

# Santa Clara University

Department of Computer Science and Engineering

Date: June 5, 2022

I HEREBY RECOMMEND THAT THE THESIS PREPARED  
UNDER MY SUPERVISION BY

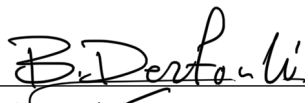
**Aastha Chawla, Nidusha Kannan, and Sreya Goyal**

ENTITLED


**EMT: Software-based Energy Management Tool  
for WiFi IoT Devices**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING**



Thesis Advisor  
Dr. Behnam Dezfouli



[N. Ling \(Jun 6, 2022 10:00 PDT\)](#)

Chairman of Department  
Dr. Nam Ling

# **EMT: Software-based Energy Management Tool for WiFi IoT Devices**

By

Aastha Chawla

Sreya Goyal

Nidusha Kannan

Submitted in Partial Fulfillment of the Requirements  
for the Degree of Bachelor of Science  
in Computer Science and Engineering  
in the School of Engineering at  
Santa Clara University,

June 5, 2022

Santa Clara, California

---

# Acknowledgments

We would like to thank our advisor Dr. Behnam Dezfouli for all of his guidance throughout the past year on this project. We would also like to thank our family, friends, and professors for all of their support during our time spent at SCU.

# **EMT: Software-based Energy Management Tool for WiFi IoT Devices**

Aastha Chawla

Sreya Goyalia

Nidusha Kannan

Department of Computer Science and Engineering  
Santa Clara University  
Santa Clara, California

June 5, 2022

## **ABSTRACT**

Many WiFi-based IoT (Internet of Things) devices rely on limited energy resources such as batteries or energy harvesting. Although monitoring and studying the energy consumption of these devices is essential, the use of external, hardware-based energy measurement tools is costly, non-scalable, and introduces many challenges regarding the connectivity of such tools with devices. In this thesis, we propose EMT, a novel tool to collect, analyze, and monitor the power cycles of IoT devices without the need for any external tools. The basic idea is to modify the WiFi Access Point's software to keep track of the power status of devices reported in packets. EMT also includes back-end and front-end components for data storage, analysis, and visualization. We demonstrate the effectiveness and features of EMT via empirical evaluations.

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>4</b>
2.1	IEEE 802.11 Power Saving Modes	5
2.2	Energy Monitoring Tools	7
2.3	Enhancing the Energy Efficiency of IoT Devices	9
<b>3</b>	<b>Proposed Solution: EMT</b>	<b>11</b>
3.1	Linux WiFi Subsystem	11
3.2	Accessing a Device's Power Status	12
3.2.1	Accessing Software Drivers	12
3.2.2	eBPF	13
3.3	Data Processing and Visualization	15
3.3.1	Use Case Diagram	16
3.3.2	Backend	17
3.3.3	Frontend	18
<b>4</b>	<b>Empirical Evaluation</b>	<b>21</b>
4.1	Testbed Setup	21
4.2	Testbed Evaluation	22
4.3	Percentage Awake Time	25
4.4	Duty Cycle	26
4.5	Number of Packets Transmitted and Received	28
<b>5</b>	<b>Conclusion and Future Work</b>	<b>31</b>
	<b>Bibliography</b>	<b>33</b>

---

# List of Figures

2.1	Operation of 802.11 PSM. The PS-Poll packet sent by the device informs the AP that the device has transitioned into awake mode and is ready for packet reception. . . . .	6
2.2	Operation of 802.11 APSD. The trigger packet sent by the device informs the AP that the device has transitioned into awake mode and is ready for packet reception. . . . .	7
2.3	The components of EMPIOT, referenced from [1]. . . . .	8
3.1	The AP architecture implemented to collect the power status of devices from the driver. . . . .	14
3.2	EMT's Use Case Diagram depicting how a user would utilize the EMT platform. . . . .	16
3.3	EMT's web-based dashboard interface. . . . .	18
3.4	EMT's web-based interface dashboard depicting the insights for device 1, a Ring Security Camera. . . . .	19
3.5	EMT's web-based interface dashboard depicting the insights for device 2, an iPhone 13 Pro. . . . .	19
4.1	The testbed setup for empirical evaluation of EMT. . . . .	22
4.2	EMT's landing page for the frontend dashboard. . . . .	23
4.3	EMT's landing page with a high level overview of how EMT works. . . . .	23
4.4	Login page . . . . .	24
4.5	EMT's dashboard of insights for the iPhone 13 Pro used in the experiment . . . . .	25
4.6	The duty cycle graph for the iPhone 13 used in our experiment. . . . .	27
4.7	The percentage awake time and packet transmission information for the iPhone 13, our test case device. . . . .	29

