types of devices as exemplified in §1. Category 5 to 13 belong to non-IoT services such as web applications, cloud storage, RSS feeds, smartphone, email, time, and location. For each service category, Table 1 lists (in percentage) its number of services, its trigger add count (*i.e.,* the total add count of applets whose triggers belong to a service within this category), and its action add count. *Overall, we find that the services provided by IFTTT are extraordinarily rich.* More than half (51.7%) of services are for IoT devices. They account for 16% of the overall IFTTT applet usage (based on the add count). This again contrasts the most recent IFTTT measurement in 2015 [28], which barely observed IoT related channels and recipes (*i.e.,* services and applets).

IoT Usage. Table 3 lists the top IoT-related trigger services, action services, triggers, and actions. Based on their add counts, the top-3 services are Alexa (Amazon's intelligent personal assistant for smart home devices like Amazon Echo), Philips Hue (smart lighting), and Fitbit (a wearable activity tracker), followed by other popular gadgets such as Nest Thermostat and Google Assist – IFTTT indeed provides a way to identify popular smart home and wearable devices on the market as well as their key usage scenarios in a "centralized" manner (assuming their vendors publish partner services). The vast majority of triggers and actions (*e.g.,* "turn on light") and henceforth their applets are rather simple, due to the simple interfaces exposed by most IoT devices, as well as the fact that most tasks (in the smart home context) we want to automate are indeed simple [27]. Figure 2 plots a heat map illustrating the interaction among different service categories. The intensity of the color block at Row i and Column j indicates the add count of applets whose trigger and action belong to service category i and j, respectively. We find that IoT services may serve as both triggers (usually paired with service categories of 1, 5, 9) and actions (paired with service categories of 1, 7, 9, 12).

Regarding non-IoT services, as illustrated in Table 1 and Figure 2, popular services used as triggers include social networks (Category 10), online services (Cat. 7), RSS feeds (8), and time/location (12). Many content providers such as YouTube and NYTimes have their own IFTTT partner services. For actions, unlike IoT actions that usually perform device control, non-IoT actions are mostly used to notify users via push notification/email (Cat. 9), to log events to cloud storage (6), or to publish posts to social networks (10).

Applet Properties. Now we shift our focus from services to applets. Figure 3 quantifies the overall applet usage by ranking the applets (X-Axis) by their add count (Y-Axis). It exhibits a heavy-tail distribution: the top 1% (10%) of applets contribute 84.1% (97.6%) of the overall add count. We thus next focus on the top applets. For non-IoT applets whose neither trigger nor action relates to IoT, we observe several top use cases such as syncing different social networks, getting notifications from online services, and triggering actions at certain time/locations. For IoT applets whose either trigger or action relates to IoT, IFTTT acts as a virtual smart home hub in the cloud by coordinating smart home devices (*e.g.,* Amazon Echo and lights). More interestingly and commonly, IFTTT helps bridge IoT devices with non-IoT services in the cloud.

In IFTTT, besides service providers, an end user can also create her applets and share them with other users by publishing them on her "channel". In our dataset, we observe 135,544 user channels, which is several orders of magnitude more compared to the number of services (around 400). Although triggers and actions can only be provided by services, most applets (98%) are home-made by users. The number of published applets per user also follows a heavy-tail distribution: the top 1% (10%) of users contribute 18% (49%) of all applets. Also 86% of add count belong to user-made applets, which thus dominate the applet usage.

4 APPLET EXECUTION PERFORMANCE

This section measures the performance of IFTTT by conducting controlled experiments.

Trigger-to-Action (T2A) Latency is a key performance metric for applet execution. It is the delay from time T_T when the trigger condition is met to time T_A when the action is executed. We pick seven popular applets A_1 to A_7 listed in Table 4 and measure their T2A latency using our testbed in §2.1. All these applets were created and deployed using regular user accounts. They are chosen to cover a variety of IoT devices and their interactions with web services: A_1 to A_4 cover different usage scenarios (IoT→WebApp, IoT→IoT, WebApp→IoT, and WebApp→WebApp); A_5 to A_7 use Amazon Alexa as the trigger, which will be found to be treated specially by IFTTT. To facilitate the experiments, we introduce in Figure 1 a Test Controller ❾ that serves two roles. First, it automates the experiments by activating the trigger. For example, to programmatically control Alexa, it plays pre-recorded voice commands. The second role of ❾ is to measure the T2A latency by recording T_T and T_A.

