Migrating IoT applications to the Fog

This Page will be Replaced before Printing

Title page logo

Abstract

The goal of this thesis is to investigate an IoT cloud application that consists of littleBits (smart electronics gadgets), Philips Hue lamps (smart home lighting system) and cloud service provider IFTTT. IFTTT provides APIs through well known service providers or device manufacturers to interconnect devices and services. The functionality of this application is to perform actions on the Hue lamps based on an input signal from littleBits. This application introduces high latency because of the various involved cloud services that are used to forward the request from littleBits to Philips Hue lamps. Initially we try to detect the source of the high latency and then we migrate the application logic from the cloud down to a fog node on the local network. With this approach all the requests will be fulfilled through the local network so the status and quality of the Internet connection and cloud services will not affect the functionality.

Dedicated to my parents.

Contents

1. Introduction

1.1 Context

Internet of Things (IoT) refers to the ability of interconnecting various daily usage devices and to exchange data between them. These devices include everyday home devices like coffee makers, cameras, locks, gadgets, lights and more smart devices that are released on the market. According to predictions, 50 billion devices will be connected by 2020. Most of those smart devices are used for home or office automation applications and are helping users to perform their daily tasks efficiently. In order for these devices to be aware of the environment and be able to react on the events that take place they should be constantly connected to the Internet [27] [6].

While IoT expands rapidly, cloud based applications for automating tasks are gaining popularity and attract new users. Smart devices are connected to cloud servers that are controlling the devices while online cloud applications decide on how they should react based on the device data. Hosting applications on the cloud is convenient for the users since interconnecting devices in another way, specially from different manufactures, requires special skills and effort.

In the current project we connect a sensor and an actuator through a cloud service to study the latency problem. As a sensor we use littleBits and as an actuator we use Philips Hue lamps. The two are connected through the IFTTT platform, allowing simple "**if** *this* **then** *that*" applets which act on triggers to execute an action like "**if** *littleBits send a signal* **then** *turn on the lights*". IFTTT is a provider that provides connectivity services between well known service providers like Google, Facebook and others but also for smart gadgets and smart home devices. All these smart devices are connected through cloud services and they either work in a plug and play way or require a simple setup process by the users. When users connect their smart devices or services to IFTTT, they can build applets that are interacting with each other by using APIs provided by IFTTT and its partners.

1.2 Latency in IoT Applications

While cloud based applications architecture is simple and straightforward for the users, it makes their devices totally dependent on reliable

Internet connections and on constantly available cloud servers. Since cloud servers can be located anywhere and possibly far away from the requesting computers, such low latencies and high availability can not be guaranteed.

Delays can arise either on the communication over the Internet or on the servers. The geographical distance between the devices and the data centers and the quality of the Internet connection can delay the response times. Furthermore, the servers might not have enough resources to process the requests at certain moments, which can also affect performance.

High latency or unreachable servers[6] can be considered as similar problems since in real-time tasks a slow application is equally non functional as an unavailable one. Having a VM running on a cloud server, cannot guarantee real time processing of the requests since the user has no control on how the resources of the server are used. Privately owned servers could provide users the ability to enforce their own policies, however an approach like this is expensive and requires technical knowledge that most users do not posses.

Cloud based applications upload and download data to cloud servers in frequent intervals. In case the transferring of the data takes place from a hardline connection, the size of the data might not affect the cost, but if the deployed device is on a 3G/4G connection then the size of the data can increase the cost significantly. When having a hardline connection is impossible, like in airplanes or remotely placed devices it is important that the data can be processed locally thus conserving bandwidth.[6]

In cases where IoT devices are located in the same home or local network, it could be argued that resources are being wasted by having to contact a remote server every time they need to interact with each other instead of performing the tasks locally[6]. Apart from the cost of uploading and downloading data to the cloud servers, some latency is unavoidable, thus causing delays or possible disruption to the services.

Fog computing refers to deploying devices or servers on the edge of the network that aim to help cloud computing overcome some of its limitations. Fog devices (computers, servers etc) are migrating - copying the functionality of cloud servers so they can perform the same tasks using the local network instead of the Internet. Some of the benefits that come with fog computing apart from the improvement in latency are related to reduced transfer of data from and to cloud servers, thus lower costs, better scalability by filtering which data are uploaded to the cloud and higher availability for the provided services. Furthermore, fog computing users can use custom privacy policies so critical data are kept safe in a local network instead of getting exposed online, thus tackling privacy issues.

This thesis addresses the high latency experienced in IoT applications involving several cloud servers of different device manufacturers and IFTTT implementing the cloud application. The focus of the project is to measure the latency on the cloud based application and then migrate the functionality to the fog devices.

LittleBits as Philips Hue lamps have their own cloud servers connected to IFTTT so requests from and to IFTTT can interact with the devices. In our application the littleBits (sensor) request has to reach the Hue lamps (actuator) through IFTTT. In order for that to happen, littleBits request has to initially reach littleBits cloud server. After that, the request is passed to IFTTT to be processed and be forwarded to Philips Hue servers. Philips Hue servers have to obtain the Hue lamps to fulfill the littleBits request. The communication through all these entities introduces high latency. Furthermore, the application relies on the state of the connection and in cases where the Internet was not working we were not able to interact with the lamps at all. This latency could have really serious consequences in cases where, instead of the lamps, a fire alarm or another similar device used for a critical task had to be triggered.

1.3 Migrating to Fog

The goal of this master thesis project is to locate and reduce the latency of the original *littleBits - IFTTT - Hue Lamps* applet to an absolute minimum, and increase the effectiveness and availability of the service by performing most of the tasks locally. IFTTT can be used to interact with smart devices and services but the latency that comes with it can be restricting for performing almost real-time tasks. Long latencies can make real-time applications useless and since data centers are located far away from requesting computers, low latencies cannot be guaranteed. By migrating all this online functionality to the fog devices it is possible to reduce the latency and become independent from a stable Internet connection for performing the same tasks.

1.4 Method

For this project we performed measurements - experiments to estimate the latency of the application. After performing the measurements we implemented solutions with different architectures to estimate the improvement. To carry out the measurements we must be able to sense when the lights turn on after a littleBits request. In order to do that, we used TI CC2650 Sensortag because we could connect external inputs to littleBits and use its light sensor to trigger time measurements. The experiments that are going to be discussed were initially performed at the beginning of the thesis work to get a good understanding of the problem. Since that, each time a new technique was added in the solution there was a new experiment to estimate the latency improvement - if there was an improvement at all. When the fog node was ready we ran an experiment to estimate the overall improvement since the beginning.

2. Background

In this section we will present the main entities that were involved during this thesis.

2.1 IFTTT

IFTTT Inc [23] is a United States based company formed by Linden Tibbets, Alexander Tibbets and Jesse Tane. The initial release of ifttt.com was on September 2011. The use of the service is free and it is supported for web browsers, Android and iOs.

IFTTT is a cloud service where users can use/create applets using well known services or smart devices to automate their tasks. There are pre-made applets by IFTTT partners but users have the ability to create their own applets based on their needs by combining the functionalities of the provided channels. The basic applets have a simple structure and consist of a trigger event and an action event in a one to one relationship. When the event that has been set as a trigger takes place, IFTTT fires the event that has been set as action for that certain trigger.

An IFTTT Channel can be a channel to a device, social media or Internet application. IFTTT partners can publish their channels on IFTTT. Some examples of channels used in IFTTT are Facebook, Twitter, Gmail, littlebits, Philips Hue and many more. The user should first connect his account of the channel he intends to use and then select from a variety of trigger and action events. The channels are using their official APIs to access the user account and fire actions when the pre-defined events take place. These APIs are "black boxes" so there is no possibility for a developer to use them in a custom-made application outside IFTTT or get any insight on how they work.

When using/creating an applet, the user should first select which channel will be used as trigger and then specify which events of the selected channel will be used to trigger the applet. The same series of events should be followed when the user sets an action; first he should select the action channel and then specify the events of that action. The triggers and the actions can be selected from a variety of channels that are provided by IFTTT.

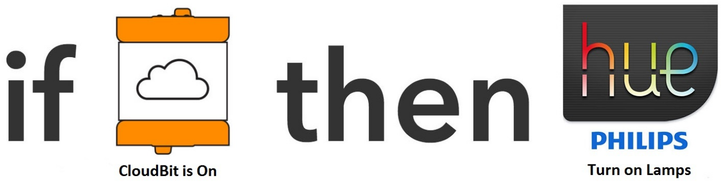


Figure 2.1. if this then that

An example of an applet could be that when a user gets tagged on a new photo on Facebook (trigger) he will automatically receive an email on his gmail account (action).

2.2 LittleBits

LittleBits electronics [16] is a company that creates electronics kits (figure 2.2) in order to familiarize people with electronics. With littleBits, users can prototype applications to perform small daily tasks using the littleBits modules (called bits) like buttons, pressure sensors, servos, buzzers, LED lights and many more. The users are not required to have any technical background to perform simple tasks and create small applications, although bits that require programming and soldering with external devices can be provided as well to build more elaborate and complicated applications. A littleBits kit contains various bits, where each serves a different usage. The bits are connected with each other through their connectors using magnets which makes it easy to use and prohibits the user from connecting them in a wrong way. Each bit has connectors on both ends except the ones that provide power which have connectors only on one end and a power source plug on the other. The connectors contain 3 pins, GND, +5v power supply and an analog signal from 0 to 5v. Each bit serves a different purpose and has a different functionality. The bits can be categorized in groups and can be distinguished by the color of their connectors. More specifically the colors of the groups are:

1. **Blue** bits are used to supply the following bits with power.
2. **Red** bits are used as inputs (buttons, sliders etc), sending signal and affecting the actions of the following bits.
3. **Green** bits are used as outputs (LEDs, baragraphs, servos etc) and act according to the signal they receive from the previous bits.
4. **Orange** bits are used between inputs (red) and outputs (green) to connect / wire littleBits applications with other devices or to help expand these applications.

Some of the red bits, as the buttons, have only a simple on/off functionality while some others as sliders can adjust the level of the signal they are sending from 0 to 5v.

The green bits, based on the level of the signal they receive, they work on that current level/percentage of their total capacity. For example when a baragraph is set as an output and a slider as an input, the baragraph will only have all five LEDs turned on when the slider is on the far end, while two or three LEDs will be on if the slider is placed in the middle. Output bits can not affect the actions of the following bits.

The orange bits are "secondary" and they don't have to be necessarily used in an application but they can be helpful when creating more elaborate applications. They can help with extending the reach by providing some extenders, power multiple bits from the same source - powerBit using forkBit, connect littleBits to external electronic devices

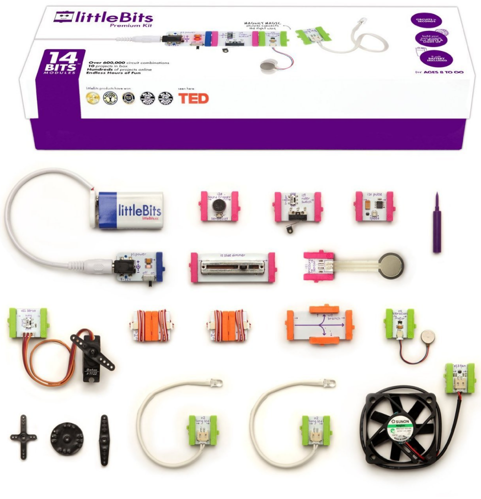


Figure 2.2. LittleBits kit

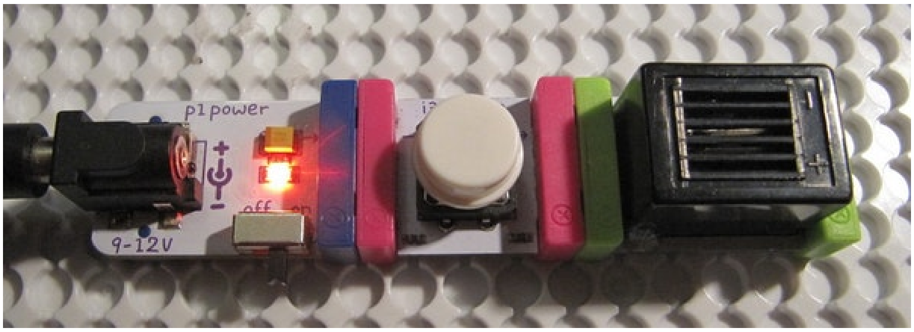


Figure 2.3. Doorbell

using protoBit, add logic to the application using an arduinoBit and of course connect to the cloud using the cloudBit.