---

# List of Abbreviations

A-PSM	Adaptive PSM
AID	Association ID
AIDs	Association IDs
AP	Access Point
APSD	Automatic Power Save Delivery
APSM	Adaptive PSM
BI	Beacon Interval
eBPF	Extended Berkeley Packet Filter
EMT	Energy Management Tool
EOSP	End of Service Period
hostapd	host AP daemon
ICD	Implantable Cardioverter Defibrillators
IoT	Internet of Things
iPerf	Internet Performance Working Group
LI	Listen Interval
MLME	MAC Layer Management Entity
PMB	Power Management Bit
PSM	Power Save Mode
PSP	Power Save Poll
PSP-N	Power Save Poll with Null
RPi	Raspberry Pi
S-APSD	Scheduled-Automatic Power Save Delivery

SP     Service Period

TXOP   Transmit Opportunity

U-APSD   Unscheduled-Automatic Power Save Delivery



---

# CHAPTER 1

## Introduction

As technologies advance and wireless networks become more widespread, wireless devices are becoming the norm. The mobility that wireless devices offer makes wireless technology more favorable for Internet of Things (IoT) applications. Most IoT devices rely on limited energy resources (e.g., battery, energy harvesting) and wireless communication to offer ease of deployment and mobility, such as security systems, smart-home cameras, and other types of monitoring devices [2]. Such design considerations, however, come with specific issues and concerns. Unexpected or premature battery depletion is a result of inefficient energy consumption, which can often lead to alarming consequences. This issue is especially prominent in IoT devices used for time-sensitive or mission-critical applications, such as those utilized towards medical monitoring. In 2016, for example, St. Jude Medical recalled thousands of heart monitoring devices, after 2 patients died when their devices' batteries suddenly failed, within 24 hours of being implanted, when they should have lasted for at least the next 5-7 months [2]. These heart monitoring devices were Implantable Cardioverter Defibrillators, or ICDs. They are a type of IoT device used to send electrical shocks to a human heart to restore the normal heartbeat. These are used to treat patients with heart arrhythmias. This disaster was shocking and incredibly disheartening, and it was also a costly setback for St. Jude Medical [2]. It served as an eye-opening case and wake-up call to the life threatening consequences of unexpected battery depletion in battery-operated devices. The rapid changes in

innovation and the increased processing and communication demands of IoT devices often result in excessive energy consumption [3]. Batteries prove to be an unreliable and irregular source of energy, and a major issue lies in the fact that IoT device users are unaware of issues with their device’s batteries. There remains a need for a solution to this problem, so that a case like this can be avoided in the future.

Due to the low-cost and widespread deployment of WiFi Access Points (APs), many IoT devices rely on the 802.11 standard for wireless connectivity. The standard provides mechanisms that allow devices to switch between sleep and awake mode, depending on their communication pattern, to enhance the energy efficiency of connected devices. However, the efficiency of such methods is highly affected by various factors such as traffic rate and interference [4, 5]. For example, as the number of devices connected to an AP increases, the amount of traffic exchanged increases as well, and each device needs to spend more time in the ‘awake’ state to communicate with the AP [6, 7]. Another factor that comes into play is the distance between a device and the AP. The greater the distance, the higher the number of retransmissions by the device, resulting in increased energy consumption.

In this thesis, we propose EMT, a software-based, real-time energy management platform for WiFi IoT devices<sup>1</sup>. Instead of connecting additional hardware to IoT devices, EMT relies on the fact that devices must inform the AP whenever they change their power status (i.e., sleep to awake, and awake to sleep). Therefore, the AP’s WiFi driver keeps track of the power status of all the devices. EMT extracts this information from the driver by using eBPF technology and kernel functions. Additionally, EMT provides a web-based interface for real-time monitoring and analysis of devices’ energy consumption trends. We then perform an empirical evaluation of energy consumption using real-world wireless devices through a testbed

---

<sup>1</sup>The implementation of EMT is available at the following link: <https://github.com/SIOTLAB/SEMTI>

simulation, in order to demonstrate the effectiveness of EMT. The testbed allows for generating varying network traffic through various device configurations, and mimicking real-world network use cases. Upon gathering this energy consumption data for multiple concurrently connected devices, we then apply data analysis techniques to extract energy usage insights. We visually display these insights on EMT’s frontend dashboard. This low-cost and viable solution helps users monitor the energy consumption of their devices and make changes to their network configurations based on EMT’s insights.