Figure 4 shows the T2A latency for the seven applets using the official IFTTT partner services (*e.g.,* Hue Service ❻ as opposed to

Table 3: Top trigger services, action services, triggers, and actions involving IoT. Add count (in million) are shown in parentheses.

Top Trigger Services	Top Action Services	Top Triggers	Top Actions
Amazon Alexa[a] (1.2)	Philips Hue[h] (1.2)	Say a phrase (Alexa)	Turn on lights (Hue)
Fitbit[b] (0.2)	LIFX[i] (0.2)	Item added to todo list (Alexa)	Change color (Hue)
Nest Thermostat[c] (0.1)	Nest Thermostat[c] (0.2)	Say a phrase (Google Assistant)	Blink lights (Hue)
Google Assistant[d] (0.1)	Harmony Hub[j] (0.2)	Ask what's on shopping list (Alexa)	Turn on color loop (Hue)
UP by Jawbone[e] (0.1)	WeMo Smart Plug[k] (0.1)	Daily activity summary (Fitbit)	Set temperature (Nest Thermostat)
Nest Protect[c] (.07)	Android Smartwatch (0.1)	Item added to shopping list (Alexa)	Start activity (Harmony Hub)
Automatic[f] (.06)	UP by Jawbone[e] (.09)	New sleep logged (Fitbit)	Send a notification (Android watch)

[a] https://www.amazon.com/echo [b] https://www.fitbit.com/ [c] https://nest.com/ [d] https://assistant.google.com/ [e] https://jawbone.com/ [f] https://www.automatic.com/
[h] http://www2.meethue.com [i] https://www.lifx.com/ [j] https://www.logitech.com/harmony-hub [k] http://www.belkin.com/us/F7C030/p/P-F7C030/

Table 4: Popular applets used in controlled experiments.

A_1	If my Wemo switch is activated, add line to spreadsheet.
A_2	Turn on my Hue light from the Wemo light switch.
A_3	When any new email arrives in gmail, blink the Hue light.
A_4	Automatically save new gmail attachments to google drive.
A_5	Use Alexa's voice control to turn off the Hue light.
A_6	Use Alexa's voice control to actviate the Wemo switch.
A_7	Keep a google spreadsheet of songs you listen to on Alexa.

Table 5: Example of applet execution timeline: A_2 under E_2.

t (s)	Event Description
0	Test controller ❾ sets the trigger event
0.04	Custome Proxy ❸ observes the trigger event and notifies Our Server ❺
0.16	❸ receives the confirmation from trigger service ❺
81.1	IFTTT engine ❼ polls trigger service ❺ about the trigger
82.1	IFTTT engine ❼ sends action request to action service ❺
83.0	Aftering querying ❺, ❸ sends the action to the IoT device
83.8	Test controller ❾ confirms that the action has been executed

Our Service ❺). Over a period of three days, the testbed executed each applet 50 times at different time. Before each test, we ensure both local WiFi and the Internet connectivity are good through active probing so the network never becomes the performance bottleneck. For better visualization, we group the latency of A_1 to A_4 and A_5 to A_7 together, because within each group the performance is qualitatively similar. We first examine A_1 to A_4. Their T2A latency values are not only large, but also highly variable, with the 25-th, 50-th, and 75-th percentiles being 58s, 84s, and 122s, respectively. This may not be a big issue for "non-real-time" applets such as A_4, However, for applets like A_2 (using a smart switch to turn on a light), such a long and highly variable latency degrades user experience. In the extreme case, the T2A latency can reach 15 minutes.

To find out the cause of such high latency and variance, we replace the involved service entities with our own implementation, which is known to be performance-wise efficient. We design the following three experiments and run them on our testbed.

