OPTIMIZING BATTERY CONSUMPTION OF EDGE IOT DEVICES

by

Mujahid Khan

A thesis submitted to the Graduate College of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2021

Committee Members:

Dr. Anne Hee Hiong Ngu, Chair

Dr. Byran Gao

Dr. Qijun Gu

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

I dedicate this work to my loving parents and my supportive friends. A very special thank you to my mother Shagufta Khizar and my father Khizar Hayat Khan, whose exceptional support and overwhelming throughput my time here away from home has nothing but been a blessing to me. I would also like to mention my friends Pseudo, GM, Paitha and Moi for making this journey of graduate studies on a foreign soil considerably easier to progress. I am very thankful to have these people in my life and I owe a significant portion of my success to them.

ACKNOWLEDGEMENTS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis facilisis convallis mi eget convallis. Maecenas ut erat purus. Nullam nibh dolor, blandit vitae blandit vel, consectetur suscipit nulla. Aliquam erat volutpat. Mauris vel elit eu ipsum rutrum luctus. In molestie id urna ut ornare. Pellentesque lacus lectus, sollicitudin sit amet risus eu, tincidunt fermentum justo. Nullam sagittis velit id purus posuere pulvinar lobortis ac arcu. Ut nec massa vehicula, tempus purus eu, malesuada nunc. Quisque aliquet tortor odio, eget efficitur mi aliquam a.

Cras libero purus, vehicula ut nibh sit amet, porta efficitur odio. Quisque ut nisi turpis. Vestibulum vel nulla ac elit varius faucibus sed ac augue. Proin quis malesuada quam. Integer semper tellus vitae rutrum convallis. Integer at orci libero. Curabitur lacus justo, ullamcorper vel lacinia nec, vulputate a nulla.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

ABSTRACT

Recent advances in the world of IoT have significantly improved what one can do with the help of personal gadgets such as smartwatches and other edge devices. This improvement has lead to the inclusion of additional functionality like fall detection, temperature and motion sensing etcetra. Although this in itself is not a bad thing, however, this does pose the problem of increased energy consumption on devices which are already constrained by limited battery. We present our study which discusses the possibility of optimizing energy consumption for edge IoT devices by reducing communication cost of applications by determining an optimal chunk size. We present a metric **energy-per-byte** or **EPB** which helps determine the optimal chunk size for respective edge devices. We evaluate our approach with the help of two case studies using different edge devices and applications, and we achieve considerable energy optimization depending on the app functionality and the hardware involved.

I. Introduction

In the past couple of years, IoT has made significant progress. From personal devices such as smartwatches and environment like smart homes, to industrially scaling projects like smart cities and smart, autonomous cars. There has been an extensive application of IoT edge devices to an extent that today, a considerably significant amount of people possess at least one such device —be it your smart car or a smartwatch that you regularly or routinely use.

Recent advances in the world of IoT have increased the amount of use cases that these personal devices such as smartwatches have. But having additional functionality also poses the problem of increased energy consumption on devices already constrained by limited battery. For example, a device logging sensor data, say temperature, in real-time is bound to use more energy if it starts logging data from a motion sensor as well.

This work studies the possibility of optimizing energy consumption for edge IoT devices by aiding the programmer in setting parameters that would ultimately lead to lesser energy consumption. We aim to focus on optimizing the communication cost of applications by determining the optimal chunk size, with which the data should be transferred. It also involves a case study comparing energy consumption of native app and the same app running in a container environment such as Docker (cite here).

There has been an extensive amount of work on energy optimization —both without and in an IoT setting. Xiao et al. [7] compared 3G and Wi-fi while Balasubramanian et al. (cite here) compared GSM, 3G and Wi-Fi with respect to energy consumption. However, neither of them compared energy consumption of respective data transfer technologies for throughput efficiency for the same task.

Gupta et al. (cite here) made a measurement study of energy consumption

when using VoIP applications with Wi-Fi connection in smartphones and showed that power saving mode in Wi-Fi together with intelligent scanning techniques can reduce energy consumption. Xiao et al. (cite here) measured energy consumption for video streaming on mobile apps and concluded that Wi-Fi is more efficient than 3G. There are other measurement studies as well which compare different modes of communication but none of them discuss energy consumption differences for different packet sizes.

The work that seems the most related to ours is by Friedman et al. (cite here) where they measured power and throughput performance of Bluetooth and Wi-Fi usage in smartphones. They concluded that power consumption is generally linear with the obtained throughput, and Bluetooth uses less energy than Wi-Fi. However, this study also showed that different hardware and different software have different results and there is no generic trend. They also concluded that an upper bound does exist, bottlenecked by the receiver, after which the sender expends more power retransmitting packets. This dependency of energy consumption on software is also discussed by Flinn and Satyanarayanan (cite here) who point out that "*There is growing consensus that advances in battery technology and low-power circuit design cannot, by themselves, meet the energy needs of future movile computers* "(cite here). This observation has been confirmed by recent advances in green software engineering, which demonstrated how the source of energy leaks can be software-related as well (cite here).

Our work on the other hand, considers multiple constraints in real-world IoT setting. Firstly, the transmission is not always continuous. The data may be communicated in regular or irregular intervals. Secondly, the amount of data to send depends on the amount of data collected by the sensors which may vary based on different applications. Thirdly, since the transmission may not be continuous, if the app demands it, based off energy consumption data from different chunk sizes, an

optimal communication interval can be set to reduce the amount of buffering required (if needed). Lastly, unique apps on different devices with different operating systems will have their own optimal throughput efficiency for energy consumption. Given the hardware configuration of such a device, our tool can determine the optimal chunk size for data transfer with respect to energy consumption.

I.1 Contributions

This thesis has the following contributions. It:

- proposes a method to estimate energy consumption of edge IoT devices
- proposes a new approach to reduce energy consumption with the help of optimal chunk size
- presents a new metric called Energy-Per-Byte or EPB
- discusses a case study to compare energy consumption between native, Lingua Franca and container version of the same app.
- evaluates the effectiveness of this approach on two real-world applications of different domains.

The organization of the rest of the thesis as as follows. In chapter II, we will provide background information about edge IoT devices, energy usage and optimization, containers and middleware accessor frameworks. In chapter III, we briefly summarize the related work and chapter IV will discuss our methodology in detail. In chapter V, we will discuss our evaluation and results, and provide our reasoning. Chapter VI, will discuss the case study of energy cost comparison between native, Lingua Franca and Docker version of the same application and finally in chapter VII, we will conclude our thesis and discuss possible future work.

II. Literature Review

Over the past few years, Internet of Things or IoT, has had a significant impact in our lives. It plays an important role whether it is a smartphone or a smarthome environment or just something as handy as a smartwatch. Different types of data is collected and exchanged among interconnected sensors/devices through modern communication network infrastructure connected by million of IoT nodes [8, 9, 10, 11, 12]. This direction of computing has already overtaken the traditional methods based on stationary computing [13]. As a paradigm, IoT expresses that most physical devices, such as smart phones, smart watches and other embedded devices are interconnected with each other. These devices communicate with data centers and exchange information —all while adhering to their routine tasks [10].