The rest of this thesis is organized as follows. We review related work around IEEE 802.11 Power Save Mode protocols in Chapter 2. We then present EMT in Chapter 3 and describe the essential technologies used, including eBPF and hostapd. We perform an empirical evaluation of our work via our testbed setup and discuss our results in Chapter 4, and conclude our paper in Chapter 5.

---

## CHAPTER 2

# Related Work

It is important for implantable and wireless medical devices to be energy efficient. They cannot unexpectedly run out of power because they are used for highly time-sensitive data transfer, and they cannot easily be removed and placed for charging. Such medical devices are also used for mission-critical data transfer, and must be extremely reliable. Since data transmission in WiFi networks is based on contention-based channel access, devices must compete with other network devices to grasp the network channel and successfully transmit their data. This is unacceptable for a device that must quickly and reliably transmit critical data. Therefore, wireless medical devices must be made energy-efficient, as this directly affects the life span of the device. It must also be made reliable, as this directly affects the efficacy of the data transmission of the device.

In this chapter, we overview the relevant research that has been conducted towards more energy-efficient power consumption and reliable data transmission. The existing 802.11 Power Save Mode (PSM) protocol aims to reduce devices' energy consumption by modifying the time they spend in sleep or listen-state. The current 802.11 Automatic Power Save Delivery (APSD) protocol builds off of PSM, but aims to reduce the excess delays caused by PSM and hence is more time and energy-efficient. We then examine existing hardware efforts to monitor and track the energy consumption of IoT devices by looking at external power management tools. Finally, we examine power management schemes that run directly on the IoT device in order

to enhance the device's energy efficiency.

## 2.1 IEEE 802.11 Power Saving Modes

The two main power saving mechanisms of IEEE 802.11 are PSM and APSD.

PSM allows devices to save energy by switching between the time the device spends in sleep state and awake state [8]. During the awake state, the device spends time actively listening and communicating with the AP. In sleep state, it does not communicate with the AP. Figure 2.1 shows the exchange of frames when a device uses PSM. While the device is in sleep state, the AP buffers all the packets that it receives for that device, because the device is not receptive to any communication at that time. The AP sets a certain flag in the *beacon* frames for each device, whenever it has buffered packets for a specific device. This marks whether or not the device needs to receive any packets after it wakes up. All connected devices are required to wake up at certain intervals to receive beacon packets. When this beacon packet is received, the device sends a PS-Poll packet to the AP, to inform it that it has transitioned into awake mode. This signals that the device is ready for data reception, and is looking to receive any packets buffered by the AP while it was asleep. As soon as the AP has received this confirmation that the device is awake, it transmits the packets to the device [9]. As long as the packet received from the AP has the *more data* bit flag set, the device needs to keep sending PS-Poll packets to the AP in order to retrieve the buffered packets. A device using PSM spends significantly more time and energy contending for channel access and sending multiple PS-Poll packets. This is because there is an increase in the number of devices and amount of traffic transmitted and received by devices connected to the AP.

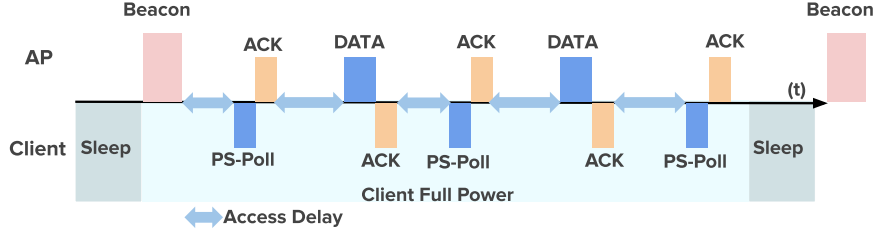


Fig. 2.1: Operation of 802.11 PSM. The PS-Poll packet sent by the device informs the AP that the device has transitioned into awake mode and is ready for packet reception.

APSD (defined in 802.11e amendment) allows the AP to buffer packets for devices in sleep mode, similar to the packet buffering that occurs in PSM. Figure 2.2 presents an example of this. However, unlike PSM, APSD does not wait for a beacon frame to arrive from the AP to trigger the delivery of the buffered packets. Instead the device can wake up at any time to *trigger* the AP to receive its buffered packets [9]. When the AP is notified that the device is awake, all the buffered packets can be delivered to the device without forcing the device to send multiple PS-Poll packets to retrieve the currently buffered packets. By removing the delay caused by the device repeatedly polling the AP, APSD is more time and energy-efficient than PSM.