Figure 2.3 shows a simple doorbell made out of littleBits. The powerBit (blue connectors) provides power to the rest of the bits. When the buttonBit (red connectors) is pressed it sends signal to the buzzerBit (green connectors) which will make a sound. More bits can be connected before or after the buttonBit and the buzzerBit based on the needs of the application, but always after the powerBit. If for example the doorbell should also turn an LED on when the buttonBit is pressed, a ledBit should be attached after the buttonBit, either before or after the buzzerBit. In case the ledBit is placed before the buttonBit, it will always be on and it will not be affected by the actions on the buttonBit as it will not affect the following bits in any way since it is a **green** bit.

Figure 2.4. Cloud Bit

2.2.1 CloudBit

LittleBits kits include cloudBit (figure 2.4) and a cloud control web application from which a user can set up a cloudBit, configure it, monitor it and interact with it. Cloudbit is a small device that has an onboard wifi dongle and an SD card that runs Linux OS. The cloudBit connects to a wifi network and can be used to either receive inputs from the red bits and show the received signal on cloud control or trigger an input from the cloud control and transform this signal to action on the green bits that follow the cloudBit on a certain application. Red bits have to be placed on the left of the cloudbit and green bits have to be placed on the right side of the cloudbit. The red bits before the cloudBit can not detour the cloudBit and send signal to the green bits following it. The cloudBit has an LED that indicates its status. In order for the cloudBit to be functional the LED should turn into a steady green color after the setup.[15]

2.2.2 LittleBits Cloud Control

LittleBits cloud control is a web based application that is used so the users can interact with their cloudBits and configure them. Initially the user has to connect the cloudBit to his home wifi network. After the connection is successful he can use the cloud control to either receive signal from the cloudBit or to send signal to the cloudBit. The setup is straightforward and the interface is quite simple to use.[15]

2.3 Philips Hue lamps

The Hue lamps are a smart wireless lighting system made by Philips. Hue lamps are screwed into all kind of lighting fixtures like normal lamps and they are controlled from the smartphone or from the computer through the home WiFi and the cloud as well. The Hue lamps package consist of low consumption white or multi colored LED lamps, a ZigBee IP Bridge and smart controllers. The bridge is the "command center" of the Hue lighting system since all the requests to the lamps have to go through the bridge. The bridge is plugged into the router and it gets located by the Philips Servers. The lamps are registered

Figure 2.5. Philips Hue lamps

on the bridge so they can be controlled from there. After the setup is successful the user can interact with his lamps from the online application that Philips provides on their website or he can download the Hue application for the smart phone which controls the lamps through the home WiFi. Each bridge, as an advised limit, can control up to 50 lamps which should be in range of the bridge but according to users even more can be controlled by the same bridge. The starter packets of Philips Hue Lighting System contain of a bridge and two or three lamps and the price depends on whether the lamps are colored or white. After the purchase of a starter pack the user has the ability to expand his Lighting System by purchasing and adding more lamps.[30]

2.3.1 Hue Lamps Developer API

Philips provides an API for Hue Lighting System developers. The API is used to program and perform actions on the lamps, switch them on/off, blink them, change their colors etc. Through this local API the developers can create custom applications that will use their lamps through their local home network. The ZigBee IP Bridge contains a button that should be pressed by the user in order to authenticate himself as someone who is physically located close to the bridge. After authenticating successfully, the user acquires the bridge token needed to use the lamps. The Hue lamps Developers API has been used in the project in order to control the lamps through the fog node [29].

2.3.2 Phue API

Phue is free python library that is used to control and program the Philips Hue lamps and it is available on GitHub. Phue runs on a computer that is on the same local network as the ZigBee IP Bridge. Initially, the user has to provide the internal IP of the bridge in order to establish a connection. Simultaneously, the user has to press the button on the bridge to acquire the token if connecting for the first time, in order for the token to be generated and saved. After the acquisition of the token the library commands can be used to interact with the lamps through the local network. The library can be used with both Python 2 and Python 3 and contains features as groups, schedulers and sensors.[33]

2.4 LIFX Bulbs

LIFX Bulbs are a wireless lighting system made by LIFX with similar functionality as the Philips Hue lamps. LIFX Bulbs are screwed into all kind of lighting fixtures like normal lamps and they are controlled through the smartphone. Instead of a bridge, every lamp has its own wifi connection with the router and there is no central control device like in the Philips Hue lamps. In order to set up the lamps the user must download the LIFX application for Android/iOS. While performing the set up there were some problems that required sometime to solve and troubleshoot. It was necessary to remove the encryption from the WiFi Hotspot since it was unable to establish a connection with the encryption activated, which is a serious security risk. The process to connect the lamp to the cloud was also not entirely - straightforward as it was with the Philips Hue lamps.[14]

2.5 IFTTT Channels

IFTTT provides a variety of channels - services where the users can use through their accounts on these service providers as for example Facebook, Gmail, littleBits, Philips Hue etc. IFTTT channels are channels to devices, social media or Internet applications. Each channel has its own events and parameters that can be used either as triggers or actions events in applets. In order to use an IFTTT channel, the user must connect his existing account on that channel - service to give permissions to IFTTT. Some channels can be used only as triggers (ex. bttn, Date & Time, Weather Underground), some only as actions (ex. Hue lamps, LIFX) and some channels as both triggers and actions (ex. littleBits, IFTTT Maker). Every channel based on its nature and the information that their API with IFTTT makes available, can include different events and parameters in the applets. For example if Gmail is set as a trigger, any *new email in the inbox (event)* from a *certain email address (event parameters)* could be set as a trigger event, while if Gmail is set as an action event to a trigger as *receiving input signal from the cloudBit*, it can *send an email (event)* to a *specific email address with a specific subject, body etc (event parameters)*. These parameters are passed from the trigger side of the applet to the action side and affect the functionality. The polling time between channels varies according to the nature of the channel. A button that is used to trigger some action as a switch has an almost real - time polling interval, while reading information from a cloud based storage file has a much more infrequent polling time. According to IFTTT, the highest possible polling time that can be expected to encounter in a service is 15 minutes.

2.5.1 LittleBits Channel

LittleBits have an official channel (partner) on IFTTT. The device that interacts with IFTTT is the cloudBit which can be used both as a trigger and as an action. As a trigger device, it can be used to trigger the applet when it receives a signal or when it stops receiving a signal from the red bits. As an action device, it can be used to activate an output, deactivate an output or set a certain output level from 0-5v (0-100%) for a certain amount of time on the green bits. All the options available for the littleBits Channel are the following:

Triggers

Event	Description	Parameters
Turned On	This Trigger fires every time your cloudBit receives an input signal from another littleBits module.	Which cloudBit?
Turned Off	This Trigger fires every time your cloudBit's input signal goes from high to low.	Which cloudBit?

Actions

Event	Description	Parameters
Activate Output	This Trigger fires every time your cloudBit receives an input signal from another littleBits module.	Which cloudBit?
Deactivate Output	This Trigger fires every time your cloudBit's input signal goes from high to low.	Which cloudBit?
Set Output Level	This Action will set the output of your cloudBit to the specified level for the specified length of time.	Which cloudBit? Level (0-100%) Duration

2.5.2 IFTTT Maker Channel

The IFTTT Maker Channel provides developers the opportunity to create custom triggers and actions events in the form of web requests. Using only the predefined partner channels would be too restricting for developers since they would have little control and flexibility to customize or expand their applications. IFTTT Maker channel can be used both as a trigger and as an action channel.

When an IFTTT user starts using the IFTTT Maker channel, a unique token is assigned to his account that has to be used in every web request that is being sent to IFTTT as a trigger. Every trigger event of the IFTTT Maker channel has to have an event name - keyword which must be also specified on the web request URL alongside the token. When IFTTT receives this information of the web request it can distinguish which event should be fired based on the token and the keyword. It is also possible for the developers to attach a payload of three values to the web requests using the IFTTT Maker channel which can be used to affect the action.

The IFTTT Maker channel can also be used as an action by sending a web request to a user - defined IP address. The user can specify a URL, a method (GET, POST, PUT, DELETE), the content type and body of the web request he wishes to send. In the following sections which describe the latency measurements, the IFTTT Maker channel has been used both as a trigger and action. In order to be able to receive these web requests from IFTTT port forwarding has to be enabled on the router.

More specifically the options that are offered by IFTTT Maker channel are the following:

Triggers

Event	Description	Parameters
Receive a web request	This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile).	Event Name

Actions

Event	Description	Parameters
Make a web request	This action will make a web request to a publicly accessible URL.	URL Method Content Type Body

2.5.3 Hue Channel

The Hue channel on IFTTT is the official channel of the Philips Hue lamps and provides an API for creating applets that include the Hue lamps. Through the Hue channel, users can interact with the lamps through other IFTTT connected channels - devices. This channel contains only actions. The user should first connect his Hue account on IFTTT and then can begin using the applets. When a new applet is created the user can choose what actions should be fired and which lamps should perform that action. Not all the actions are available for all lamps since some are only white lamps and can not use the color features. The user can choose from the following actions:

No Triggers
Actions

Event	Description	Parameters
Set a scene in a room	This Action well set a scene on your hue lights.	Which scene?
Turn on lights	This Action will turn on your hue lights.	Which lights?
Turn off lights	This Action will turn off your hue lights.	Which lights?
Toggle lights on/off	This Action will toggle your hue lights on or off.	Which lights?
Blink lights	This Action will briefly turn your hue lights off then back on.	Which lights?
Dim lights	This Action will dim or brighten your hue lights to a value between 0-100.	Which lights? Brightness
Change color	This Action will change the color of your hue lights.	Which lights? Color value or name
Change to random color	This Action will change your hue lights to a randomly selected color.	Which lights?
Change color from image	This Action will change the color of your hue lights to match the dominant colors found in an image you specify.	Which lights? Color mode Image URL
Turn on color loop	This Action will turn on a slow color loop effect for your hue lights.	Which lights?

2.5.4 LIFX Channel

The LIFX Channel on IFTTT gives the user the ability to connect and interact with their LIFX Bulbs through IFTTT. It contains only actions as it happens with the Hue channel. The user has to connect his bulbs on the cloud and then connect his LIFX account to IFTTT to authorize access to the bulbs. The actions and the parameters that can be used in an applet are the following:

No Triggers

Actions

Event	Description	Parameters
Turn lights on	This Action will turn your lights on.	Which lights? Fade in duration Color Brightness Advanced options
Turn lights off	This Action will turn your lights off.	Which lights? Fade out duration Advanced options
Toggle lights on/off	This Action will turn your lights off if they are on, and turn them on if they are off.	Which lights? Advanced options
Activate scene	This Action will activate a Scene.	Scene Transition duration Advanced options
Change color of lights	This Action will change the color of your lights.	Which lights? Color Brightness Turn on first? Transition duration Advanced options
Blink lights	This Action will make your lights quickly blink the color of your choice.	Which lights? Turn on first? Number of blinks Turn on first? Color Brightness Advanced options
Breathe lights	This Action will make your lights slowly breathe the color of your choice.	Which lights? Turn on first? Number of breaths Turn on first? Color Brightness Advanced options

2.5.5 IFTTT Platform (with fee)

IFTTT Platform is a premium service that allows partners to publish their services through IFTTT. The existing IFTTT channels are created by IFTTT or by IFTTT partners. Becoming a partner and publishing an API with IFTTT requires an annual fee and possible cooperation between IFTTT and the customer. Most of the existing channels are referred to well known service providers since the cost to become a partner can be as high as 499 US Dollars per month, billed annually. There are two partner packets subscriptions available, the first one, the simple partner packet costs 199 US Dollars per month, billed annually and consists of the following 5 services:

1. Publish one service
2. Access in-depth analytics
3. Unlimited Users
4. Unlimited Applets
5. Embed Applets in your app and website

The second packet, the partner Plus packet costs 499 US Dollars per month, billed annually but IFTTT does not provide any information about the services that user will be provided with and it requires to fill in a "get in touch" form for more information.