Following various popular technologies these days, such as smart homes, smart grid and smart healthcare, IoT has become one of the essential components of people's home and workplace existence. It will continue to impact the daily life of people and its not just limited to technology. IoT has been reported to be one of the most important technologies that will impact US interests in 2025 [10]. The number of interconnected physical devices has already transcended the human population for a couple years now. In 2012, there were 9 billion interconnected physical devices [13]. This rapid increase in the number of mobile devices suggested that conventional centralized cloud computing would struggle to satisfy the Quality of Service (QoS) for many applications. However, with the introduction of 5G technology, edge computing becomes a viable and key solution to solve this issue [14, 15, 16]. The edge computing platform allows edge nodes to respond to service demands which results in reduced bandwidth consumption and network latency. Figure 1 shows a basic edge computing architecture.

Edge computing based IoT helps solve some critical issues and improves

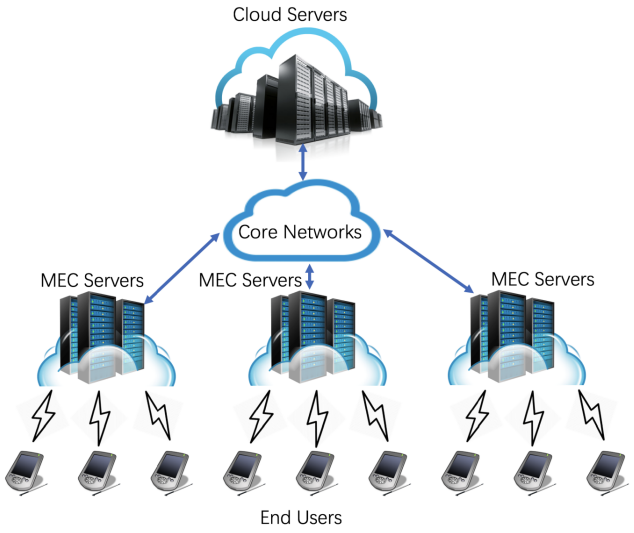

Figure 1: The basic edge computing architecture

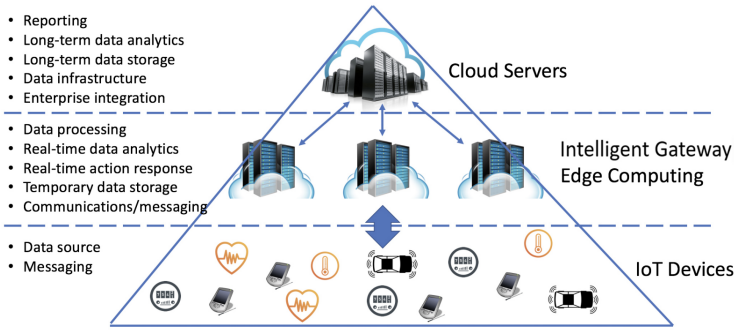

Figure 2: Edge computing based IoT architecture

performance. Not to mention, IoT and edge computing share some characteristics which is further clarified by figure 2.

It further illustrates that IoT devices are end users for edge computing and IoT can benefit from both cloud and edge computing. The latter helps with faster response times and provides a tolerable computational capacity and storage space. Nowadays, the physical end user devices such as smart phones are considered edge devices. They may have an exclusively local component of an application but most generally, some sort of data is communicated with the cloud server, with or without

the help of a gateway device.

One of the restricting factors among IoT edge devices is limited battery life (among limited storage and other things). Even though IoT and mobile devices are enabling a connected future that promises huge amounts of time and money saved with better automation, control in industry and our everyday activities, as well as other benefits such as better health care via remote monitoring and efficient fuel usage in smart and increasingly autonomous vehicles [1], it is a matter of fact that amongst these are portable devices which require a battery to operate. Usually, these devices have small form factors which limits the size of battery that can be used in these devices. There haven't been dramatic improvements in terms of alternative, more reliable battery types which in turn limits the capabilities of components in such edge devices. Figure 3 shows a breakdown of average energy usage across 1520 users of Samsung Galaxy S3 and S4 mobile devices [1]. In this experiment, users were divided into five groups based on their activity levels. It shows that each group of users has varying needs for which respective hardware component consumes more energy. In the recent years, there has been a strong motivation to minimize energy and power across all of these components so that mobile and IoT devices last longer on a single charge, and to also allow more sophisticated components such as CPU, GPUs and NPUs [1].

This need of minimizing energy consumption of such devices had lead to a plethora of techniques being developed to reduce the overall energy consumption of edge devices. Researchers have used machine learning approaches to categorize applications during execution and choose a suitable pre-existing power plan [17, 18, 19]. However, it was later realized that this does not capture dynamic workload variations [20]. Later on, a new technique was proposed which involved phase level instrumentation to collect workload statistics at runtime. Essentially, the worload was divided into snippets and performance application programming

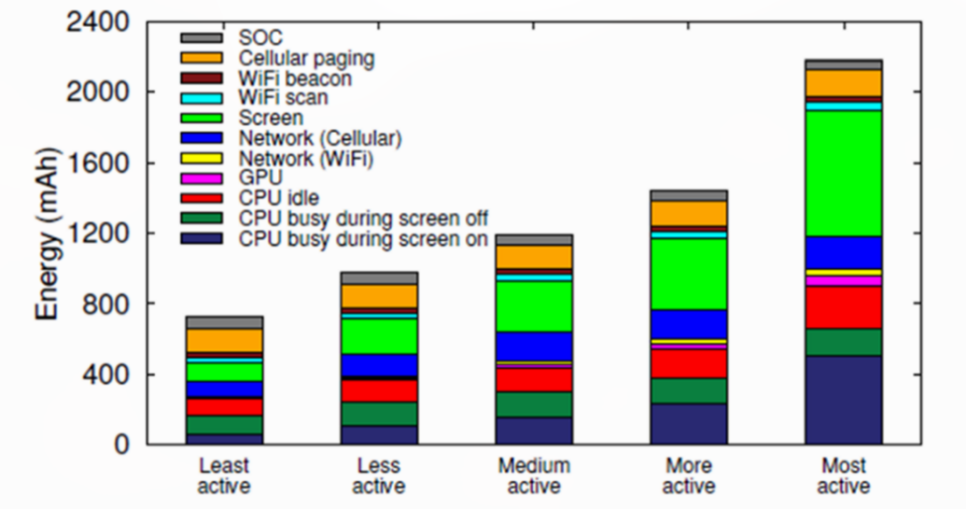

Figure 3: Average daily energy drain breakdown of 5 groups of 1520 apps [1]

interface calls were inserted between each snippet. The data collected from each snippet was then used to control the power states of processing elements [21]. There are other ML based approaches as well where some are better than the other using Reinforcement Learning and Imitation Learning [22, 23, 24, 25, 26, 27, 28]. Other than display optimations [29, 30, 31], wireless radio optimizations [32, 33, 34, 35, 36, 37] and main memory optimizations [38], there have also been software and user-centric optimizations. A variety of OS level energy management techniques have been proposed in literature [39, 40]. Runtime software profiling can identify the most power and energy hungry resournces that should be targetted by dynamic managament techniques. Powerscope [41] is one such example of a tool. It maps energy consumption of active applications to program structure. It further combines hardware instrumentation to measure current levels with kernel software support to perform statistical sampling of system activity. For mobile devices, the display is the main user interface. User satisfaction is directly impacted by brightness and content on the display. CAPED [2] uses an online learning algorithm that dynamically controls the display brightness to meet individual user's