APSD can be ineffective in situations with greater data transfer rates. Since devices using PSM only receive their data once in a Beacon Interval, the AP is able to buffer a greater amount of data before transmitting it. APSD devices, however, trigger the AP more often than PSM devices. Due to this, the AP is unable to account for larger amounts of buffered data or packet aggregation activity [9]. This makes it ineffective during scenarios with traffic congestion.

A-PSM is another power saving method that is used to reduce communication delay while preserving energy. A-PSM is an enhancement of APSD; it allows a device to stay in awake mode for a short duration after the last packet is exchanged with the AP. This short duration is called *tail time*.

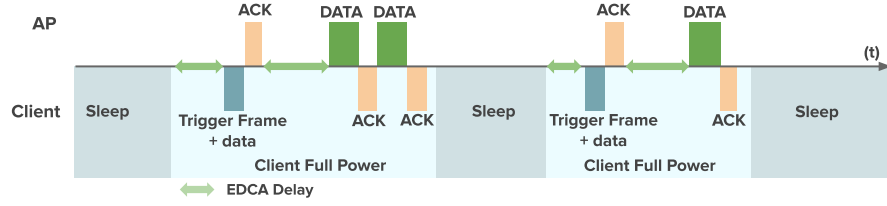


Fig. 2.2: Operation of 802.11 APSD. The trigger packet sent by the device informs the AP that the device has transitioned into awake mode and is ready for packet reception.

In all the IEEE 802.11 power-saving protocols, the device relies on a special flag, called the *Power Management Bit*, or PMB. The PMB informs the AP whether the device has transitioned to sleep mode or awake mode. When a device using one of the IEEE 802.11 power saving methods wakes up, it sends a packet with the PMB set to zero. At the end of the data exchange, when the device goes back to sleep, it send a packet with the PMB set to one.

Since most wireless devices are inefficient and battery-powered, fast data transmission and energy-efficient networking are critical. However, current WiFi technologies such as PSM and APSD either compromise energy-efficient power consumption methods to attain real-time and reliable data transmission, or vice versa. WiFi networks cannot guarantee real-time, reliable, and energy-efficient transmission of data all at once, which means they cannot adequately meet the needs of wireless devices and the networks they belong to.

## 2.2 Energy Monitoring Tools

In addition to the previous software driven solutions, a straightforward hardware approach to monitoring and studying the energy consumption of IoT devices is to use an external power management tool.

External devices like oscilloscopes can be used to analyze the energy consump-

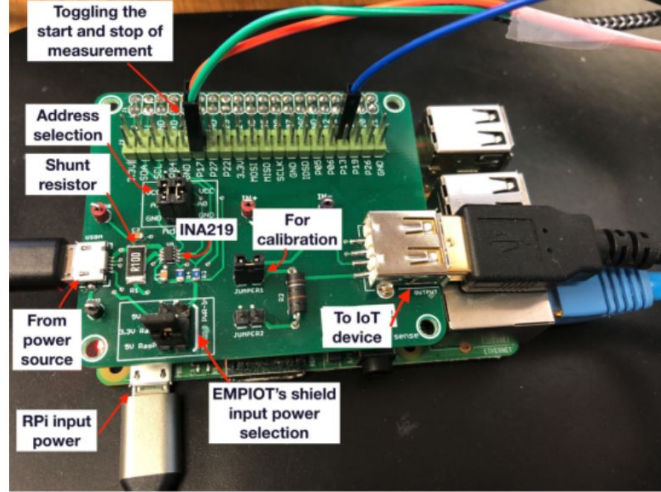


Fig. 2.3: The components of EMPIOT, referenced from [1].

tion of WiFi devices [10]. A current probe can also be used to detect the magnetic field and perform energy measurements, as the authors in [11] mention. However, with the added hardware and usage of costly technology, such solutions can be both bulky and expensive.