2.5.6 IFTTT Maker Platform (without fee)

During the end of May 2017, IFTTT launched a beta version of platform for IFTTT Makers. The users have to request IFTTT for access to platform so they can create platform applets. IFTTT platform provides the users the same events and parameters for the provided channels as in the regular applets, but it allows users to add extra features for extra functionality and for having better control over them. When using the platform, the user can create custom made private applets (only for his account) with all the existing channels and events available, which he can publish on ifttt.com if he wishes to make them public to other users as well. On top of the regular applet features, platform applets can use the following features as well:

1. **One trigger - multiple actions:**
   One of the challenges we had to face before the launch of the platform was that every time a single event needed to trigger multiple actions, a chain of IFTTT Maker events was needed to be implemented since applets have this one trigger - to - one action relationship. Python threads had to sniff the incoming traffic and fire new IFTTT Maker events when the preselected body was found in the incoming request. Platform simplifies this problem by just adding multiple actions for the same trigger.
2. **Fields that can - cannot be modified in an applet:**
   Each applet, based on the action channel it uses it gives the maker the option to personalize some the fields that are about to be used. For example when using the Hue channel as the action channel

the user can select *which lights* will be affected and if there is an extra field, like *brightness level* or *color code*, the user can also select the value in that field. In case the IFTTT Maker channel is used as the action channel, the user can specify the *url of the web request*, the *method of the request (GET, POST, PUT, DELETE)*, the *content type* and the *body*.
When a maker creates an applet on platform, he can select which of the above mentioned fields can be accessible to the user.
In the case of the Philips Hue, the maker can select if the extra field will be accessible to the user or if the value there will be fixed - locked. The *which lights* field can not be locked since he can not know which are the lights of that future-to-be user.
In the case of IFTTT Maker channel, the maker can lock all of the fields or he can allow the user to be able to personalize some of them. For example the maker can predefine a URL, a content type and the method and allow the user to only customize the body.

3. **Filter Code:**

   The third feature that is being introduced in the platform is that the maker can actually insert some logic in the applet which can be used combined with the payload that is attached on the requests. The logic is inserted between the trigger and the action(s) and uses the payload of the request to IFTTT as trigger data. In the case of littleBits, this payload is fixed as the *device name*, *power percent* and *turned on at* while on the IFTTT Maker channel the user can specify 3 custom variables as well. The maker can use these variables to either modify the content of the output or skip the predefined action entirely. With the addition of the filter code in the applet we can also tackle a more general problem when combining cloud and fog computing, the problem of having double requests, one from the cloud and one from the fog. By specifying a certain payload we can instruct IFTTT to drop the request if the fog node is able to perform the request.

3. Related Work

Distributed Systems
Distributed systems consists of different computers which work together to appear as a single computer[3]. However, they are complicated to design and there are various distributed systems architectures that are used according to the nature of the applications that will be served.

Data-centered architectures evolve around the idea that processes communicate through a common (passive or active) repository[3]. It resembles cloud servers which store information that can be accessed by multiple users and processes which can also subscribe to certain repositories and get notifications when the content is updated.

Communication is at the heart of distributed systems and it can be synchronous or asynchronous[3]. Asynchronous communication is easy to implement but there can be some delay between the request and the response. Synchronous communication on the other hand is more complicated to achieve but the communication happens in real time. In real time IoT applications communication has to be synchronous since the responses should be immediate.

A class of distributed systems are the edge - server systems. The servers are placed on the edge of the network between the ISP and the enterprise - home network[3]. Edge servers can provide filtering functions and optimize the content and application distribution.

Latency
Latency to cloud environments has been a known issue. Smart devices need instantaneous feedback to make decisions and people would not use these devices if they are not responsive. High latency can turn IoT applications non functional or even dangerous[26].

Cloud based games is also a latency sensitive category of applications. Kämäräinen et al. has measured the latency of cloud based games on smartphones that use either WiFi or LTE. In case of wireless connections the response time heavily depends on the signal strength. For a signal strength up until -100 dbm, the network delay to the first pingable IP address was stable and was not exceeding 40 ms while for a signal strength of -120 dbm the network delay was measured at 200 ms.[21] Moreover, the geographical location of the data center can affect latency and it is not possible that a user will always be served by the closest data center.

Amazon, Microsoft, Google and Rackspace are four well known cloud service providers. Measurements have indicated that latency varies between these cloud providers. A measurement for three of the providers revealed that response times for downloading a 1K Blob, for two of the three providers does not exceed 50 ms in 90% of the samples, while for the third it can go up to 100 ms. Uploading a 1K Blob, for two of the providers in most cases does not exceed 100 ms, while for the third provider on 95% of the samples exceeds 150ms. In the same measurement, using 10 Mb Blobs instead, downloading and uploading time can exceed 2000 and 3000ms respectively [22]. Li et al. has measured the time required to send and retrieve messages from a queue and the propagation delay of a queue for two of the providers. The message size is set at 50B. The results show that there is a lot of variation in the response times. For these two providers, the response times to send a message to the queue are above 50 ms for 60% and 45% of the samples and below 100 ms for 65% and 85% of the samples respectively. The response times to retrieve a message from the queue are above 50 ms for 40% and 70% of the samples and below 100 ms for 80% and 80% of the samples respectively. The queue propagation delay is below 200 ms for 80% of the samples for one provider and for 95% of the samples for the other [22].

Migration

Virtualization is a vital technology in both fog and cloud computing that enables virtual machines (VMs) to coexist in a physical server (host) to share resources[28]. Migrating services that are encapsulated in VMs is a way to ensure good performance in applications [24]. Several VM migration techniques are proposed. One technique is migrating a VM from one physical host to another. Another category of migration techniques are cold, hot and live. Cold and hot migration techniques both interrupt the service while live migration guarantees continuous service while moving the VM from one physical host to another. The disadvantage of live migration is the intensive consumption of resources. Live migration can be further separated to pre copy and post copy live migrations. The former involves the transfer of memory contents of the VM from a source to a target through several iterations before the VM is restarted; whilst the latter only sends the virtual central processing unit (vCPU) and the device state to the target at an initial stage. Pre copy is the predominant approach used in Xen, VMware and KVM hypervisors[28].

Migrating services to mobile edge clouds (MECs) is an approach to serve applications that require low latency. Machen et al. focuse only on live migration techniques. Service migration can be separated to stateless and stateful; stateless refer to a migration that simply redirects the requests to a new server with a separate instance of a server running. However, stateful migrations, which migrate the running states of the applications, are the focus of this paper since they meet the needs of modern applications. Migrations can result in service interruption and communication and computation resource overheads. Service applications are often encapsulated in self sufficient and pre configured environments for easy distribution. VMs and containers are examples of these environments. VMs fully emulate the OS kernel and hardware, whereas containers directly share the hardware and kernel with their host machines. Containers require less resources than VMs but they are less adaptable [24].

A layered migration framework is presented for both VMs and containers. When the base layer (the whole server) is migrated, a process during which the service is down, the average migration time is 25 seconds for a container and 160 seconds for a VM when using an 100 Mbps connection [24]. By dividing the "package" in layers and having a generic base layer pre installed in every MEC, the migration time can be reduced by having only to migrate the application layer on the 2-layer approach and the application and instance layer on the 3-layer approach. Machen et al. measure the performance of the layered framework for a variety of applications. The host VMs are given 2 virtual CPU cores and 2 GB of virtual memory each, from a 2.6 GHz Inter Core i7 physical machine with 16 GB 1600 MHz DDR3 memory. The connection between the MECs and the user was configured with 100 Mbps, 25 ms latency and 5 ms jitter. From the tested applications, the less demanding one was a Game Server with 0.7 Mb installation footprint and 1 Mb of memory requirement while the most demanding one was a Face Detection application that processes an incoming video stream. Its installation footprint is 655 Mb for LXC (container) and 565 Mb for KVM (Virtual Machine) and the memory requirement is 100 Mb. Additionally, a guest OS with no applications was also installed to represent the minimum bound of the migration [24].
For the LXC container, the guest OS without any application required 6.5 seconds for a total data transfer of 1.4 Mb for the 2-layer approach and 11.0 seconds for a total data transfer of 1.9 Mb for the 3-layer approach. The downtime for both approaches were 2 seconds. The Face Detection application required 52.0 seconds for a total data transfer of 363.1 Mb for the 2-layer approach and 70.1 seconds for a total data transfer of 365.0 Mb for the 3-layer approach. The downtime for both approaches were 47.6 and 3.7 seconds respectively[24].
For the KVM, the guest OS without any application required 81.5 seconds for a total data transfer of 65.3 Mb for the 2-layer approach and

141.8 seconds for a total data transfer of 65.9 Mb for the 3-layer approach. The downtime for both approaches were 55.8 and 56.1 seconds respectively. The Face Detection application required 381.2 seconds for a total data transfer of 1027.3 Mb for the 2-layer approach and 558.3 seconds for a total data transfer of 1033.7 Mb for the 3-layer approach. The downtime for both approaches were 355.0 and 107.5 seconds respectively[24].

Fog Computing
Networks failures can occur and make cloud servers unavailable [6]. However, by deploying fog nodes / cloudlets[7] this unavailability can be masked by working with a local copy of a deployed fog node for a certain amount of time [32].

Fog Computing products offer cloud computing capabilities at the edge of the network with low latencies for IoT applications [19] .

Fog or Edge computing is a model where deployed devices transmit data to nearby computing devices that process the data instead of sending them directly to a data center. Applications that use this model can achieve near real time analysis of the data. This technology tied with the IoT is gaining popularity and estimations predict that by 2020 5.6 billion IoT devices will utilize fog computing [20].

3.1 Role of Fog in IoT

Fog computing can achieve low latency since fog devices and mobile devices / sensors are physically located close to each other[32]. Whether they are connected via cable or via wireless connection the proximity and small number of hops can guarantee low latency.

Fog computing can contribute in scaling and filtering the size of the data that is uploaded to the cloud. Raw data are filtered and processed locally on the fog devices which after processing the data, can extract only the necessary / useful information. This information is uploaded on the cloud, thus saving resources from uploading all the raw data in the first place[32]. This filtering process can also preserve the privacy, since the user can adjust the policies of which data will be sent to the cloud and under which circumstances.

4. Latency in IoT

High latency in IoT applications is a serious bottleneck that can affect performance and functionality. Users would not be willing to use a smart lock for their house which would need several seconds to open or even get locked out if the Internet connection is off.

The first task of the project is to measure the latency of the initial application. This is necessary in order to understand the problem better and to have a benchmark to compare the initial and the improved functionality. After measuring the overall latency we will investigate the latency part-by-part in all the involved entities to locate how much latency each entity introduces.

4.1 Architecture

The architecture of the original application consists of different entities which forward the request from the cloudBit to the Hue lamps. It combines three different cloud services which are triggered serially when the cloudBit is activated. Initially the cloudBit is triggered by the buttonBit and sends a request to littleBits cloud control. Cloud control forwards the request to IFTTT which realizes that the cloudBit is registered in an applet as a trigger for the Hue lamps. IFTTT then contacts the Hue cloud server which locates the ZigBee IP Bridge and performs the action on the lamps. As it can be seen there are many entities that should collaborate in order to perform the task successfully and each one's reaction time appends to the final delay. Figure 4.1 illustrates the architecture.

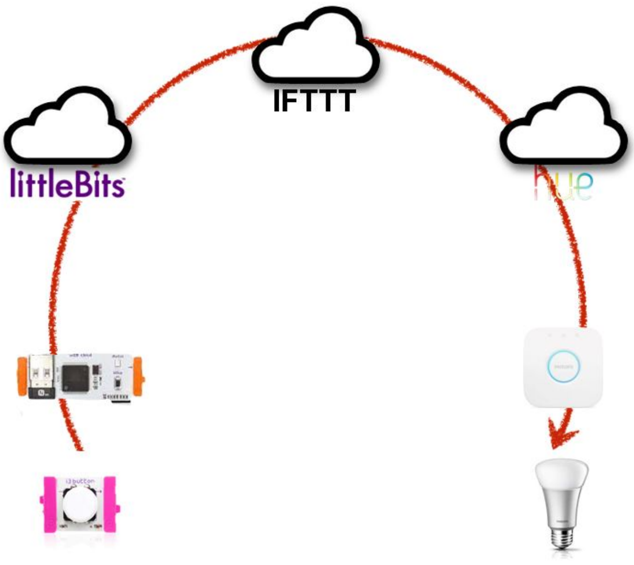


Figure 4.1. Application Architecture

4.2 Measurements

Estimating the latency with an accurate and scientifically implemented method was the first task of the project. The latency measurement should begin when the buttonBit is clicked and stop when the light gets bright. The measuring device should include a light sensor so it can sense when the lights are getting bright but at the same time the device should also "sense" when the request was sent from littleBits. The fact that the buttonBit (trigger) is a separate device than the light sensor introduces the problem of not being able to have the same timer for the measurement. Thus, it is necessary to find a way so the measurement is performed by the same device - timer since the timestamps of two different devices can deviate. Unfortunately there is no trivial way to perform this measurement so we had to build a custom made measuring prototype. This prototype consists of littleBits and TI CC2650 Sensortag, which contains a light sensor. TI CC2650 is a wireless MCU that runs Contiki OS. The two devices are connected through wires between littleBits protoBit and TI CC2650 Sensortag's GPIO Pins. ProtoBit is a littleBits orange bit that allows wiring littleBits with external devices. By having protoBit after a buttonBit (or any other red bit), it is possible that external devices will "sense" the signal that was produced from the buttonBit while at the same time bits that follow the protoBit (in our case a cloudBit) will still receive the signal. Alternative solutions must use the same logic, having the same device sensing the request and the increase in the luminance, but could use different devices. More details on the measurements are provided on the *How we measure* section.