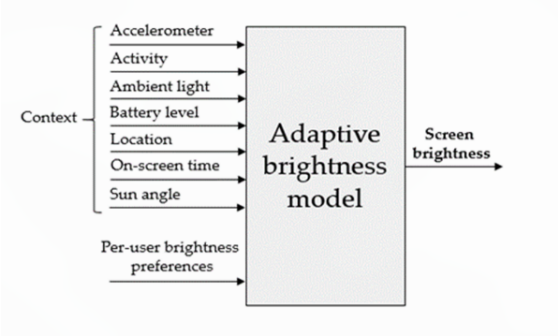

Figure 4: Online learning control of display brightness [2]

preferences. Figure 4 depicts how CAPED's online learning control works.

Virtualization is commonly defined as the act of creating a virtual version of something, including but not limited to, hardware platforms, storage devices and network resources. It has many advantages including flexibility, capacity, processing power, growth in demand, and energy efficiency [42]. A virtual machine instance represents an entire isolated environment. Multiple isolated environments can run and multiplex the resources of the same host. However, a new model has emerged which has been quite in demand for the past couple of years. *Containers* [43] allow multiple user-space instances and are generally lightweight. They have limited overhead compared to Virtual Machines (VMs) and unlike virtualization, paravirtualization or full virtualization, they do not require an emulation layer to run. Instead they use the host OS [44]. One such notable example of a container framework is Docker [43]. It unarguably started the container movement. Docker is open source system that provides an automated way to deplot applications inside containers [42].

Docker consists of multiple core components, described quite aptly by [43]:

- **Docker client and daemon:** the client is basically a command-line binary which performs requests for the daemon. The daemon itself can either be run remotely or on the same host.

- **Docker images:** labelled as the "source code "for the containers, images are the building blocks of Docker.
- **Registries:** used to store the images built. There are two types of registeries, public and private
- **Docker container:** applications and services are packaged inside containers. They are released from images and may contain one or more services.

Container performance is an important attribute to evaluate the quality of offered service. It represents system throughput, responsiveness, resource utilization, response time, latency, failure rate and fault tolerance [45]. With containers being very flexible and having the ability to fine tune specific metrics, the important question to ask is whether they can help preserve energy in certain scenarios. This will be further discussed in a later section.

Another technology that is quite interesting is edge based IoT middleware frameworks. One good example of is the Apache Cordova [46]. A typical IoT middleware comprises of a three-layer architecture i.e. edge, gateway and cloud [47]. A middleware is generally designed to be the intermediary between IoT devices and applications. The interaction is performed with the help of an *accessor*. Accessors provide the abstraction for smart things across different hardware or software platforms to interact. They further allow for smarter interactions, sharing and portability. They are usually implemented in Javascript to keep it lightweight and ubiquitous, allowing things to communicate to share information in a message oriented fashion [4]. An accessor has inputs, by which the swarmlet makes requests, and outputs, by which the service issues responses. The responses can be asynchronous and an accessor does not need to have an input; it can spontaneouly produce outputs [3]. This is also depicted in Figure 5. Accessor hosts function as a middleware service which is used to instantiate and execute accessors. They have

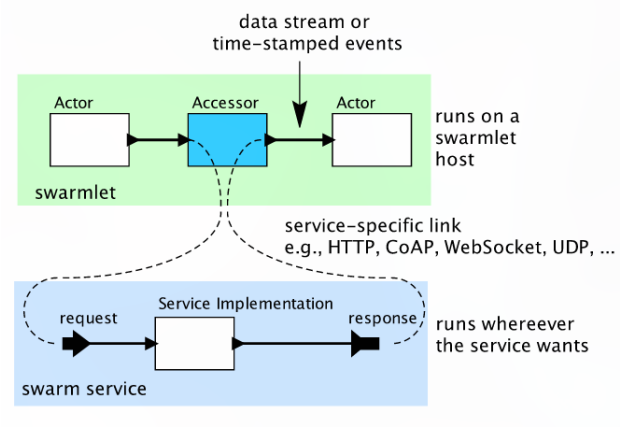

Figure 5: Design pattern of accessors [3]

their own specific set of actions that they can perform but the most common accessor hosts include Browser, Node and Cordova. The first supports executing accessors in a web browser whereas a Node host is just and Node.js engine with support for common host's capabilities [4]. A Cordova host is an extension of the Apache Cordova's cross mobile program development platform. It is used for building application using basic web development elements such as HTML, CSS and JS, in a unified code base and targeted to multiple platforms like Android and IOS. Other than writing the plugin for the respective target device, no additional programming is required as its javascript interface interacts with native language APIs of physical or virtual IoT devices.

The architecture of Cordova Accessor Host is given by Figure 6. Briefly, it consists of a webview and a Cordova Plugin. Webview comprises of an implementation for the Common Host, an implementation for accessor host specific to Cordova and a bundle of all community developed accessors that can be shared. This is what makes up the Cordova Accessor Host. The swarmlet.js component acts as an entry method, used to launch the accessor pipeline.