To resolve this problem, low-cost and more miniature power measurement tools have been developed. For example, Dezfouli et al. [1] proposed an energy measurement platform for wireless IoT devices. EMPIOT is an external hardware tool capable of collecting both voltage and current samples. This platform externally connects to a singular IoT device and relies on a low power source to function, as seen in Figure 2.3. If we just connect an external power management tool, like EMPIOT, to each device, we would easily get that device's energy usage readings. This is a great solution for one or two devices. But what if we were to have twenty IoT devices connected to twenty different power management tools, or a hundred IoT devices connected to hundred power management tools. This is expensive, and simply not scalable. Additionally, external power management tools cannot easily integrate with existing IoT devices, like smoke detectors or surveillance cameras. This is because the external tool must be attached to the input power boards and



circuitry of the device, which are usually not available or exposed in commercial tools. Thus, the scope of external power management tools becomes limited to only the devices used for research and development purposes, which have easily accessible circuit boards.

Although external power measurement tools are a simple solution for monitoring a few devices, they do not scale for hundreds or thousands of IoT devices. This solution quickly becomes expensive and infeasible, making it impractical to be considered a standard solution for the power monitoring needs of the billions of IoT devices on the market. Considering the sheer size of the IoT device market and usage today, we need a solution that is inexpensive, scalable, and viable.

## 2.3 Enhancing the Energy Efficiency of IoT Devices

Apart from external hardware tools to attach to IoT devices, the research community has also proposed various methods that work to enhance the energy efficiency of the IoT device itself.

Rozner et al. [12] propose *NAPman*, which works by prioritizing the delivery of PSM traffic, and it ensures that other devices on the network are not affected by the prioritized PSM traffic of one device. They also propose dynamic beacon interval adjustment per device, by modifying the device's drivers. Sheth and Dezfouli [13] propose an AP-based mechanism for packet scheduling, which is called *Wiotap*, which works for IoT devices that utilize A-PSM. In order to capitalize the chance of packet delivery before tail-time expiry, *Wiotap* uses an Earliest Deadline First (EDF) scheduling strategy. Gawande et al. [14] dynamically obtains information including the power level, channel status, and the type of network service per user. They then group these statistics based on their network effects, and utilize

these groupings to implement power-saving mechanisms in devices that are specifically tailored towards the information previously obtained. Lim et al. [15] propose a scheme that manages the power of the device, by utilizing reinforcement learning techniques that dynamically adjust the listen-intervals of mobile devices. They manage trade-offs between energy and delay by analyzing the variables produced by changing network conditions and configurations. Peck et al. [16] propose PSM-AW (PSM with adaptive wake-up), which includes a metric called PSM penalty to formalize the energy-delay trade-off. Server delay can be defined as the delay between sending a request to a server and receiving a reply back. In order to achieve the desired trade-off, the sleep duration dynamically adjusts based on the delay variations of server delay. Li et al. [4] propose multiple methods that aim to improve the energy efficiency of WiFi devices that are used towards industrial monitoring applications. The EMT platform proposed in this work incorporates existing power-saving methods and facilitates the development of novel energy-monitoring solutions.

---

## CHAPTER 3

# Proposed Solution: EMT

In this section, we present EMT, a novel, scalable, low-cost, and entirely software-based energy monitoring platform. EMT is capable of monitoring multiple devices that are all concurrently connected to the AP, and it eliminates the need for external hardware-based energy monitoring tools. Using the data it logs from the devices, it extracts insights about the devices' energy consumption. This information is then presented to the user via a web interface. For instance, if the battery usage of one of the connected devices is suddenly unexpectedly high, the user will be able to detect this inconsistency via the insights presented on EMT's web interface. Also, in a case where the energy consumption of a device is being negatively affected by interference or excessive channel utilization by other devices, the network manager can quickly and conclusively identify the problem and prevent premature battery depletion. The following sections present the technologies used, design, and implementation of EMT.

### 3.1 Linux WiFi Subsystem

hostapd [17] is a user-space daemon that runs on an AP, which handles the functionality of authentication, association, and disassociation of WiFi devices. For EMT, we rely on the logs generated by hostapd to obtain this information from the AP, and construct a timeline of the devices that were connected and disconnected from

the AP. `hostapd` configures `mac80211` and the driver via the `cfg80211` APIs, in order to generate control and management frames. We are able to utilize the `mac80211` module to understand the interface between the driver, `hostapd`, `qdisc`, bridge, and other components of the Linux networking subsystem. The `mac80211` module also administers the MAC Layer Management Entity (MLME) functions for SoftMAC drivers. These sample functions are building MAC headers and assigning sequence numbers. SoftMAC drivers implement a portion of layer-2 functionalities in the software. To do so, they must make use of the hardware and software resources of the host system. Only the time-critical MAC functions, which includes ACK transmission, inter-frame spacing, and channel access backoff, are implemented in the WiFi NIC. Finally, the WiFi driver is responsible for packet aggregation to the WiFi NIC for transmission on the channel.