In order to execute the measurement we should first implement the following tasks:

4.2.1 Initial Measurement Setup

- Make an IFTTT applet that has the cloudBit as a trigger and Hue lamps as an action. The applet should turn on the lamps when there is a signal from the cloudBit.
- Connect protoBit with TI CC2650 GPIO Pins

The technical details of the setup are described here 6.4

Contiki OS
Contiki is an open source RTOS developed by Adam Dunkels et al. at the Swedish Institute of Computer Science. The operating system is written in C and uses an event-driven kernel. Contiki was developed for the Internet of Things and offers official support for Texas Instruments MSP430 and Atmel AVR. In this project Contiki OS was used since we needed TI CC2650 Sensortag to perform the latency measurements [9].

TI CC2650 Sensortag
TI CC2650 Sensortag [17] is a device made by Texas Instruments inc. It is a wireless MCU which contains multiple sensors which can be used to get measurements. TI CC2650 Sensortag does not include a USB port but it can be connected with the Debugger / Devpack through JTAG which contains a USB port. Through the Debugger / Devpack it is possible to flash software to the TI CC2650 Sensortag and get serial output on the terminal. TI CC2650 Sensortag can run Contki OS and contains the following sensors that can be used with Contiki:

1. Temperature Sensor
2. Optical / Light Sensor
3. Humidity Sensor
4. Gyroscope
5. Pressure

Light Sensor
TI CC2650 Sensortag contains a light sensor that can read and print out light values based on the luminance around it. The light sensor on the TI CC2650 Sensortag is named opt3001. Example code on how to use it with Contiki OS can be found on the examples/cc26xx directory of the

Contiki OS. The light sensor is used so we can sense the increase in the luminance, thus measuring the latency.

4.2.2 How we measure

In order to measure the latency between the littleBits and the Hue lamps we log the timestamps on the TI CC2650 Sensortag. The application (Figure 4.2) that we performed the measurements consists of littleBits, an IFTTT applet, TI CC2650 Sensortag and Hue lamps. As was described above, after connecting it with littleBits, TI CC2650 Sensortag can sense both the request from littleBits and the increase of the luminance so the measurement is performed by the same timer. TI CC2650 Sensortag receives a signal when the request towards IFTTT is sent from littleBits, and calculates the time from that moment until its light sensor senses that the nearby Hue lamp gets bright. In particular the littleBits application consists of a powerBit to provide the power supply, a buttonBit as a trigger, a protoBit connected to the TI CC2650 Sensortag's GPIO Pins and the cloudBit.

The input signal that comes from the buttonBit triggers an input in the cloudBit which is registered on the trigger side of the IFTTT applet. The Hue lamp is on the action side of the IFTTT applet and after the button triggers the applet, IFTTT requests through their online API with Philips to switch on the lamps. Philips server has to reach the ZigBee IP Bridge which will contact the lamps as soon as it receives the request and switch them on. Every time the button is pressed the protoBit sends signal to the TI CC2650 Sensortag, at the same time the cloudBit gets the signal as well. When TI CC2650 Sensortag senses an input signal, it records the time and activates the light sensor. Then the light sensor waits until the lamp gets bright and the luminance value exceeds a certain threshold. TI CC2650 Sensortag is placed directly below the lamp so it will notice instantly the increasing luminance. When that happens the latency is calculated by subtracting the initial time recorded from the current time. Performing this experiment multiple times gives us an average latency time for the application.

More technical details can be found on appendix 6.4

At this point it should be mentioned that the measurement of the Philips Hue lamps has been done using the old firmware of the ZigBee IP Bridge which is introducing latencies above 20 seconds on average. According to developers forum the latency improved significantly when the firmware was updated but we were not able to perform any measurements with the new firmware because IFTTT banned our bridge after an extensive automated five day experiment, probably because they realized it was used by a 'bot'. Because of the ban, after

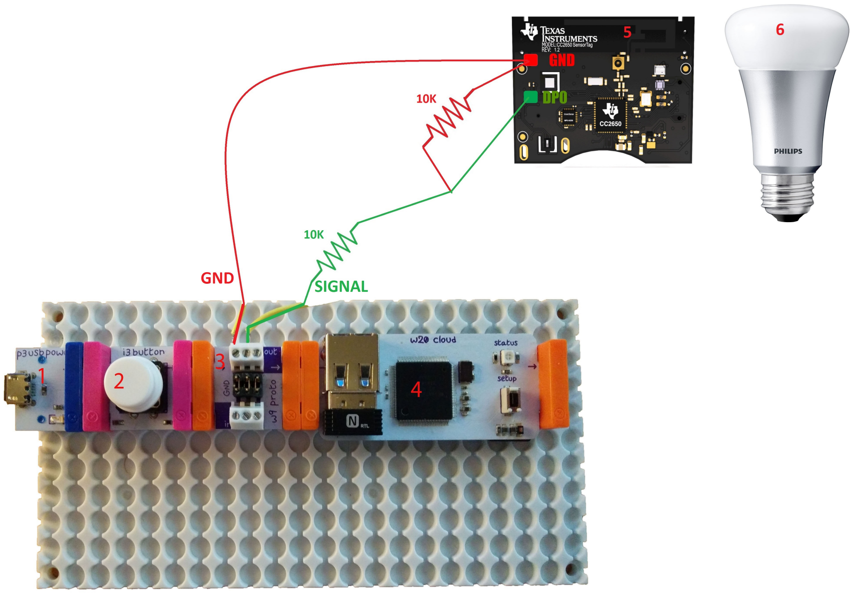


Figure 4.2. 1. powerBit, 2. buttonBit, 3. protoBit, 4. cloudBit, 5. TI CC2650 Sensortag, 6. Hue lamp

that point it was not possible to perform any more Hue lamps measurements through IFTTT. Although by observing with naked eye the performance on another bridge with the new firmware it seems that the latency has indeed improved to times between 3-5 seconds but this can not be stated as a fact since it has not been measured scientifically. The same methodology and devices (Figure 4.2) have been used to measure the latency between littleBits and LIFX Bulb.

4.2.3 Additional Experiments

As was described in the architecture, there are three different cloud services forwarding the request from the littleBits to the Hue lamps. However, by calculating only the overall latency, is not possible to locate which entity (device(s) or cloud server(s)) introduces the highest part of that latency. For example, if the latency is at 15 seconds overall, it is critical to find out if this latency is distributed equally on every entity or only on one, which can then be detoured. In order to measure how much latency each entity introduces and calculate the part-by-part latency we have to make some modifications to the measuring process and try to measure the latency in each entity individually. Thus, we should have different applets running simultaneously, one responsible for every entity and a way to trigger them all together. This is due to that before the introduction of the IFTTT platform we could not have a single trigger firing multiple actions, so we had to find a way to trigger different applets simultaneously. IFTTT Maker channel helps to estimate latency on the involved entities because it gives us the ability to send and receive something to IFTTT directly without having to go through a IFTTT partner's API of which we have no control. In this way we are breaking the measurements in smaller pieces and record the latency from one entity to the next one.

Setup

Some more tasks had to be implemented for the additional experiments (part by part measurement):

- Make three different applets on IFTTT, one with littleBits as trigger and IFTTT Maker as action, one with IFTTT Maker as both trigger and action and one with IFTTT Maker as trigger and Philips Hue lamps as action.
- Implement port forwarding on the home router.
- Download Python library scapy to sniff the incoming requests.

4.2.4 Latency part by part

Since between the littleBits and the Hue lamps are many entities involved we tried to divide these measurements in parts and perform the measurements in those parts individually so we can have a better understanding on how the latency is distributed. We could not trigger three actions simultaneously through cloudBit from IFTTT because IFTTT platform was not active at the moment, thus we were restricted in this *one trigger for one action* relationship for the IFTTT involved applets. With that in mind and since there are three cloud services that we want to measure, we made three different applets, one for each cloud

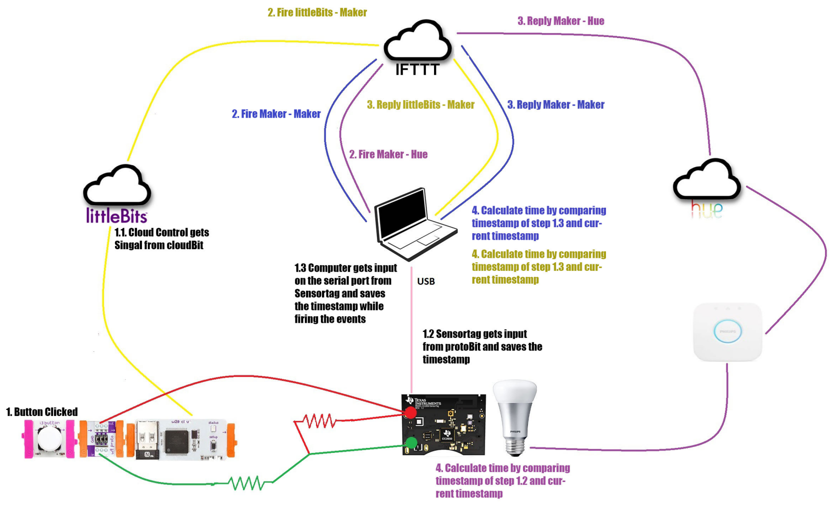

2. Fire littleBits - Maker
3. Reply Maker - Hue
IFTTT
2. Fire Maker - Maker
3. Reply littleBits - Maker
3. Reply Maker - Maker
2. Fire Maker - Hue
littleBits
1.1. Cloud Control gets Singal from cloudBit
4. Calculate time by comparing timestamp of step 1.3 and current timestamp
4. Calculate time by comparing timestamp of step 1.3 and current timestamp
hue
1.3 Computer gets input on the serial port from Sensortag and saves the timestamp while firing the events
USB
1.2 Sensortag gets input from protoBit and saves the timestamp
1. Button Clicked
4. Calculate time by comparing timestamp of step 1.2 and current timestamp

Figure 4.3. Part by part measurement

service. The main principle we used to design these measurements, since we can not measure through the IFTTT APIs directly, was that if **A -> B** needs **x** seconds and **A -> C** needs **y** seconds, although we can not directly measure **B -> C**, we can estimate **B -> C** by subtracting **x** from **y**. For this experiment we have also modified the code on TI CC2650 Sensortag so when the sensor has an input from protoBit, it will send an output to the computer through the serial port so the computer can trigger the applets immediately. One applet is measuring from littleBits cloud service to IFTTT actions side, one from IFTTT trigger side to IFTTT action side and one from IFTTT trigger side to Hue lamps. Figure 4.3 helps to visualize the logic of these measurements. The important thing to understand in this figure is that after the first step, clicking the buttonBit, 1.1, 1.2 and 1.3 are happening almost simultaneously because when the sensor notices the signal from protoBit notifies directly the computer.

The first step is to measure how much time is required for littleBits requests to be received by IFTTT. Unfortunately, it is not possible to know when a request arrives in IFTTT in order to be processed and fire an action. The time required from when clicking the buttonBit on the littleBits application until the signal gets received on the littleBits cloud control application is really low but there is no way to be measured scientifically either, since the API is too restrictive. With a naked eye observation, cloudBit to cloud control latency is approximately 0.2 - 0.3 seconds. This observation can be supported by the measurements which prove that the main part of the latency is not introduced by littleBits cloud control. Although, by measuring the time required to send a littleBits request to IFTTT and get back a response through IFTTT Maker channel and by knowing from the next step, the latency between an IFTTT Maker trigger and an IFTTT Maker action (approximately how much time IFTTT needs to process a request), we can subtract these numbers and estimate how much time the littleBits request needs to reach IFTTT.

The next step is to measure how much time it takes for IFTTT to fire an action after receiving a request. By using the predefined channels that is not possible so IFTTT Maker channel, which gives the user the ability to create custom triggers and actions in the form of web requests has to be used. The user uses a token-string and an event keyword which are both used in the URL to trigger an event just by calling this URL. On the action part, a URL, which the IFTTT request will use should also be specified, though this part is more tricky since the user should perform port forwarding in his NAT router in order to receive the web request from IFTTT and run a sniffer to record the packet's arrival time. After that is done successfully there can be an estimation for the latency that IFTTT introduces.