The usage of actors in concurrent programming is a common occurrence. Actors are concurrent objects that communicate by sending each other messages [48]. In

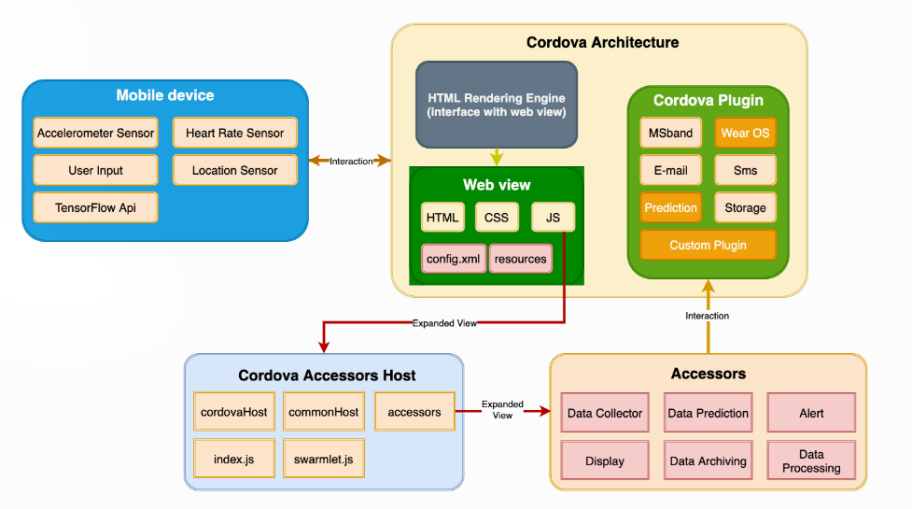

Figure 6: Design pattern of accessors [4]

the recent years, another framework called Lingua Franca, has been proposed and developed. Lingua Franca or LF, is a polyglot coordination language for concurrent, time sensitive applications which can range anywhere from low level embedded code to distributed cloud and edge applications [49]. An LF program consists of reactors and reactions. Reactions define the functionality of the specified reactors in target languages. Reactors are similar to actors in a sense that they are used to send messages to each other. However, these messages are timestamped and the reactors are deterministic by default. Any non determinism has to explicitly specified within the LF program. It must be noted that LF is a coordination language and not a programming language, and is only used to write interfaces and composition of reactors [49]. Going into additional details about the inner workings of LF and its components is out of scope for this study however, the general abstraction of an application's functionality regardless of the target language is something that is similar to the Cordova Host. One question one would like to ask is whether code generated by LF performs better than native code. This too, like Docker, will be discussed in a later section with the help of case study.

III. Related Work

IV. Methodology

As mentioned in the previous sections, the rapid increase in IoT devices as well as its use-cases has significantly impacted and at times more often than not, improved our experiences. However, when these personal, and usually hand-held or wearable devices like smart phones and smart watches, have a plethora of added functionality, it also poses the problem of increased energy consumption. This results on a less uptime for one charge of a battery which corresponds to the device not being utilized for proper amounts of time. This is especially important in the healthcare sector where a device monitoring a patient's vitals should be able to do so for prolonged lengths of time.

In this thesis, our study focused on optimizing energy consumption for communication. Most applications on an edge device usually communicate with a remote server to exchange different data and it varies from application to application. But within these apps, communication is a core aspect and is widely ignored. The best general practice used when designing these apps is to send data in sizes multiples of 2, which generally means faster communication without unwanted packet drops but on the other hand, is no where near optimal. This is due to the fact that each edge device may have different hardware and different software versions, and the energy cost for communication for each possible variation of these is different [50]. It has no optimal trend, therefore, the best practice is to use general values of chunk sizes that the protocol can easily handle. Our work focuses on communication cost because most of the times other hardware related factors are not flexible or would otherwise comprise the integrity of the task.

This work is divided into two portions. First, we present a pretty straightforward way to estimate an application's energy consumption over time. Next, we present an approach to compute optimal chunk sizes for respective edge

devices. We evaluate and confirm whether this optimal chunk size approach consumes less energy or not with the help of our energy estimation approach, on two real-world apps.

IV.1 Estimating Energy Usage

There have been other approaches in the past to estimate energy consumption of apps [51] but we adopt an approach that is similar to energest [52] which is an energy profiling approach for Contiki OS. We combine runtime information from applications with static information from data sheets of specific hardware components to estimate energy consumption. Before getting into the details, we will define some terminologies.

Definition 1 ***Current*** *(I) is defined as the flow of electrical charge (q) over time (t). It is measured in Amperes (A).*

Definition 2 ***Power****(P) is defined as the amount of charge (q) moved through voltage (V) in a time interval (t). It is measured in Watts (W).*

For devices powered by a battery, both current and voltage is an important factor and is heavily dependent on the components used on the device itself. Different electrical components operate on different voltages, thereby, drawing different amounts of current, resulting in different power consumption. However, in order to calculate the energy consumption, the dynamic factor is time. Power used or spent over a specific period of time equals the energy used by the specific device or specific component of device.

Definition 3 ***Energy****(E) is defined as the amount of power (P) expended over a time interval (t). It is measured in Joules (J).*

We use these definitions to be able to express how we estimate the energy consumption of a specific edge IoT device. Almost all hardware components when designed and brought to market have a corresponding data sheet publicly available, with important information such as circuit diagrams, and current and voltage values for different modes (if possible). Current and voltage are dynamic factors however by using information from said data sheets, we can estimate these values by computing the mean average sum of values corresponding to different operating modes. The data sheet for BCM43455, the wifi adapter used in Raspberry Pi 4 Model B [53] contains valuable information regarding different technical details such as frequency, circuit design, power management etc. However, what is interesting to us is current and voltage values. Figure 7 provides a snapshot from one such table. It outlines the different current values corresponding to different voltage values for different modes. In a real world setting, the devices switch between these modes depending on various factors. A reasonable assumption in this case would be to take the mean average sum of values for different modes.

Given this information, we can estimate the power `P` that the wifi adapter would use in this case when its turned on. Consider an app `A` which finishes executing in about `t` seconds. The total energy consumed (by the wifi adapter) by running `A` would then be calculated by `E = P * t`.

In order to calculate the estimated energy for specific block of code, such as ones which handle communication, all one needs to do is instrument the app to log execution time for that specific block of code. And then we can calculate the energy usage of that specific portion of program. The main idea behind this approach to estimate energy is that software developers don't usually have the technical equipment required to individually measure current and voltage values of actual devices. Given that setback, estimating energy consumption off of publicly available data sheets and runtime information from the app itself is a pretty reliable way to

Mode	V_{BAT} = 3.6V, V_{DDIO} = 1.8V, T_A25°C	
	$V_{BAT, mA}$	$V_{IO, uA}$[a]
Sleep Modes		
Radio off [b]	0.006	5
Sleep [c]	0.020	200
IEEE Power Save: DTIM = 1, single RX [d]	1.25	200
IEEE Power Save: DTIM = 3, single RX	0.45	200
Active RX Modes		
Continuous RX mode: MCS7, HT20, 1SS [e, f]	55	60
CRS: HT20 [g]	50	60
Mode	V_{BAT} = 3.6V, V_{DDIO} = 1.8V, T_A25°C	
	$V_{BAT, mA}$	$V_{IO, uA}$[a]
Active TX Modes – Internal PA		
Continuous TX mode: 1 Mbps @ 21.5 dBm [h]	400	60
Continuous TX mode: MCS7, HT20, 1SS, 1 TX @ 19 dBm [h]	350	60

a. VIO is specified with all pins idle (not switching) and not driving any loads.
b. WL_REG_ON and BT_REG_ON are both low. All supplies are present.
c. Idle, not associated, or inter-beacon.
d. Beacon Interval = 102.4 ms. Beacon duration = 1 ms @ 1 Mbps. Average current over 3× DTIM intervals.
e. Duty cycle is 100%. Carrier sense (CS) detect/packet receive.
f. Measured using packet engine test mode.
g. Carrier sense (CCA) when no carrier present.
h. Duty cycle is 100%.