We use `hostapd` in EMT to convert a Linux machine into a wireless access point. This helps us create an exclusive network of IoT devices, from which we collect power consumption patterns and their associated data in a controlled environment.

## **3.2 Accessing a Device's Power Status**

### **3.2.1 Accessing Software Drivers**

The AP's WiFi driver keeps track of the power status of the devices connected to the AP by maintaining a data structure of the relevant information. The device's power status is a vital observation that we rely on in order to collect this information from the driver. There are two approaches we could utilize in order to interact with the driver.

A seemingly straightforward solution is to directly modify the AP's kernel and

WiFi driver, and retrieve the relevant information and data structures from there. However, directly modifying a device’s kernel could cause kernel instability. It is also not time-efficient as it would require modifications whenever the network configuration is changed or updated. While we would be modifying the power management information, our changes could indirectly affect the modules and drivers that interact with this information. Additionally, it is not a scalable solution for networks of tens or hundreds of IoT devices, as we would have to manually make changes to all the relevant drivers and kernels.

The alternative solution builds off of our discussion in Section 2.1. When a device switches to awake mode, its PMB flag is set to 0, and when the device switches to sleep mode, the flag is set to 1. EMT utilizes this concept to understand the operational status of each device at any given point in time. In order to access the status of the device’s PMB, we need to interact with the device’s kernel, where the PMB information is stored. However, directly modifying the kernel could have consequences on the rest of a device’s operation or functionality, and doing so would also be costly in terms of time and resources. A preferred approach is to indirectly interact with the kernel to access the device’s PMB status. To this end, EMT leverages the eBPF technology. By utilizing eBPF to interact with the device’s software driver, we are able to establish a hook with the device’s kernel from its user space, without making any modifications to the kernel code itself.

### **3.2.2 eBPF**

Through eBPF, we are able to use probe events to interact with the kernel while remaining in the user space. eBPF’s user-defined logic is managed by probe events in the kernel code, which serve as markers for its execution. We use a type of probe

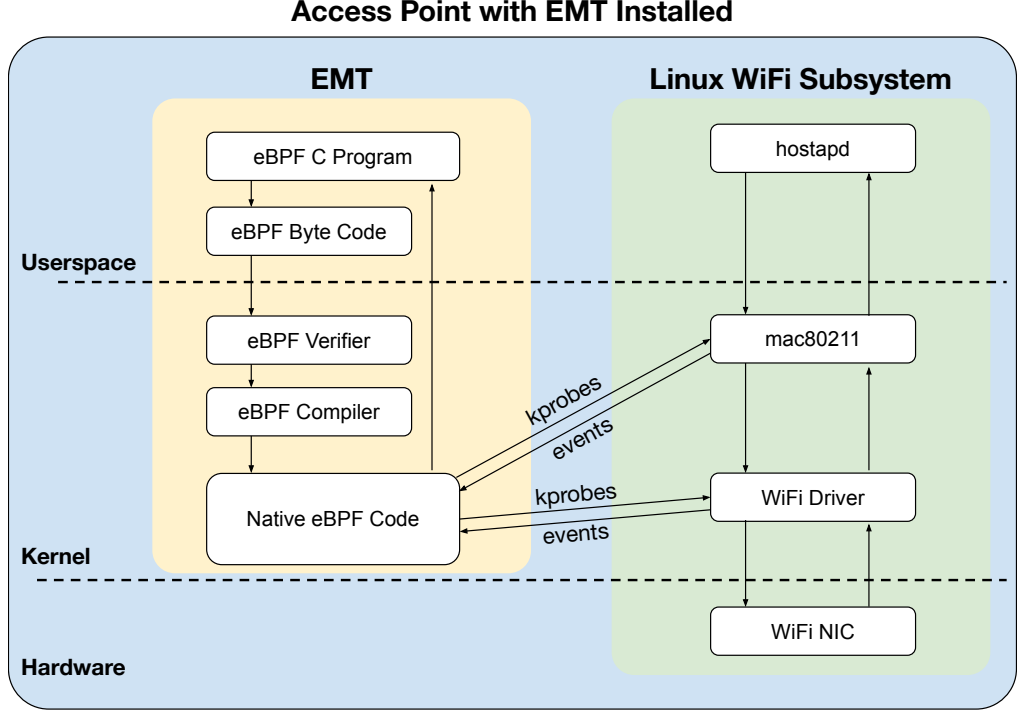


Fig. 3.1: The AP architecture implemented to collect the power status of devices from the driver.