The final step would be to send a request directly to Hue lamps online API and measure how much time it would take for the lamps to switch on, but this is not possible, since Philips has not yet released an official online API for developers. What can be done is to send a request to the Hue lamps through IFTTT Maker channel, and by knowing from the previous step how much time is required by IFTTT to process an IFTTT Maker request, we can estimate the Hue latency by subtracting these times.

For this measurement we must make sure that the three applets are triggered simultaneously. As in the initial measurement for the overall latency, we used the same application (Figure 4.2) that was used for the initial measurement. More technical details on how we start the three threads simultaneously can be found here 6.4.

The three applets that have been used for the current measurements are the following:

1. **IFTTT Maker - Philips Hue**
   This applet measures how much time is required to send a web request to IFTTT Maker (trigger side) and get an action from the Hue lamps through the online API. When the first thread starts, it sends a web request to IFTTT Maker channel that will switch the lamps on. This web request was initiated when the computer got notified through the serial port that TI CC2650 Sensortag had an input signal. The TI CC2650 Sensortag as in the overall latency measurement will notice the increase of the luminance and having already a timestamp from when it received input signal, it will calculate the difference between the two timestamps. By comparing this

measurement with the overall latency we can get an estimation if it is faster to trigger something through the littleBits channel or through the IFTTT Maker channel.

2. **LittleBits - IFTTT Maker**
   This applet measures how much time is required to send a request from the cloudBit and receive a web request on the computer from IFTTT Maker (action side). When the second thread starts, it runs a sniffer on the ethernet port waiting a for an IFTTT response. This response is the action of an applet that has the cloudBit as a trigger and an IFTTT Maker web request to the computers IP as a action. This applet was triggered when the littleBits button before the protoBit was clicked and the signal reached cloudBit. This measurement is performed with the computer's local timestamp since we record when TI CC2650 Sensortag sends an output in the serial port (buttonBit was clicked) and when the response was received by the computer.
3. **IFTTT Maker - IFTTT Maker**
   This applet measures how much time is required to send a web request to IFTTT Maker (trigger side) and receive a web request on the computer from IFTTT Maker (action side). When the third thread starts, it sends a web request to IFTTT Maker channel that will trigger a web request back to the computer. This measurement is performed with the computer's local timestamp since we record when the web request to IFTTT was fired and when the response was received by the computer.

By comparing the times of the second and third applet, we were able to notice that the littleBits channel is faster than the IFTTT Maker channel, since both had IFTTT Maker channel as an output but different input channels. Instead of running all three applets together and complicate the process we could run the applets one by one but since the measurements would occur in different times it could be argued that the environment was not identical.

4.3 Size of Samples

In order to perform the latency measurements for the various forms of the cloud based applications we ran a series of experiments in a range of 1-3 hours with a frequency of approximately 1 measurement per 50 seconds.

4.4 Results

In this section we present the results of the measurements and reasoning about their validity. The following table shows six different measurements, the first one, *littleBits - Philips Hue* (original applet) was taken on a different day than the *IFTTT Maker - IFTTT Maker*, *IFTTT Maker - Hue* and *littleBits - IFTTT Maker* that were taken simultaneously and belong to the part-by-part latency measurements. The last two measurements are the measurements with the LIFX bulbs that took place on a different date than the above mentioned ones.

In all of the measurements the requests are forwarded through IFTTT. LBCC (LittleBits Cloud Control) is only used when littleBits are set on an applet as a trigger. Hue online server is only used when Philips Hue lamps are set as an action on an applet.

Results

Channels		*Cloud Servers Used*			*Seconds*			
Trigger	Action	LB CC	IFTTT	Hue	Latency	St dev	Max	Min
littleBits	Hue	✓	✓	✓	20.3	5.5	26.9	4.75
Maker	Maker	×	✓	×	3.9	1.9	6.9	2.1
Maker	Hue	×	✓	✓	21.8	3.6	29	6.5
littleBits	Maker	✓	✓	×	3.0	1.7	10.9	1.5
Maker	LIFX	×	✓	×	3.4	0.7	6.8	2.4
littleBits	LIFX	✓	✓	×	3.5	1.2	7.6	1.1

For the **original applet** the average latency was measured at 20.3 seconds with a standard deviation of 5.5 seconds. Figure 4.4 illustrates the latency in a period of approximately 1 hour. The values are fluctuating a lot and can be anywhere from 5 to 25 seconds. Given the inability to make continuous measurements while developing and testing the application, by printing out latency values without storing them, we were also getting latencies of the magnitude of 20 seconds. Therefore, and for reasons we will explain below, we consider the samples to be representative.

For the three simultaneously implemented measurements (part-by-part) we obtained much lower latencies, in the range of 3 - 3.9 seconds, when the action channel was IFTTT Maker. The results indicate that the trigger channel does not affect the output significantly, while when using Philips Hue as the action channel the latency exceeds 20 seconds on average.

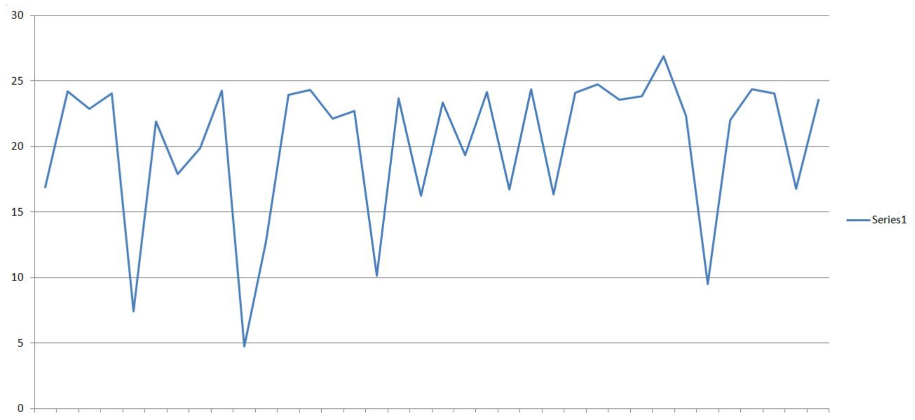
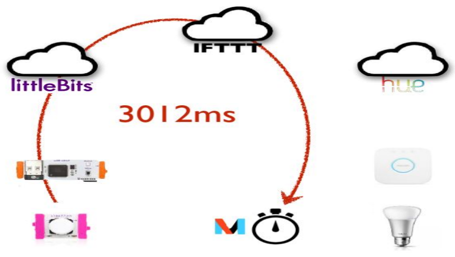


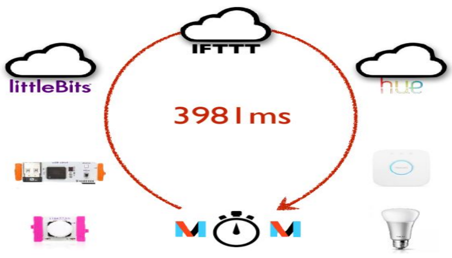
Figure 4.4. Latency from littleBits - Hue lamps (seconds)

The results of the part-by-part and LIFX measurements are supporting our conclusion that all measurements are representative. This is due to that average latencies for *IFTTT Maker - LIFX* and *littleBits - LIFX*, which as can been seen from the results table, use the same cloud services as *IFTTT Maker - IFTTT Maker* and *littleBits - IFTTT Maker* respectively, both introduce similar latencies in the range of 3 - 3.9 (3.4 and 3.5).

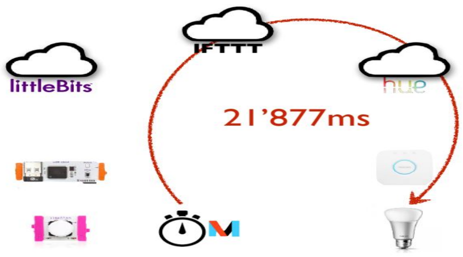
Overall, the results indicate that having littleBits or IFTTT Maker as triggers does not have a significant impact on the latency, which leads us to conclude that when Philips Hue online API is set as action the application consistently introduces the high latency. Furthermore, since the overall and the part-by-part measurements were taken on two different days and the average latency for the *original* and the *IFTTT Maker - Hue* applets are 20.3 and 21.8 seconds respectively, as well as Max and Min values varied a lot on both measurements, it proves that when an applet uses Hue online API the latencies exceed 20 seconds on average and fluctuate a lot as well. Finally, although *IFTTT Maker - IFTTT Maker* and *littleBits - IFTTT Maker* measurements were taken on a different day than the *IFTTT Maker - LIFX* and *littleBits - LIFX* measurements, the average latency is quite similar which strengthens the conclusion that the measurements are valid.



(a) Latency from **LittleBits** to **IFTTT Maker**



(b) Latency from **IFTTT Maker** to **IFTTT Maker**



(c) Latency from **IFTTT Maker** to **Philips Hue**

Figure 4.5. This is a figure caption

4.5 Conclusions

Cloud is an architecture that has changed the way people communicate. It simplifies tasks like connecting various devices and services from different manufacturers and provides a safe and effortless storage mechanism. The functionality that is provided by IFTTT would be impossible to achieve for some of the services while for some others it would require highly skilled users. Nowadays that Internet connections have high bandwidth and good area coverage, cloud is an excellent tool for performing everyday tasks that can tolerate some latency like sharing photos, collaborating on text editors and using certain IFTTT services as well.

Although, for applications that are not tolerant to latency and need to be constantly connected to the Internet, cloud approaches can have drawbacks. Even if the cloud infrastructure is state of the art and guarantees minimum latency, other parameters can also compromise performance. For example the reception of a mobile phone or another device might be bad, bad weather might cause problems or cables can be damaged due to various reasons, thus making the access to a cloud service impossible. A user that tries to unlock the smart lock of his garage through his phone might face the problem that his mobile phone does not have signal on a basement. In cases of emergency where networks are overloaded and infrastructures are damaged cloud based solutions might be unavailable. During our experiments there were two occasions where the Internet connection was down so we could not use any IFTTT applets at that time. With that in mind it is necessary to have an alternative when low latency is required and when due to extreme conditions with no Internet connection the functionality must remain intact.

Performing measurements that involve several devices and online servers can be tricky since all the devices should be connected with each other but only one of the devices should be responsible to generate the timestamps. Building a measuring prototype and performing these measurements requires both software and hardware skills. The maker has to improvise and the prototype that is created might not look entirely orthodox. For our experiments where we had to make the prototype that consisted of the TI CC2650 Sensortag and the littleBits, we ran into unexpected problems due to the different voltage of the two devices and the difficulty to read and write to the GPIO Pins. Developing a prototype like this is not a trivial task and can take more time and effort than expected but it can not be avoided. Kämäräinen et al. have also used various devices for a prototype that measures the latency of the cloud based game.

The online application introduces latencies of the magnitude of 20 seconds when using the old firmware on the ZigBee IP Bridge but the response time is almost instantaneous when using the local Hue API. According to our own naked eye observation and to comments of Hue lamps users, the latency of the online application is reduced significantly when using the new firmware. With that in mind we can speculate that the latency is located on the connection between the bridge and the Hue cloud server but we can not elaborate or prove this, since both the bridge and the cloud server are "black boxes".

Latencies of the magnitude of 20 seconds are restricting for performing any kind of almost real-time application. Though as was described above, in the original applet there are many entities involved so we performed the part-by-part measurement in order to identify how much latency each entity introduces. By observing the result in the part-by-part measurement it becomes obvious that we must detour the Hue online API as a first step to reduce latency.

It could be argued that with a 3 - 3.9 seconds latency, if we would use IFTTT Maker instead of Hue online server for switching on the lights, the applet could still be used satisfactorily enough. Although if someone looks at the big picture of IoT applications, in many cases latencies even of the magnitude of 3 seconds can not be tolerated, for example in an application that would have to do with navigation while driving a car.

5. Migrating to Fog

High latency and availability are some of the limitations that are introduced in cloud based IoT applications. In cases where the latency occurs either due to connection problems or due to problems with the servers, the applications functionality can be disrupted. Migrating IoT applications to fog devices by moving some of the functionality from the cloud servers down to the edge of the network, helps dealing with these limitations by performing some latency sensitive tasks locally on the fog nodes. Figure 5.1 illustrates the cloud entities that are being detoured after the fog node has fetched the functionality from the cloud server (IFTTT). Migrations to fog nodes are usually implemented by migrating VMs from the cloud servers. In our case, migrating a VM is not an option since we only have access through the webpage.