Figure 7: Current and Voltage values for BCM43455

gather the required data.

IV.2 Calculating Optimal Chunk Size

As mentioned before, our study focuses on optimizing communication cost of apps for IoT edge devices. Even though the best practices involved in this case do perform well for average case, our hypothesis is that after a communication pipe is established between the client and the server, having data sent or received in different chunk sizes could have an impact on the energy consumption of the whole process. The motivation behind this is that even though hardware optimizations such as power saving modes improve energy consumption efficiency, there are still software related configurations that can be altered to reduce the overall energy usage. We chose to focus on communication cost because it is an integral part of more or less every app deployed on an edge device where it communicated with the server one way or another to transmit some sort of data.

But why communication cost? Indeed, there are multiple code related factors that can help optimize energy consumption but our work is done under the assumption that these factors, such as data collection, via the sensor, the disk I/O operations etc. differ heavily from use-case to use-case and any alteration in code functionality to them, may affect their usage in real world. For example, a device monitoring a patient's heartbeat continuously is bound to use more energy than, let's say, monitoring every five minutes. However, in such critical examples, these types of optimizations severely undermine the actual functionality of the app. Therefore, our approach addresses a factor that is independent of the sensor usage in such a context and can potentially work on diverse examples.

To better explain how we calculate the optimal chunk size, we will first define energy-per-byte or EPB. It is a metric we use to determine which chunk size consumes the least amount of energy.

Definition 4 ***EPB*** *or Energy-per-Byte is defined as the amount of energy consumed when a single byte of data is sent from one device to another.*

But why is EPB an important factor rather than the overall energy consumed for the corresponding chunk size? We will explain this with the help of an example. Consider an app `C` on the edge device `EdgeD` that needs to send 256 bytes to the server-side app `S`. Assume you calculated the energy to send data at a chunk size of 64 bytes was the lowest at `E1` Joules. However, if you used 128 bytes at a time to send this data, you consume `E2` Joules which is slightly more than `E1`. In this case, we can not say the 64 bytes is the optimal chunk size for `EdgeD` because it would take `E1*4` amounts of energy to send the complete data and `E2*2` on the other hand. For the sake of this example, assume `E1` has the arbitrary value of 2 Joules (actual values are way way less and are usually in microJoules) and `E2` has the arbitrary value of 3 Joules. 64 bytes would take 8 Joules in total to send 256 bytes of data where as 128 bytes of chunk size would only use 6 Joules. That is why, EPB

is an important factor and metric in this case. So what we need to do is divide `E1` by 64 and `E2` by 128 to get the EPB for their respective chunk sizes. Now, the lower energy consumption value of the two would be the optimal one as it would consume the lowest amount of energy among different chunk sizes.

To calculate the optimal chunk size on an edge device, we deploy a Python script which comprises of a list of different chunk sizes of order 2^n where `n` by default can be anything between 1 and 30 (included). The upper limit can be adjusted depending on the use case of the program. It then proceeds to send data in different chunk sizes and then measure the time it took for the communication to take place. This time is then used to calculated the overall energy consumption for sending a specific amount of data.

This is a pretty straightforward but novel approach to aid programmers in developing applications for a target edge device where they can optimize the communication cost in terms of energy. In some use cases, the difference is considerable while in others it may not be that much. In short, it depends on the nature of the app itself. The following section will clarify more as to why it is as we describe our experimental and nature of the apps, and discuss our results.

V. Evaluation

In order to evaluate the efficiency of our approach, we evaluated it on two real-world applications. These apps are on completely different hardware with a completely different framework and different use cases. We will present and discuss our results for both of them individually by first explaining the experimental setups involved and then provide reasoning for the results obtained.

V.1 Experimental Setup —TempSens

For our first experimental setup, the application we use is called `TempSens`. It is a mock app that was developed from scratch. It continuously monitors and log sensor data from a DHT11 sensor which is used to monitor temperature and humidity levels. Overall, the app is very simple and has a basic functionality: continuously monitor data and communicate it with the server every 60 seconds. The hardware configuration in this setting utilizes a Raspberry Pi 4 Model B running a headless Raspbian OS as an edge device where it functions as a client, and a remote server deployed on a MacBook Pro running Big Sur.

Raspberry Pi 4 Model B houses a System-on-Chip (SoC) which makes it much more affordable and reliable in terms of computing capability. It uses BCM2711 as its SoC type which is one of more energy and cost effective alternatives than its previous counterparts. The `TempSensClient` is written in Python 3 and uses RPi.GPIO to interface with the sensors. Figure **??** shows an extension board that is used to connect sensors to the Raspberry Pi. Figure 9 on the other hand provides the naming methods used for WiringPi, Board and the intrinsic pin name for GPIO extension board. For instance, for GPIO27, the WiringPi naming method is 2, Board naming method is 13 and its intrinsic name is GPIO2 [5]. The overall

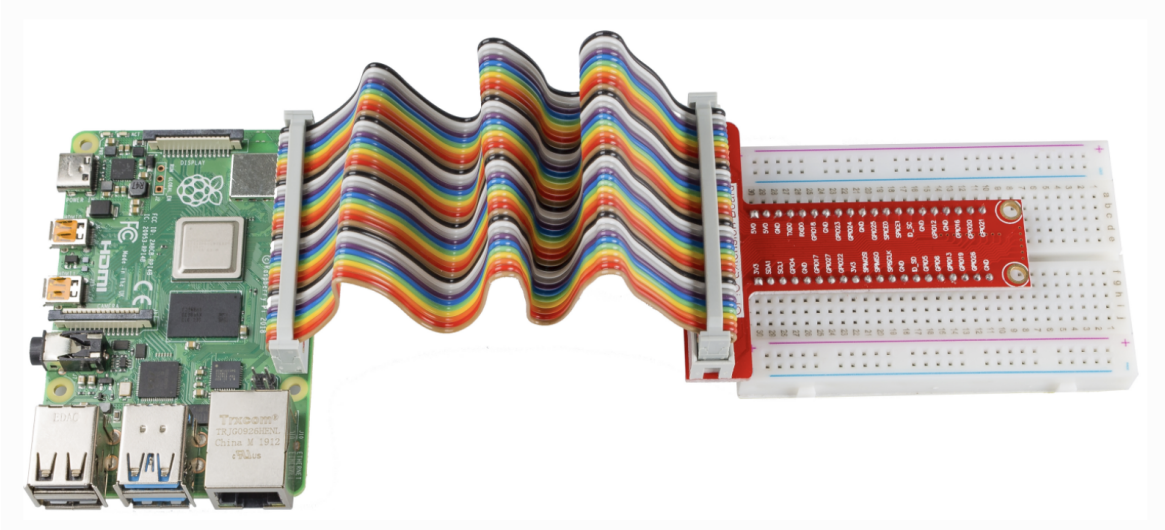

Figure 8: A GPIO extension board [5]

circuitry resembles that in the figure 10. WiringPi [54] is the CPP library for Raspberry Pi to interact with a GPIO extension board and Board [55] is a Python library for the same purpose.