event called *kprobes*; these are already defined in a kernel's symbol table. The left-half of Figure 3.1 depicts how EMT utilizes eBPF and the kernel components of a device [6]. By allowing us to execute user-defined logic, eBPF facilitates the run-time patching of the kernel image through function calls or syscalls. High level languages can be used to create eBPF programs, which are compiled into byte code via the LLVM-clang compiler [18].

eBPF programs are attached to probe events (i.e., kprobe or a tracepoint) that mark the instantiating points for the execution of user-defined logic. At the time of a system or function call, a breakpoint instruction is inserted by the eBPF system at the address of the currently executing function or instruction. As soon as the kernel execution reaches the breakpoint instruction, the system saves information about the currently executing code, and begins executing the user-defined logic in the eBPF program. The instruction that was replaced by the breakpoint is executed

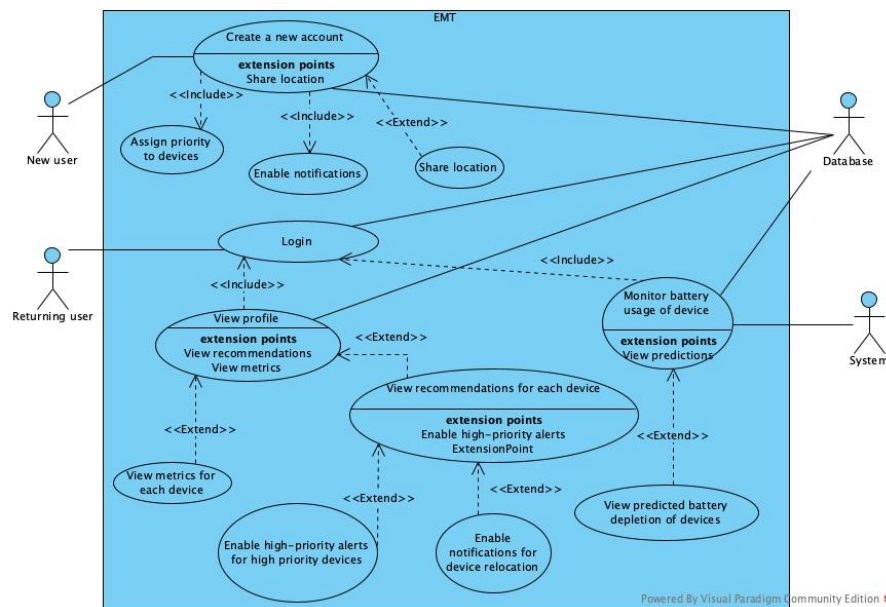
by the kprobe after the user-defined logic has completed execution [19]. After this has completed, the kernel can resume normal system execution.

In 802.11 networks, the power-save bit informs an AP about each device’s power state transitions. This bit transitions from low-power sleep state to active awake mode, and vice-versa. The AP uses this information to buffer the packets it receives for a device in sleep state, which it continues to do until the device transitions to active mode. In order to understand and analyze the power states of all the devices connected to an AP, EMT utilizes eBPF and logs the channel activity by attaching kprobes to the following functions: (i) `ieee80211_rx_napi()` is called when the AP receives a packet. We log the power-save bit inside any NULL or data packets received by the AP to understand what the power status of the device is, and whether it is in sleep or awake mode. (ii) `ath_tx_process_buffer()` is called upon receiving a layer-2 ACK. This indicates that packet has been successfully transmitted, and it helps us determine the number of retransmissions. (iii) `ieee80211_beacon_add_tim()` is responsible for beginning the generation and transmission of beacons in the WiFi network stack. We can monitor and record these beacon announcements by keeping a record of the instances when this function is called. Therefore, in order to calculate the energy consumption insights, we monitor the *uninformed* device wake-up times by keeping track of all the beacon announcements.