The fog node is the device in the local network that will scrape, store and - thus - migrate the online functionality from IFTTT so it will perform the latency sensitive tasks locally. In the current project the fog node is a RPi connected to the home router, the same router the ZigBee IP Bridge is connected to. RPi is a small sized and low priced computer that runs Linux OS. An arduinoBit of a littleBits application is connected to the RPi through a USB cable. The RPi receives the requests from littleBits in order to perform the predefined actions on the Hue lamps or to migrate IFTTT applets locally. Furthermore, the RPi runs a REST Server to perform local requests. Figure 5.2 illustrates the architecture of the fog node. On this chapter we will present how the migration on the fog node was performed, some features that were added to personalize the migration, a littleBits application that is used to demonstrate the migration and the results of the fog node measurements.

5.1 Login and Navigation in IFTTT

IFTTT does not provide an API for registered developers to get information about their running services. As a result the only way to get the online information from IFTTT is to use a script to login and navigate to ifttt.com, scraping all relevant data from the webpage. Python provides some libraries which can be used for that task. The three main libraries that have been used are mechanize [13], cookielib [1], and BeautifulSoup [31] [25].

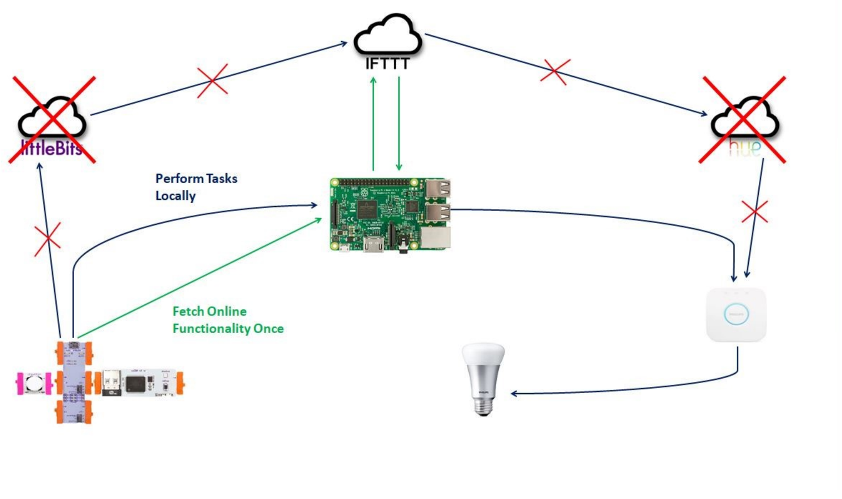


Figure 5.1. Migrating functionality to the fog node

5.1.1 Main Libraries

Mechanize is a library that is used to implement programmatic web browsing. After the login is successful, it can be used to navigate to pages further down the website hierarchy tree which require an active log in session in order to be accessible.

Cookielib library handles HTTP cookies on the client machine. Cookies are sent from the server through an HTTP response to the client and then are used by the client on later HTTP requests. Cookies help the servers remember information about the browser's previous activity. As an example cookies can be used to avoid log in every time a user enters a website, remember navigation choices etc.

BeautifulSoup library is used to perform the actual scraping. It provides the developer the ability to select various elements from the html content of the page. By using that library and scraping online information, it is possible to build an "offline" copy of the online functionality on the fog node.

5.1.2 Implementation

In order to have the IFTTT applets migrated on the RPi, a local copy of them should be created on the device. By navigating programmatically to the webpage we can scrape the relevant content and create a local copy. To perform the scraping there are three main tasks that should be implemented, Login to IFTTT through a script, locate all the applications that meet the migration criteria and then scrape their content.

1. **Login**:
   Initially the script has to login the user into his account on IFTTT. The script has the IFTTT login page hardcoded and the user has to provide his credentials on the script. *Mechanize* detects the login form fields by their names and inserts the corresponding user credentials. IFTTT uses a mechanism called *authenticity_token* to prevent attacks that submit data without an active login session. This mechanism is "invisible" when the user logins through the browser since the browser takes care of the *authenticity_token* and submits it along with the user credentials, but since we are trying to login through a script the token is not automatically submitted. The *authenticity_token* is created on every new page that is being requested from IFTTT and it should be submitted alongside with the credentials on the POST request of the login form. *Mechanize* provides some functions that locate the *authenticity_token* and attach it on the POST request. Without a library handling the *authenticity_token* login would not be possible. Next step is submitting the form and now the user can login. The browser object that is created by *mechanize* uses *cookielib* to store the cookie. From now on this browser object can navigate to pages of IFTTT that require authentication since it it contains an active cookie.
2. **Locating applets to migrate**:
   After logging in successfully, the script uses the *mechanize* browser object that was created on the login to access the applet pages. The script navigates to *My Applets* page which contains all the existing applets of the current user. On that page lies an overview of all the existing applets, their trigger channel, their action channel, whether they are active or not and the link that redirects to the configuration page of every applet where all the details of that applet are stored. The script scrapes *My Applets* page and through *Beautiful Soup* it filters which applets need to be scraped based on the channels that they are using. For every applet that satisfies the filtering criteria - being active and using some of the selected predefined channels - the link to the configuration page is scraped and stored on a custom class object that represents an applet that is about to be migrated.
3. **Scraping applet information**:
   Next step is looping on those scraped links of the configuration pages to extract all the specific information for every applet. When a user opens a configuration page through the browser, the page loads with the preselected parameter values (preselected cloudBits, lamps, lamp brightness level if the applet modifies the bright-

ness of the lamps) that were chosen by the user before. These values should be used as the default values on the fog node for the migrated applets. Apart from the preselected values, the configuration page includes details about which is the applet id, other application parameters and the labels of the applets which will be used to extract keywords when matching the online functionality to the local one. This information is included in JSON objects that are scraped with the page. Every configuration page includes a new *authenticity_token* as well. Some of these services provide the user with a drop down list containing different options that they can select as trigger or action devices. For the original applet that we used, these elements on the drop down lists are the available cloudBits that the user has and a list of all the Hue lamps that are connected on the current user ZigBee IP Bridge. IFTTT retrieves this information to populate the drop down list by performing a GET request on the IFTTT servers. This URL contains the dynamically created *authenticity_token* for that current page, so it can not be called randomly and out of context. In order to be able to execute this GET request ourselves and retrieve the information, we should scrape the page and paste the *authenticity_token* on the URL. The response on this GET request is a JSON object with all the available devices for the services that are used on the current applet.

During the scraping of the applets all necessary information is fetched. This information is stored on the RPi in a JSON file so everything is migrated - stored locally. 6.4

In many cases, while the Python script is scraping information from IFTTT, an *HTTP 500 Internal Sever Error* occurs. In case that happens, the script restarts automatically until all data are scraped and stored successfully. *HTTP 500* [10] is a server-side error, meaning that there is an issue while requesting IFTTT for its content. Although it is a server error, the cause of the error might be in the user's end. The rapid amount of requesting data from IFTTT through *mechanize* browser might trigger some protection mechanisms to fire this error. Some of the suggested solutions are deleting the cookies - which in our case are necessary while navigating to applets pages - and clearing the cache which would not help either since every time we run the script, a new *mechanize* object is created.

5.2 Parse IFTTT on RPi

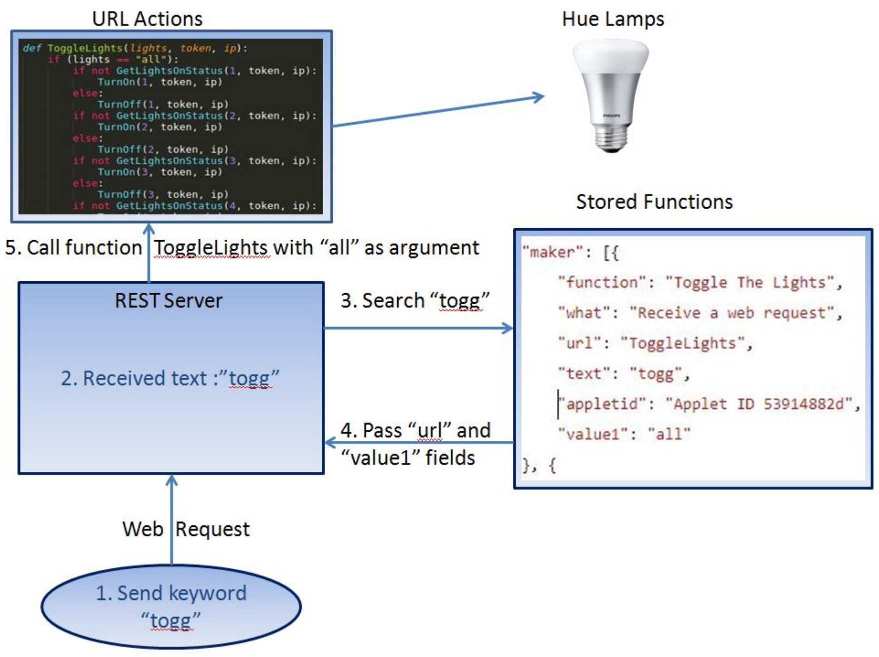
Now all the information of the used applet has been scraped and stored successfully and the applets can be migrated - parsed on the RPi. By looking for keywords on the scraped labels it can be distinguished which action the applet performs - turn on a lamp, blink a lamp, toggle a lamp etc. Based on those labels the Python script is able to call the corresponding function on the developers Hue API and use the scraped preselected values to distinguish which lamp should be subjected to changes. The names of the Hue lamps on IFTTT always match those on the official Hue online application and those stored on the ZigBee IP Bridge so there is never obsolete information that will not match. When the online applet has been successfully migrated to the RPi it will be triggered without having to contact IFTTT or use the Internet.

In order to have these functions migrated on the fog node, there should be an entity that stores all this information and a method to imitate the online functionality. When the triggers are fired they should execute an action based on the information that has been migrated. In order to store that scraped information we have used a JSON file. Since the migrated applets will probably not exceed a hundred, reading information from that file is efficient enough. Instead of JSON, a simple text file could be used, but it would require a lot of text parsing which should be generally avoided.

When the migration script is executed, based on the trigger and action channels that are used, a JSON file with all the scraped information is created. This file contains in case of littleBits channel the id and the name of the cloudBit and in case of IFTTT Maker channel the event keyword. From the labels that are scraped it is possible to distinguish whether the applet is made to turn on/off the lamps, toggle them, blink them etc. When the purpose of the applet is obtained the name of the function that needs be executed is stored under a certain key on the corresponding JSON object. More details about the JSON object can be found here 6.4.

5.3 REST API

IFTTT Maker channel performs web requests by defining an event keyword on the URL. In order be able to migrate the applets that use IFTTT Maker as trigger channel, there should be a way to perform these web requests locally. Every web request to ifttt.com except of the event keyword contains the unique token of the IFTTT Maker - developer. Based on that event keyword, IFTTT can distinguish which action will be executed. On the fog node, after the migration has taken place, the same event keyword will be used to perform the same action on the lamps locally. This time the web request will be send to a local web server that is running on the RPi. The web server is running on Python and the main library that was used to build the REST API is the web library.



```python
def ToggleLights(lights, token, ip):
    if (lights == "all"):
        if not GetLightsOnStatus(1, token, ip):
            TurnOn(1, token, ip)
        else:
            TurnOff(1, token, ip)
        if not GetLightsOnStatus(2, token, ip):
            TurnOn(2, token, ip)
        else:
            TurnOff(2, token, ip)
        if not GetLightsOnStatus(3, token, ip):
            TurnOn(3, token, ip)
        else:
            TurnOff(3, token, ip)
        if not GetLightsOnStatus(4, token, ip):
```

```json
"maker": [{
    "function": "Toggle The Lights",
    "what": "Receive a web request",
    "url": "ToggleLights",
    "text": "togg",
    "appletid": "Applet ID 53914882d",
    "value1": "all"
}, {
```

Figure 5.2. How the fog node works

When a web request is send to the web server, the event keyword is extracted from the URL and a script searches the JSON file with the migrated functions for it. If the keyword is found, the corresponding function is called and fires an action.

The JSON file with the stored functions is contacted in order to find if the the cloudBit (littleBits set as trigger) or the event keyword (IFTTT Maker set as trigger) exists in any of the stored objects. In case they do, the corresponding action for this entry is fired through the Hue developer API.

By migrating IFTTT Maker applets that run on the local REST Server we are able to have a variety of applications migrated on the RPi, since each cloudBit can only be used for one IFTTT applet.