- E_1: replace the official trigger service (*e.g.,* Hue Service ❻) with Our Service ❺.
- E_2: replace the trigger and action services with Our Service ❺.
- E_3: in addition to E_2, further replace the IFTTT engine ❼ with our own implementation that follows the IFTTT protocol and performs frequently polling (every 1 second).

Figure 5 plots the T2A latency for A_2 under scenarios E_1/E_2 and E_3 (for each scenario we run 20 tests). The results clearly indicate that the performance bottleneck is the IFTTT engine itself, as E_3 dramatically reduces the T2A latency compared to E_1 and E_2 that exhibit similar performance. Specifically, we observe that IFTTT employs very long polling interval that dominates the overall T2A latency. To illustrate this, Table 5 exemplifies the breakdown of the T2A latency for a typical execution of applet A_2 under the scenario E_2. As shown, our service ❺ is notified about the trigger event at

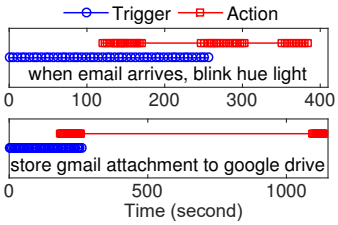

Figure 6: Two examples of sequential applet execution.

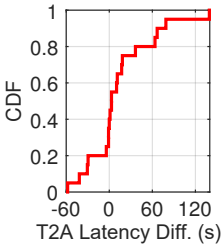

Figure 7: Example of concurrent applet execution.

t=0.16 sec; but the polling request from the IFTTT engine ❼ arrives much later at t=81.1 sec.

Besides performing regular polling, IFTTT also provides *real-time API*, which allows a trigger service to proactively send a notification to IFTTT about a trigger event. Through experiments, we find that using the real-time API brings no performance impact for our service (figure not shown). Note the real-time API merely provides hints to the IFTTT engine, which still needs to poll the service to get the trigger event delivered. In other words, the IFTTT engine has full control over trigger event queries and very likely ignores real-time API's hints. We provide more discussions in §6.

Another observation relates to the low T2A latency of A_5 to A_7 in Figure 4. Since they all use Alexa as the trigger, it is likely that IFTTT customizes the polling frequency or more likely, processes the real-time API hints for some services (such as Alexa) with timing requirements. But as indicated by our previous results (*e.g., A_2*), such customization does not yet cover all applets requiring low latency. When we use our own service to host Alexa, its latency becomes large.

Sequential Execution of Applets. We next test the performance when a trigger is activated multiple times sequentially (every 5 seconds in our experiment). As exemplified in the top part of Figure 6, due to the long and highly variable polling latency, 119 seconds later, the action associated with the first trigger is executed together with a cluster of subsequent actions. The second and third cluster come at 247 and 351 seconds respectively. The actions are sequentially mapped to triggers but the actions' timing is "reshaped" by IFTTT. Such a clustered pattern, which is observed from all triggers for A_1 to A_4, is caused by the batched process of IFTTT polling. Upon receiving a polling query, the trigger service should return many buffered trigger events (up to k) to IFTTT. k is a parameter in the polling query (50 by default). Because each polling query response contains multiple trigger events, the resulting actions naturally form a cluster. The bottom part of Figure 6 shows one extreme case (possibly when IFTTT experiences high workload) where the polling delay between two clusters inflate to 14 minutes.

Concurrent Execution of Applets. Users can create two applets with the same trigger, say "if A then B" and "if A then C" to realize "if A then B and C". When A is triggered, ideally B and C should be executed at the same time. Figure 7 plots the CDF for T2A latency difference between "turn on Hue light when email arrives" and "activate WeMo switch when email arrives", which share the same trigger, across 20 tests. As shown, the T2A latency difference ranges from -60 to 140 seconds. This is because (1) the polling delay is highly fluctuating, and (2) the polling response of one applet cannot be piggybacked with that of another applet. The results indicate that in reality, IFTTT cannot guarantee the simultaneous execution of two applets with the same trigger. This may cause unexpected results to end users (*e.g.,* a user wants to use a smart switch to turn on the heat *and* close the window).