The communication is done over TCP/IP with the help of socket programming. The `TempSensServer` is a bare-bone server which just displays the received information in a proper format. The client collects data for one minute before sending the accumulated data to the server. By default, it uses 512 bytes as chunk size.

Based off figure 7, the estimated power consumption of wifi adapter is `1.542` Watts. This is calculated after computing the mean average sum of different power modes for different current values corresponding to the voltage. In order to compute the optimal chunk size, the python script to determine the chunk size was run on the Raspberry Pi 4.

Figure 11 shows that energy consumption does not linearly depend on chunk size. If that were the case, we would have got a straight line. However, what we observe is a global minima and around 2^{14} on the x-axis, we see that's where the

Name	WiringPi	Board	BCM		Board	WiringPi	Name
		GPIO Extention Board					
3.3V	3V3	1	3V3	5.0V	2	5.0V	5V
SDA	8	3	SDA	5.0V	4	5.0V	5V
SCL	9	5	SCL	GND	6	GND	0V
GPIO7	7	7	GPIO4	TXD	8	15	TXD
0V	GND	9	GND	RXD	10	16	RXD
GPIO0	0	11	GPIO17	GPIO18	12	1	GPIO1
GPIO2	2	13	GPIO27	GND	14	GND	0V
GPIO3	3	15	GPIO22	GPIO23	16	4	GPIO4
3.3V	3.3V	17	3.3V	GPIO24	18	5	GPIO5
MOSI	12	19	MOSI	GND	20	GND	0V
MISO	13	21	MISO	GPIO25	22	6	GPIO6
SCLK	14	23	SCLK	CE0	24	10	CEO
0V	GND	25	GND	CE1	26	11	CE1
IN_SDA	30	27	EED	EEC	28	31	ID_SCL
GPIO21	21	29	GPIO5	GND	30	GND	0V
GPIO22	22	31	GPIO6	GPIO12	32	26	GPIO26
GPIO23	23	33	GPIO13	GND	34	GND	0V
GPIO24	24	35	GPIO19	GPIO16	36	27	GPIO27
GPIO25	25	37	GPIO26	GPIO20	38	28	GPIO28
0V	GND	39	GND	GPIO21	40	29	GPIO29

Figure 9: Naming methods for WiringPi and Board [5]

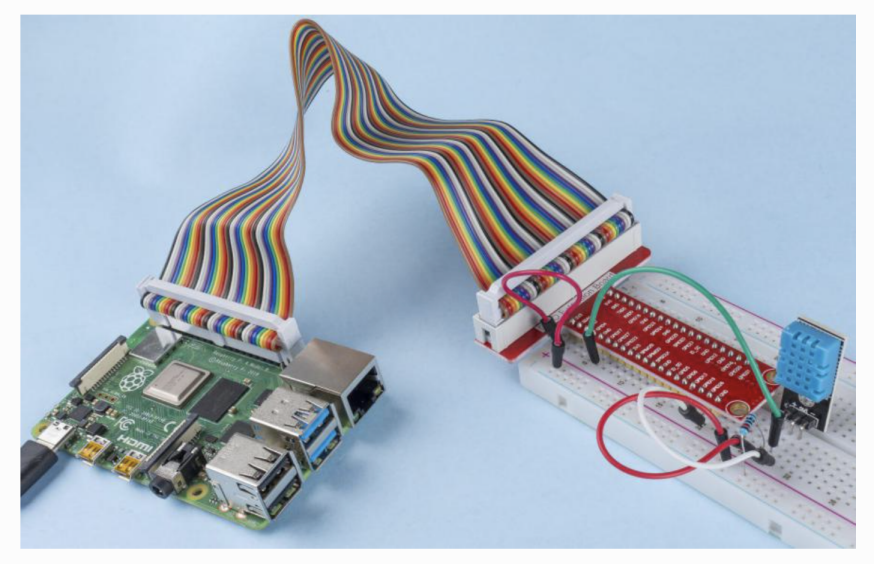

Figure 10: TempSens Hardware Configuration [6]

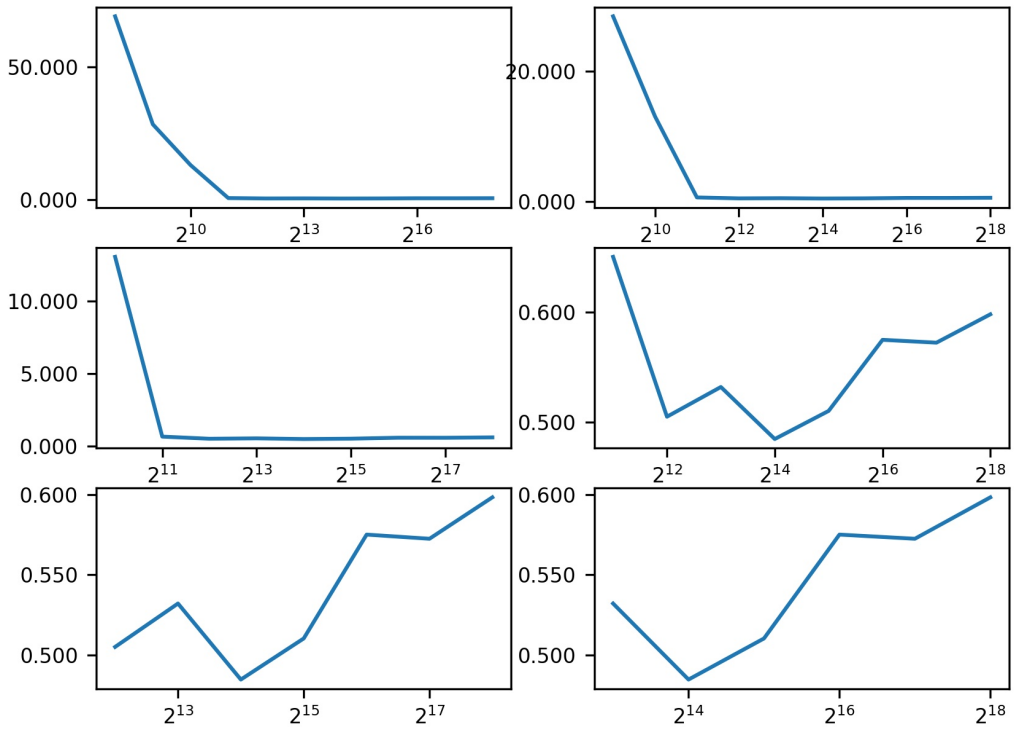

Figure 11: EPB for different chunk sizes (Chunk Size (x-axis) and Energy in microJ (y-axis))

lowest amount of energy is being consumed per byte. This suggests that for the `TempSens` app to have optimal energy consumption for communication, the preferred chunk size should be 2^{14} as it would consume the lowest amount of energy. It can also be seen that EPB at 2^{14} is significantly lower than that at the initial value of 2^8. Figure 11 shows multiple subplots each for a different range of chunk sizes. The idea behind this is to show that even though the energy usage looks pretty much the same when seen in the top-left and top-right plot, but when you actually zoom in and see the bottom-left and bottom-right plots, you realize that the energy usage is not that linear anymore. There's clearly a global minima present and it will be different for different devices due to varying hardware and software configuration but with our work we can definitely find it.