5.4 Personalization

Since different ZigBee IP Bridges and different IFTTT accounts will have different local IP addresses and tokens, it is important that the fog node can acquire these parameters dynamically so it can be used in a *Plug and Play* way when it is connected to a local home network. In order for that to happen some of the parameters that are used should be imported dynamically for the current local home network settings and the current user account on IFTTT. More specifically the parameters than need to be imported dynamically are the following:

- **The internal IP of the ZigBee IP Bridge** - This parameter is necessary since the URLs of the Hue developer API need the IP of the bridge in order to be able to interact with the lamps. 6.4
- **The token of the ZigBee IP Bridge** - The bridge generates tokens that should be included in the requests towards the bridge. These tokens are generated with a POST registration request by the user and in order to be generated successfully, the user should press the bridge button in a time window of 30 seconds before sending the registration request. Without a valid token it is not possible to interact with the lamps. When the token is acquired successfully, it is stored in the same text file as the local IP and used every time a request is performed. The part of the code which acquires the token has been implemented in the same way as in the phue library [33].
- **The IFTTT Maker channel user token** - As it was mentioned before, in order for a developer to use the IFTTT Maker channel, a token generated by IFTTT should be included in all the web requests. The IFTTT Maker token is available on the webpage and in order to be acquired it should be scraped from the page in the same way as the applet content is scraped. After the acquisition, it is saved in a config file and used every time an IFTTT Maker request is made to IFTTT. 6.4

Every time an action is about to be executed on the lamps, these parameters are imported dynamically from the configuration files.

5.5 Demo

In order to control the lamps and demonstrate the functionality, a littleBits application has been made. The functionalities that the demo includes are triggering a migration of applets to the fog node, switch between cloud and fog and trigger the action event of the migrated applet. Since the are three different functionalities there should be three different inputs as well. The most typical input that littleBits are using is a buttonBit. Since three buttonBits must be connected in parallel on the same application, we should have a way of powering all of the three at the same time and also have a device following them, which will distinguish the action that must be executed based on which buttonBit is clicked. This device is the arduinoBit because it can take three bits as inputs and also because we need a gateway with the RPi. After the arduinoBit we connected the cloudBit and some output bits to illustrate the status of the application. Figure 5.3 shows the demo device.

As inputs (red) bits, the application consists of three different buttonBits which are getting power from a powerBit combined with a forkBit, which provides power to all three buttonBits simultaneously without having to be connected one after the other. An arduinoBit is placed after each buttonBit and it is used to add some logic to the application and act as a gateway between the littleBits and the RPi through the serial port. After the arduinoBit are a baragraphBit (green bit), a ledBit (green bit) and the cloudbBit.

The top button is used to trigger a new migration from IFTTT to the RPi. When it is clicked, the baragraphBit is getting bright for 200 ms to indicate the action. While the migration is ongoing, the ledBit blinks continuously until the migration is finished. The arduinoBit triggers the migration when it receives signal from this buttonBit by communicating through the serial port with the RPi. When the migration is over, the arduinoBit receives this verification through the serial port and stops the ledBit from blinking.

The bottom buttonBit is used to switch between cloud and fog. When this buttonBit is clicked the ledBit gets toggled. When the ledBit is bright the request will be fulfilled through the fog and when it is turned off the request will be fulfilled though the cloudBit. Since the cloudBit is a black-box and we can not interfere, we have to put all the logic before it - on the arduinoBit. When the ledBit is on, the arduinoBit will forward all the requests through the serial port to the RPi while when is it off it will send a signal to the cloudBit.

The middle buttonBit is the one that actually triggers the action. The bottom buttonBit toggles between fog and cloud but it does not send a request, the request will only be sent when the middle buttonBit is pressed. When it is pressed the baragraphBit attached to the arduinoBit will also get bright to indicate the click of the buttonBit. The action now will be executed on the lamps either though the cloud or through the fog.

By using this demo we can efficiently demonstrate the full potential of the project since we can migrate applets and execute actions both through the fog and the cloud only by clicking a button. On top of that the indications on the green bits can make the demo more user friendly and easy to understand.

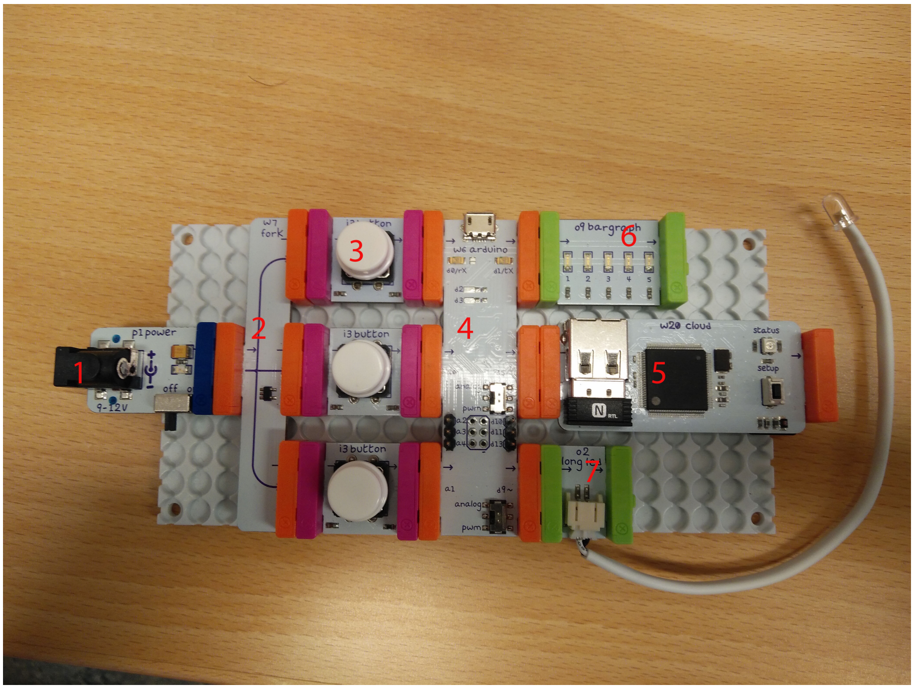


Figure 5.3. LittleBits Demo: 1. powerBit, 2. forkBit, 3. buttonBit, 4. arduinoBit, 5. cloudBit, 6. baragraphBit, 7.ledBit

5.6 Results

When the migration was completed, a new measurement was performed to estimate the improvement of the latency when using the fog node. In order to perform the measurement, the same methodology as before was followed and the same devices were used. The littleBits were connected through protoBit to the GPIOs of TI CC2650 Sensortag which stored a timestamp when there was signal from protoBit. TI CC2650 Sensortag was at the same time connected to the Ubuntu Computer through the serial port which turned the lights on through the Hue developer API when it received an input signal. As a result, TI CC2650 Sensortag calculated latency by subtracting its saved timestamp from the current.

The results have shown that the average latency now is 275ms (see 5.4), which is a lot faster than the original **littleBits - IFTTT - Philips Hue** applet. On top of that, the system now is completely independent of an Internet connection and its quality. In order to migrate six applets 25 seconds on average were needed and the total size of data migrated - downloaded were 0,42 Megabytes. As shown in the graph 5.4, the fluctuation between the measurements is very small and it is probably due

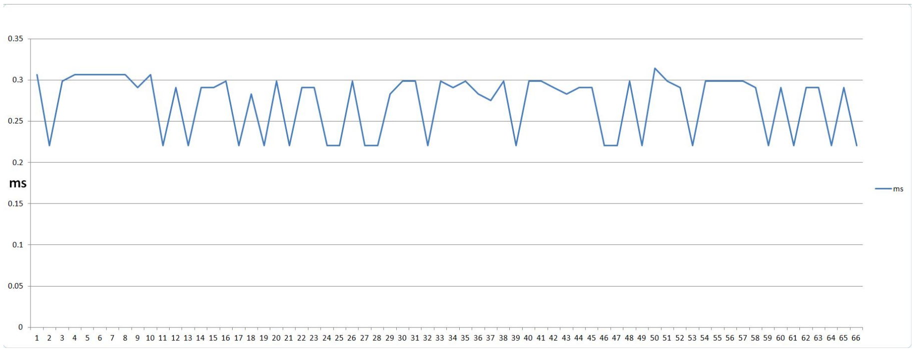


Figure 5.4. Fog Latency (ms)

to the speed of the serial port and the time that TI CC2650 Sensortag needs to start the sensor and read a new value from the register.

The improvement and consistency of the latency is shown on the CDFs that follow. Figure 5.5 shows a CDF graph in seconds for the applets that have been measured. In CDFs, a straight vertical line implies that the sample consists of measurements that do not differ significantly, while horizontal lines between two points indicate that there are very few measurements with values in this range. The fog line (dashed dotted red line) is an almost straight vertical line, which means that all the values in the measurement are almost similar, while in all the cloud based solutions the values have significantly longer latencies and larger fluctuations as well. As it can be seen **IFTTT Maker - Philips Hue** (dashed green line) applet has an almost horizontal line from 7 to 17 seconds almost stable at 0.05. That means that the values in the sample ranging from 7 to 17 seconds are few and so, the probability of finding a value between 7 and 17 is really low. The gradient of the line starts to increase between 17 and 25 seconds, meaning that most measurements (95%) lie in this range. On the **IFTTT Maker - IFTTT Maker** (dotted yellow line) applet, the curve has a "step" from 2 to 6 seconds with the relevant probability remaining constant and equal to 0.7. This means that 70% of the measurements are below 2 seconds and that there are very few samples between 2 and 6 seconds.

From the CDFs it becomes clear that the fog node has by far better performance and consistency compared to all the other implementations.

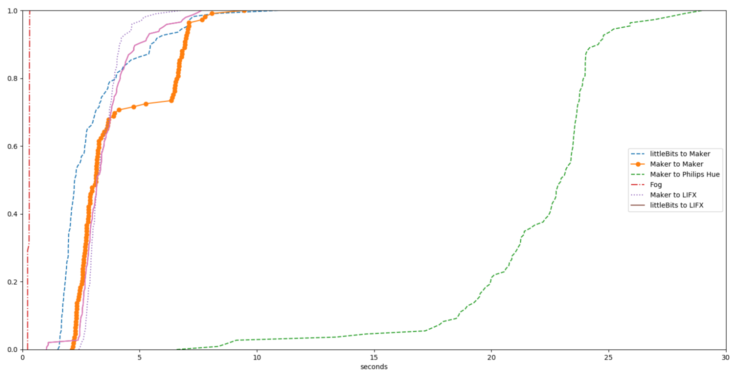


Figure 5.5. CDF for all measured applications (seconds)

6. Discussion

6.1 Design Choices

6.1.1 Parsing the IFTTT applet

The only way to parse IFTTT is to scrape the webpage. IFTTT provides partnership packets that can be purchased which can possibly provide the partner with API to monitor and interact with his applets and services. Although, at that time, developers and users without any special subscription can only retrieve information about their account through the webpage. Python provides some libraries that do just that, scraping webpages that require authentication, so it was the best way to approach this issue.

6.1.2 JSON Format

JSON format has been widely used in this master thesis. IFTTT APIs return JSON objects so using that data format for all objects in the system seemed the most reasonable approach. The file that stores all the information about the migrated applets is also in JSON format. A simple text file would also work but parsing text is always a last option solution which should be generally avoided. XML would also work but since JSON was used by IFTTT there was no reason to add extra complexity.

6.1.3 Usage of littleBits

IFTTT provides connectivity services for various DYI Electronics devices. Some of the devices only consist of a button that can be connected to IFTTT and trigger an event when pushed. These devices could be used as the littleBits buttonBit to use the Hue lamps but littleBits gives the user the opportunity to use various devices as triggers apart from buttons, like pressure sensors, light sensors, sound sensors etc and also provide an arduinoBit to add logic. Furthermore, someone using littleBits with some imagination can make various smart applets rather than just clicking a button. The only other DYI electronics devices that resemble littleBits, in the sense that consist of different modules that can be connected in many ways are Adafruit. The issue with

Adafruit is that the modules can not be easily connected as the littleBits with magnets and also require electronics knowledge. In that sense littleBits are the best choice to be used as triggers because their design provides simplicity and versatility at the same time.

6.1.4 Usage of Hue lamps

IFTTT provides connectivity services for 15 different smart lighting systems. It is not possible to purchase or investigate thoroughly all the services provided for all of these systems but Philips is the best known light manufacturer from the ones provided. Philips lighting system provides a user friendly interface and we have not noticed any bug that prevents the smooth operation of the lights. Furthermore, the developer API is straightforward and easy to work with. The only negative aspect when using the Philips Hue lamps, was the long latency when using the old firmware, that was exceeding 20 seconds on average. Comparing Hue lamps with LIFX, LIFX had a buggy setup process but the latency was not exceeding 3.5 seconds on average. Given the large number of light manufacturers that IFTTT provides to the users, we can assume that some of the other brands would also provide a smooth service and could be successfully used for the project.

6.2 Pros and Cons

The migration process is quite simple and last only a few seconds.