Infinite Loop. Multiple applets can be chained in IFTTT. However, users may misconfigure chained applets to form an "infinite loop" *e.g.,* A triggers B, which further triggers A. An infinite loop may waste resources and even damage the IoT equipment. Through experiments, we confirm that despite a simple task, no "syntax check" is performed by IFTTT to detect a potential infinite loop. Furthermore, we also experimentally confirm that an infinite loop may be *jointly* triggered by IFTTT and 3rd-party automation services. For example, a user applies the following IFTTT applet: *add a row in my Google Spreadsheet when an email is received.* Meanwhile, the user has also enabled in her spreadsheet the notification feature [12], which sends her an email if the spreadsheet is modified. The applet and the enabled notification thus cause an implicit infinite loop. Since IFTTT is not aware of the latter, it cannot detect the loop by analyzing the applets offline. Instead, some runtime detection techniques are needed.

5 RELATED WORK

IFTTT Characterization. Ur *et al.* investigated the human factor of trigger-action programming in smart home [27]. As a part of the study, they collected 67K IFTTT recipes (*i.e.,* applets) in 2013 to demonstrate that users can possibly create a large number of combinations of triggers and actions. A follow-up CHI note [28] by the same authors analyzed basic statistics of 224K IFTT recipes crawled in 2015. Surbatovich *et al.* [25] used the dataset of [28] to analyze the security and privacy risks of IFTTT recipes. Huang *et al.* [19] investigated how to make IFTTT-style TAP better fit users' mental model. Our study distinguishes from the above in several aspects. First, we provide an up-to-date characterization of the IFTTT ecosystem with new observations that differ from previous measurements (*e.g.,* the dominance of IoT services described in §3.2). Second, we build a real IFTTT testbed and show the interaction among different entities in the ecosystem. Third, we conduct experiments to analyze the applet execution performance.

IFTTT-like Platforms. There also exist commercial platforms such as Atooma [2], WigWag [14], Android Tasker [11], Zipato [15], Stringify [10] and WayLay [13]. Some use more complex languages such as flowchart [10] or even Bayesian Networks [13]. All of them have registered less popularity compared to IFTTT.

Trigger-action Programming (TAP) has been studied for more than a decade. It is oftentimes used in automation for smart homes [17, 21, 26, 29, 30], smart buildings [22], and general IoT/context-aware systems [16, 18, 20, 23]. We instead conduct an empirical study of IFTTT, the most popular commercial TAP platform [19].

6 DISCUSSIONS AND CONCLUSIONS

Performance Improvements. We observe that oftentimes the T2A latency, which is dominated by the polling delay, is long and highly variable[2]. Instead of doing polling, an effective way to reduce the latency is to perform push (or utilize the real-time API with the same concept). However, we believe there are reasons why IFTTT has not yet fully adopted this approach[3]. One reason we can possibly imagine is that, if all trigger services perform push, the incurred instantaneous workload may be too high: IoT workload is known to be highly bursty [24]; for IFTTT it is likely also the case (consider popular applets such as "*update wallpaper with new NASA photo*"). On the other hand, this creates opportunities for predicting the trigger events to perform polling smartly or provisioning the resources for accepting more real-time hints. Such optimizations only need to apply to top applets that dominate the usage (Figure 3).

Distributed Applet Execution. For now, all applet executions need to be handled by the centralized IFTTT engine. In fact, many applets can be executed fully locally by using users' smartphones or tablets as a local IFTTT engine. In this way, the scalability of the system can be dramatically improved. Nevertheless, designing such a hybrid (centralized + distributed) applet execution scheme is challenging in many aspects: what are users' incentives for adopting the local version (maybe better privacy or operating without Internet)? How to determine which applets to execute locally? How to quickly recover when the local IFTTT engine goes down? More research is needed in this direction.

Permission Management. We notice that IFTTT performs coarse-grained permission control at the *service* level: for a service involved in *any* trigger or action installed by the user, IFTTT will need *all* permissions of the service. For example, installing an applet with the trigger "new email arrives" requires permissions for reading, deleting, sending, and managing emails. This facilitates the usability (as the user will not be bothered when future applets involving the same service are installed) but incurs potential security issues (as the "least privilege principle" is violated). We need better permission management schemes that balance the tradeoff between usability and security.