Based on the values obtained from this experiment, and then replace them with the default values in `TempSensClient`, the total amount of time for which battery life was prolonged solely due to saving energy in communication cost. was about 3 minutes. However, one other use-case of this approach is to be able to determine the intervals at which one should communicate data, if it allows. In the case of `TempSens`, its quite flexible as the information being communicated is not time sensitive and the use-case is not that critical. This suggests that an optimal interval would be just when the app collects enough amount of data (2^{14}). For this app, it turned out to be 120 seconds which is double the original interval break. Applying this optimization, we were able to prolong the battery time of one charge cycle by 7 minutes. Even though apparently it doesn't sound like much but keep in mind this is just with optimizing communication cost which is usually in terms of microJoules. Comparatively, that is a considerable improvement in the overall scheme of things. On the other hand, it would not have mattered much if the energy usage of the whole device was dependent on a sensor which consumes much more energy as compared to the wifi adapter as we will discuss it in the next experimental setup.

V.2 Experimental Setup —Fall Detection App

For this experiment, we test our approach on the Fall Detection App [56].

VI. Case Study

VII. Conclusion

APPENDIX SECTION

APPENDIX A

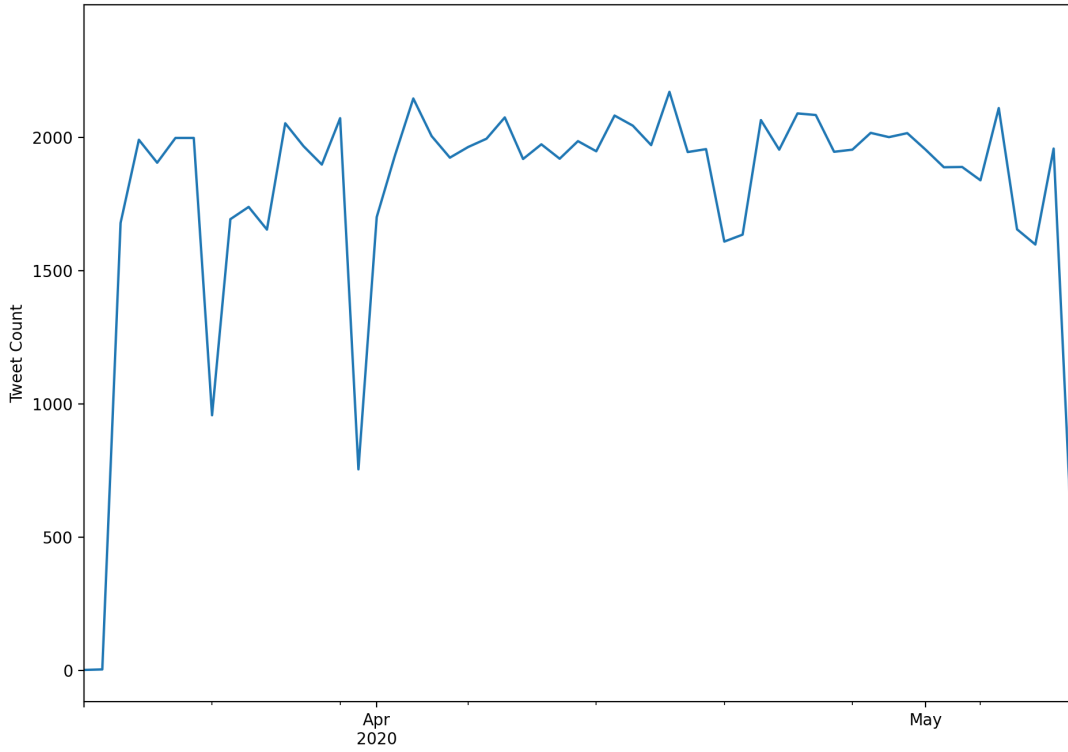

Figure A.1: First figure of appendix A. Captions of figures and tables in the Appendix section should not show up in the table of contents.

Table A.1: First table of appendix A. Captions of figures and tables in the Appendix section should not show up in the table of contents.

Dataset name	Number of records	Number of users
Dataset A	248	20
Dataset B	464	28
Dataset C	348	7
Dataset D	419	5
Dataset E	854	15

Table A.2: Second table of appendix A. Captions of figures and tables in the Appendix section should not show up in the table of contents.

Dataset name	Number of records	Number of users
Dataset F	1000	10
Dataset G	2000	20
Dataset H	3000	30
Dataset I	4000	40
Dataset J	5000	50

APPENDIX B

Table B.1: First table of appendix B. Captions of figures and tables in the Appendix section should not show up in the table of contents.

Column 1	Column 2	Column 3
1	2	10000
3	4	20000
5	6	30000

Table B.2: Second figure of appendix B. Captions of figures and tables in the Appendix section should not show up in the table of contents.

Column 1	Column 2	Column 3
A	B	10000
C	D	20000
E	F	30000

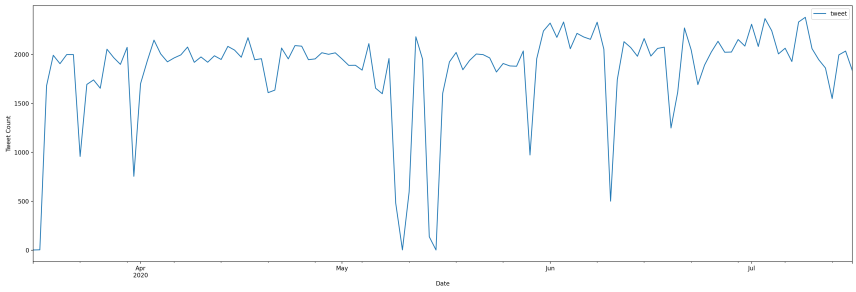

Figure B.1: First figure of appendix B. Captions of figures and tables in the Appendix section should not show up in the table of contents.

REFERENCES

[1] S. Pasricha, R. Ayoub, M. Kishinevsky, S. K. Mandal, and U. Y. Ogras, "A survey on energy management for mobile and iot devices," *IEEE Design Test*, vol. 37, no. 5, pp. 7–24, 2020.