The applet migration is easy to expand to more channels and devices than the Hue lamps. Provided that a user has the hardware that is used (smart coffee makers, smart locks etc) by IFTTT channels to make an applet and that there are local APIs provided by the manufacturer to build applications with, then it would be possible, with little additions to migrate more applets for more channels.

The IFTTT Maker Channel has been widely used in the project both as a trigger and as an action channel. While using it as a trigger is easy and straightforward, using it as an action can introduce some challenges. Firstly the user has to write or use a script to sniff the incoming request and trigger new events after that. The script that needs to be written is simple but it requires a user with some technical knowledge. Also the router should support port forwarding so the request will not be dropped by the firewall, which also requires some level of technical skills from the user. Last, in case the network is part of a wider private network with strict rules or if the user is behind a double NAT, then the port forwarding might not work at all and user can not do anything about it.

In case IFTTT changes the structure of the web page the scraping script running BeatifulSoup will probably crash or it will not return all the information that are needed. If that happens the script should be updated to adapt to the new changes.

In case IFTTT decides to enhance its security mechanism by adding an extra field in the login page, like a token or a "I am not a bot" field, then the login script will not work anymore and the migration will not be possible. Of course this is unlikely to happen because none of the big service providers like Google, Facebook etc have used a policy like this which will make the webpage not so easily accessible to the users.

6.3 Conclusions

By the results that have been produced and the measurements that have been done we can say that migrating from cloud to fog can reduce latency significantly. In the measurements that have been presented in the beginning, the average latency ranged of the magnitude of seconds when the application was cloud based. With the use of fog, the latency has dropped to an average of 275 ms. The latency when using cloud was in many cases prohibitive for running almost real-time applications.

Migration is entirely possible by using well known open source tools and without the need of a premium subscription. Fog is a new architecture that has come to the attention of IT community recently but it is all about using current technology in a more versatile and out of the box way.

When someone looks on the services that are available on IFTTT, realizes how many devices someone can control just with a click of a button. Various house devices and services that handle personal data are easily accessible. As a result security becomes extremely important since a hack can have serious consequences for the users.

Two out of the three smart devices that were used during the thesis had some issues regarding their operation. Philips Hue initially had an unexpected high latency and LIFX Bulbs had a buggy setup process. With that in mind there is still room for improvement in the provided services of many devices.

6.4 Future Work

The migration can expand to support more services as actions. For this project we only used littleBits, LIFX Bulbs and Philips Hue lamps as hardware to interact with, but it is possible to expand to more devices as long as they provide an API for developers or if they can get hacked. Looking on IFTTT a user can realize the large amount of smart devices available on the market, like cameras, locks, gardening tools, gadgets etc. After migrating their functionality it would be possible to control many of these devices through the local home network with minimal latency and maximum availability no matter what is the status and the state of the Internet connection.

The project can expand in more ways than just including more IFTTT supported services and devices. It is possible to deploy sensors like the TI CC2650 Sensortag that has been used to perform the light measurements, to produce real time information and adjust the output devices to act based on the values that the sensor produces. The sensors can be used either by connecting their GPIO Pins to the ones of the RPi or by using 802.15.4 protocol to communicate wirelessly with deployed border routers connected on the RPis.

The cloudBit has a mounted SD card that runs Linux OS. During the development of the project we were able to sniff the packets from and to the cloudBit using wireshark. In order to do so, the RPi was set a WiFi access point which hosted the sniffer. The cloudBit packets sends and receives TCP packets under an encryption that was not possible to be decrypted. If the packets were decrypted it would probably be possible to read the payload of packet and understand when an input signal is sent to cloud control. In order to hack it and read the contents of the packets, the SD card was extracted and inserted to a Linux Computer[8]. Some configuration files were modified in order to deactivate the encryption but without success. Eventually the task was put on hold since there were more tasks that needed to be done and there were other alternatives to detour this issue. For future work it would be an interesting task to try to decrypt the contents of the cloudbit packets so it would not be necessary to use the arduino bit connected to the RPi through USB.

IFTTT contains many different smart Lighting Systems that can be used. Some of these systems contains triggers as well as actions. Since Philips provides a local API for the Hue lamps it is possible to create a custom trigger for our lamps. For example in order to create a trigger to notify when the lamps are on, a daemon can loop over the lamps state and when the lamps are turned on it can send a web request on IFTTT through the IFTTT Maker channel to initiate and action. The same principal can be implemented for other services with local APIs that we want to create triggers for, or even for devices that are not connected to IFTTT at all which can use IFTTT Maker requests as triggers.

At the moment only one ZigBee IP Bridge can be connected to an IFTTT account, so if a user has two different houses with two different bridges, he must also have two IFTTT accounts and switch manually from one account to another. It is possible that the same principle is used for more smart devices that connect to IFTTT. An improvement would be to "merge" various IFTTT accounts and bridges in one fog node so when the node is connected to a certain router it will use the right settings and applets for that bridge.

The fog node can store information regarding the user behavior, type and frequency of the applets that are used, schedules etc. After storing this information it could be possible to apply AI algorithms that will provide feedback and suggestions to the user.

A UI must be implemented if the service gets public. Some of the necessary features that must be implemented are a form for the users to provide their IFTTT credentials and a list of the services that they intend to migrate on their fog nodes.

References

[1] Cookie handling for HTTP clients.
https://docs.python.org/2/library/cookielib.html.

[2] IEEE 802.15.4. http://www.ieee802.org/15/pub/TG4.html.

[3] Maarten van Steen Andrew S. Tanebaum. *DISTRIBUTED SYSTEMS Principles and Paradigms*. Vrije University, Amsterdam, Netherlands, 2007.

[4] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1519-7. doi: 10.1145/2342509.2342513. URL
http://doi.acm.org/10.1145/2342509.2342513.

[5] Eric Bruno. A Cloud-based Approach for the Internet of Things.
https://www.linkedin.com/pulse/cloud-based-approach-internet-things-eric-bruno.
Retrieved: October 2015.

[6] Cisco. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.
Retrieved: 2015.

[7] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. Privacy mediators: Helping iot cross the chasm. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, pages 39–44, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4145-5. doi: 10.1145/2873587.2873600. URL
http://doi.acm.org/10.1145/2873587.2873600.

[8] Nitesh Dhanjani. *Abusing the Internet of Things Blackouts, Freakouts, and Stakeouts*, chapter 7, pages 207–211. O'Reilly Media, 1005 Gravenstein Highway NorthSebastopol, CA 95472 USA, 3 edition, 8 2015.
http://shop.oreilly.com/product/0636920033547.do.

[9] Adam Dunkels. Contiki OS. http://www.contiki-os.org/.

[10] Tim Fisher. How to Fix a 500 Internal Server Error.
https://www.lifewire.com/500-internal-server-error-explained-2622938. Retrieved: July 2017.

[11] RASPBERRY PI FOUNDATION. Raspberry PI. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[12] The Linux Foundation. Michael Enescu - From Cloud to Fog Computing and IoT | LinuxCon + CloudOpen North America 2014.
https://www.youtube.com/watch?v=1WfbnMU7CMc. Retrieved: 2014.

[13] Kovid Goyal. mechanize 0.3.3.
https://github.com/python-mechanize/mechanize.

[14] LiFi Labs Inc. LIFX. https://www.lifx.com/.

[15] LittleBits Inc. GETTING STARTED WITH THE CLOUDBIT.
https://littlebits.cc/cloudstart,.

[16] LittleBits Inc. LittleBits Inc.
https://littlebits.cc/how-it-works,.

[17] Texas Instruments Inc. CC2650 SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU.
http://www.ti.com/product/CC2650,.

[18] Texas Instruments Inc. CC2650 SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU.
http://www.ti.com/product/CC2650,.

[19] European Telecommunications Standards Institute. Multi-access Edge Computing. http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.

[20] BI Intelligence. Edge Computing in the IoT. https://www.businessinsider.com/intelligence/research-store?IR=T&utm_source=businessinsider&utm_medium=content_marketing&utm_term=content_marketing_store_text_link_internet-of-things-insurance-home-life-auto-trends-2016-10&utm_content=report_store_content_marketing_text_link&utm_campaign=content_marketing_store_link&vertical=iot#!/Edge-Computing-in-the-IoT/p/68220396. Retrieved: June 2016.

[21] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 88–99, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5002-0. doi: 10.1145/3083187.3083191. URL
http://doi.acm.org/10.1145/3083187.3083191.

[22] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2. doi: 10.1145/1879141.1879143. URL
http://doi.acm.org/10.1145/1879141.1879143.

[23] Alexander Tibbets Linden Tibbets and Jesse Tane. IFTTT Inc.
https://ifttt.com.

[24] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bongjun Ko, and Theodoros Salonidis. Live service migration in mobile edge clouds. *CoRR*, abs/1706.04118, 2017. URL
http://arxiv.org/abs/1706.04118.

[25] Holy Mackerel. How to scrape a website that requires login first with Python. https://stackoverflow.com/questions/20039643/how-to-scrape-a-website-that-requires-login-first-with-python.

[26] Lori MacVittie. Networking in the iot age: The cloud latency factor. *Network Computing*, pages 0–1, 9 2016. `http://www.networkcomputing.com/networking/networking-iot-age-cloud-latency-factor/1283791166`.

[27] Amy Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated. `http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-202`

[28] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K. K. R. Choo, and M. Dlodlo. From cloud to fog computing: A review and a conceptual live vm migration framework. *IEEE Access*, 5:8284–8300, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2692960.

[29] Philips. Philips Hue API. `https://www.developers.meethue.com/`,.

[30] Philips. Hue Personal Wireless Lighting. `http://www2.meethue.com/sv-se/`,.

[31] Leonard Richardson. beautifulsoup4 4.6.0. `https://pypi.python.org/pypi/beautifulsoup4`.

[32] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1): 30–39, Jan 2017. ISSN 0018-9162. doi: 10.1109/MC.2017.9.

[33] studioimaginaire. A Python library for the Philips Hue system. `https://github.com/studioimaginaire/phue`.

Appendices

Measurement Technical Details

Every time the button is pressed the protoBit sends signal to GPIO pin DP0 on the TI CC2650 Sensortag, at the same time the cloudBit gets the signal as well. The TI CC2650 Sensortag runs a loop which waits for any input signal in pin DP0. When there is input signal, it records the time and activates the light sensor. Then the light sensor loops reading light values continuously and waits until the lamp gets bright and the luminance value exceeds a certain threshold. TI CC2650 Sensortag is placed directly below the lamp so it will notice instantly the increasing luminance. When that happens the latency is calculated by subtracting the initial time recorded from the current time.

Detailed steps for setting up the measurement

- Make a voltage divider to use between the protoBit and TI CC2650 Sensortag.
  That task is needed since littleBits work on 5V and TI CC2650 Sensortag works on 3.3V.
- Connect the protoBit after the buttonBit and before the cloudBit.
  That task is necessary so when a signal is sent to cloudBit we can receive the signal to the protoBit.
- Connect the cables from protoBit to TI CC2650 Sensortag's GPIO Pin DP0 and GND using the voltage divider.
- Write contiki code so when there is input signal on TI CC2650 Sensortag's GPIO Pin DP0, a timer starts and a process loops on the light sensor until the luminance is up to a certain threshold (lamp is on).

How threads start simultaneously

A python script reads the serial port and when the GPIO pin DP0 on TI CC2650 Sensortag gets an input signal, the timer starts and TI CC2650 Sensortag prints an output through the serial port. When the computer reads this output, fires all of the three events using three different threads.

Class Object

For every applet that will be migrated a new custom class object is created that includes all the necessary information. These objects are populated during the scraping of the applet parameters from IFTTT and contain fields as which are the involved channels, the labels, the selected devices, the applet id, flags etc.

Details about the JSON Object

Along with the name of the function the arguments that need to be used are stored on the same object under predefined keys. For some applets these arguments are just the id of the lamps and for some others there might also be an other field indicating an extra value that has to used. For example when the lamps will be dimmed, the dim level should also be included as a parameter. When a trigger is called, the script locates which JSON object contains that trigger and calls the function that lies under a certain JSON key on that same object. When the function and the arguments are obtained, the script initiates the action through the Hue developer API. In this way, the same event that would trigger an action on the lamps through IFTTT performs that action locally with minimum latency.

Internal bridge IP

The internal - local IP of the bridge is acquired by performing a GET request to *www.meethue.com/api/nupnp*. The result of this request is a JSON object containing the local IP and the id of the bridge. This information is stored in a config file and it is acquired every time a local request is performed.

IFTTT user token

The Python script which performs the scraping should request *https://ifttt.com/services/maker_webhooks/settings* and scrape the page to acquire the token.