Limitations. We acknowledge that observing the IoT ecosystem from the perspective of IFTTT is interesting but still limited as IFTTT may not cover all available IoT devices and their owners.

To conclude, in this study, we observe the fast growth of the IFTTT ecosystem and its increasing usage for automating IoT-related tasks, which correspond to 52% of services and 16% of the applet usage. We observe several performance inefficiencies and identify their causes. We plan to study future IFTTT features such as *queries* and *conditions* [25].

ACKNOWLEDGEMENTS

We would like to thank our shepherd, Hamed Haddadi, and the anonymous reviewers for their valuable comments and suggestions. This research was supported in part by the National Science Foundation under grant #1629347 and #1566331.

[2]We contacted IFTTT and they confirmed this issue.
[3]In their response, IFTTT said that they were working on the push support.

REFERENCES

[1] Amazon echo - what we know now (updated). http://files.constantcontact.com/150f9af2201/70c07fdd-a197-4505-9476-e83aa726f025.pdf.
[2] Atooma. https://www.atooma.com/.
[3] Google APIs. https://console.developers.google.com/.
[4] IFTTT. https://ifttt.com/.
[5] IFTTT API (2017). https://platform.ifttt.com/docs/api_reference.
[6] IFTTT Egg Minder Service. https://ifttt.com/eggminder.
[7] OAuth 2.0. https://oauth.net/2/.
[8] Philips Hue. http://www2.meethue.com/en-us/.
[9] Philips Hue API. https://www.developers.meethue.com/philips-hue-api.
[10] Stringify. https://www.stringify.com/.
[11] Tasker for Android. http://tasker.dinglisch.net/.
[12] Turn on notifications in a Google spreadsheet. https://support.google.com/docs/answer/91588.
[13] Waylay.io. http://www.waylay.io/index.html.
[14] WigWag smart home. https://www.wigwag.com/home.html.
[15] Zipato. https://www.zipato.com/.
[16] F. Cabitza, D. Fogli, R. Lanzilotti, and A. Piccinno. End-user development in ambient intelligence: a user study. In *Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter*, pages 146–153. ACM, 2015.
[17] L. De Russis and F. Corno. Homerules: A tangible end-user programming interface for smart homes. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2109–2114. ACM, 2015.
[18] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. icap: Interactive prototyping of context-aware applications. In *International Conference on Pervasive Computing*, pages 254–271. Springer, 2006.
[19] J. Huang and M. Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 215–225. ACM, 2015.
[20] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu. Sift: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pages 298–309. ACM, 2015.
[21] S. Mennicken, J. Vermeulen, and E. M. Huang. From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 105–115. ACM, 2014.
[22] A. A. Nacci, B. Balaji, P. Spoletini, R. Gupta, D. Sciuto, and Y. Agarwal. Buildingrules: a trigger-action based system to manage complex commercial buildings. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 381–384. ACM, 2015.
[23] M. W. Newman, A. Elliott, and T. F. Smith. Providing an integrated user experience of networked media, devices, and services through end-user composition. In *International Conference on Pervasive Computing*, pages 213–227. Springer, 2008.
[24] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang. A first look at cellular machine-to-machine traffic: large scale measurement and characterization. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):65–76, 2012.
[25] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1501–1510. International World Wide Web Conferences Steering Committee, 2017.
[26] K. Tada, S. Takahashi, and B. Shizuki. Smart home cards: tangible programming with paper cards. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 381–384. ACM, 2016.
[27] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 803–812. ACM, 2014.
[28] 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 2016 CHI Conference on Human Factors in Computing Systems*, pages 3227–3231. ACM, 2016.
[29] M. Walch, M. Rietzler, J. Greim, F. Schaub, B. Wiedersheim, and M. Weber. homeblox: making home automation usable. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 295–298. ACM, 2013.
[30] J.-b. Woo and Y.-k. Lim. User experience in do-it-yourself-style smart homes. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, pages 779–790. ACM, 2015.