[2] M. Schuchhardt, S. Jha, R. Ayoub, M. Kishinevsky, and G. Memik, "Caped: Context-aware personalized display brightness for mobile devices," in *2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 1–10, 2014.

[3] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber, "A vision of swarmlets," *IEEE Internet Computing*, vol. 19, no. 2, pp. 20–28, 2015.

[4] A. H. Ngu, J. Eyitayo, G. Yang, C. Campbell, Q. Z. Sheng, and J. Ni, "An iot edge computing framework using cordova accessor host," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[5] "Gpio extension board."

[6] "Gpio dht circuit."

[7] Y. Xiao, R. S. Kalyanaraman, and A. Yla-Jaaski, "Energy consumption of mobile youtube: Quantitative measurement and analysis," in *2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*, pp. 61–69, 2008.

[8] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[9] D. Linthicum, "Responsive data architecture for the internet of things," *Computer*, vol. 49, no. 10, pp. 72–75, 2016.

[10] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[11] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, 2014.

[12] J. Wu and W. Zhao, "Design and realization of winternet: From net of things to internet of things," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, Nov. 2016.

[13] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash, "Internet of things (iot): A vision, architectural elements, and security issues," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 492–496, 2017.

[14] W. Yu, H. Xu, H. Zhang, D. Griffith, and N. Golmie, "Ultra-dense networks: Survey of state of the art and future directions," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10, 2016.

[15] M. Agiwal, A. Roy, and N. Saxena, "Next generation 5g wireless networks: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016.

[16] P. Demestichas, A. Georgakopoulos, D. Karvounas, K. Tsagkaris, V. Stavroulaki, J. Lu, C. Xiong, and J. Yao, "5g on the horizon: Key challenges for the radio-access network," *IEEE Vehicular Technology Magazine*, vol. 8, no. 3, pp. 47–53, 2013.

[17] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, "Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems:," pp. 368–373, 08 2016.

[18] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack amp; cap: Adaptive dvfs and thread packing under power caps," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 175–185, 2011.

[19] G. Dhiman and T. S. Rosing, "System-level power management using online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 676–689, 2009.

[20] P. Bogdan and R. Marculescu, "Workload characterization and its impact on multicore platform design," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 231–240, 2010.

[21] U. Gupta, C. Patil, G. Bhat, P. Mishra, and U. Ogras, "Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs," *ACM Transactions on Embedded Computing Systems*, vol. 16, pp. 1–20, 09 2017.

[22] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1521–1526, 2015.

[23] F. M. M. u. Islam and M. Lin, "Hybrid dvfs scheduling for real-time systems based on reinforcement learning," *IEEE Systems Journal*, vol. 11, no. 2, pp. 931–940, 2017.

[24] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2019.

[25] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep q-learning approach for dynamic management of heterogeneous processors," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 14–17, 2019.

[26] S. Ross, G. J. Gordon, and J. A. Bagnell, "No-regret reductions for imitation learning and structured prediction," *CoRR*, vol. abs/1011.0686, 2010.

[27] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic vfi control in large-scale manycore systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[28] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "Dynamic resource management of heterogeneous mobile platforms via imitation learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2842–2854, 2019.

[29] X. Chen, J. Mao, J. Gao, K. W. Nixon, and Y. Chen, "Morph: Mobile oled-friendly recording and playback system for low power video streaming," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[30] M. Dong, Y.-S. K. Choi, and L. Zhong, "Power-saving color transformation of mobile graphical user interfaces on oled-based displays," in *ISLPED*, 2009.

[31] X. Chen, K. W. Nixon, H. Zhou, Y. Liu, and Y. Chen, "Fingershadow: An OLED power optimization based on smartphone touch interactions," in *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, (Broomfield, CO), USENIX Association, Oct. 2014.

[32] I. Constandache, S. Gaonkar, M. Sayler, R. Choudhury, and L. Cox, "Enloc: Energy-efficient localization for mobile phones," pp. 2716 – 2720, 05 2009.

[33] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy trade-off for continuous mobile device location," pp. 285–298, 01 2010.

[34] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," *Wireless Networks*, vol. 11, 09 2002.

[35] M.-R. Ra, J. Paek, A. Sharma, R. Govindan, M. Krieger, and M. Neely, "Energy-delay tradeoffs in smartphone applications," pp. 255–270, 06 2010.

[36] B. Kellogg, V. Talla, S. Gollakota, and J. R. Smith, "Passive wi-fi: Bringing low power to wi-fi transmissions," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 151–164, USENIX Association, Mar. 2016.

[37] F. Lu, G. M. Voelker, and A. C. Snoeren, "Slomo: Downclocking wifi communication," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, (Lombard, IL), pp. 255–258, USENIX Association, Apr. 2013.

[38] C. Chou, P. Nair, and M. K. Qureshi, "Reducing refresh power in mobile devices with morphable ecc," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 355–366, 2015.

[39] N. Vallina-Rodriguez and J. Crowcroft, "Erdos: achieving energy savings in mobile os," 01 2011.

[40] A. Javed, M. Shahid, M. Sharif, and Y. Mussarat, "Energy consumption in mobile phones," *International Journal of Computer Network and Information Security*, vol. 9, pp. 18–28, 12 2017.

[41] J. Flinn and M. Satyanarayanan, "Powerscope: a tool for profiling the energy usage of mobileapplications," pp. 2–10, 03 1999.

[42] N. G. Bachiega, P. S. L. Souza, S. M. Bruschi, and S. d. R. S. de Souza, "Container-based performance evaluation: A survey and challenges," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 398–403, 2018.

[43] J. Turnbull, *The Docker Book*. s.n., 2014.

[44] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614, 2014.

[45] S. Olabiyisi, E. Omidiora, F. Uzoka, and B. Akinnuwesi, "A survey of performance evaluation models for distributed software system architecture," 10 2010.

[46] "Cordovahost."

[47] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[48] M. Lohstroh, C. Menard, S. Bateni, and E. Lee, "Toward a lingua franca for deterministic concurrent systems," *ACM Transactions on Embedded Computing Systems*, vol. 20, pp. 1–27, 05 2021.

[49] Lf-Lang, "Overview Â· lf-lang/lingua-franca wiki."

[50] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of wifi and bluetooth in smartphones," *IEEE Transactions on Mobile Computing*, vol. 12, no. 7, pp. 1363–1376, 2013.

[51] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 92–101, 2013.

[52] A. Sabovic, C. Delgado, J. Bauwens, E. De Poorter, and J. Famaey, "Accurate online energy consumption estimation of iot devices using energest," pp. 363–373, 11 2019.

[53]

[54] "Reference: Wiring pi," May 2013.

[55] "Rpi.gpio."

[56] T. Mauldin, M. Canby, V. Metsis, A. Ngu, and C. Rivera, "Smartfall: A smartwatch-based fall detection system using deep learning," *Sensors*, vol. 18, p. 3363, 10 2018.