### 3.3 Data Processing and Visualization

EMT aims to be an intuitive, user-centric, and user-friendly platform. Its goal is to enable the user to quickly and seamlessly track the energy consumption insights of their IoT devices. We conceptualize this through the Use Case Diagram in Section

3.3.1. EMT’s functionality is split between two parts. To use EMT, a user connects their IoT devices to an AP that has downloaded and installed the EMT backend. The backend tracks, logs, and analyzes a device’s energy consumption information. The backend’s functionality is further discussed in Section 3.3.2. Then, EMT’s backend processes the data transfer information of all the connected devices and display the insights to a user on a web-based interactive frontend interface. We dive deeper into the frontend components of EMT in Section 3.3.3.



The *Use Case Diagram* in Figure 3.2 represents the functionality of the application that users can use to view and monitor energy metrics for their devices. It illustrates the interactions between various actors, including those that are internal and external to the system, and the use cases within the system. New users are



able to create an account and store their information in the database, and then pair their IoT devices with EMT. Returning users will be able to login and interact with the application to view their device recommendations and metrics. The insights that EMT offers will allow users to observe when battery depletion is predicted in one or more of their devices.

### 3.3.2 Backend

After the user connects their IoT devices to the AP, the energy monitoring begins. In order to determine the timestamp of when a device connects to the AP, EMT logs the eBPF and hostapd records and processes these logs. We begin logging the timestamps for the low-level events discussed in Section 3.2.2 once the device has been associated. After which, the user is able to monitor the devices' data transfer information.

EMT generates hostapd and eBPF logs for all the devices connected to the AP simultaneously. EMT then initiates scripts to separate the logs for each of the concurrently running devices into a log file per device. EMT chronologically organizes these logs to form a timeline of events that occurred while the devices were connected to the AP. Let's say, for instance, a user concurrently connects a variety of IoT devices to their AP such as phones, surveillance cameras, smoke detectors, etc. By monitoring the devices that are connected simultaneously and then processing their logs individually, we can determine the device's power consumption information in relation to and in the context of other devices that were connected simultaneously.

Each device's *duty cycle* is the most critical insight that we aim to collect from these logs. The ratio of a device's awake time to the total time it spends awake and

asleep is how the duty cycle for each device is calculated. We process and analyze the timestamps of when the device was awake vs. when the device was asleep and correlate these to beacon frames, packet transmissions, and reception instances in order to ascertain the duty cycle for a particular device.

### 3.3.3 Frontend

The frontend facilitates EMT's main goal, which is to be a user centric platform where a user can easily track the energy consumption of their IoT devices.

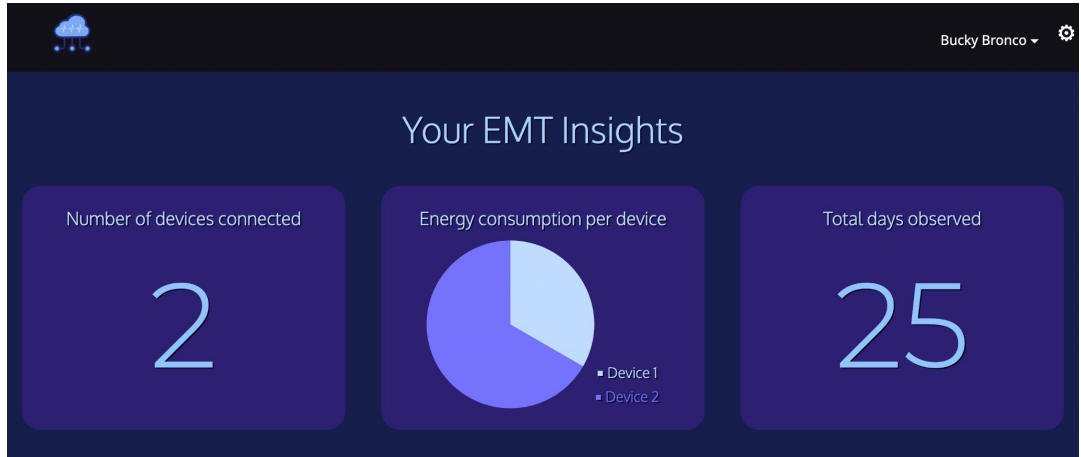


Fig. 3.3: EMT's web-based dashboard interface.

We developed a web-based interface to visually display device information and statistics computed in the backend.

When a user logs into their EMT account and begins pairing one or more of their devices with an AP on their network, EMT begins data collection for the devices and displays an energy consumption dashboard for each device. The user can then monitor the health and energy usage of their device using this interface. The overall insights for all a user's connected devices are depicted in Figure 3.3. The user can observe the number of devices connected to the AP and also view how long these devices have been monitored. The user can also observe how energy is

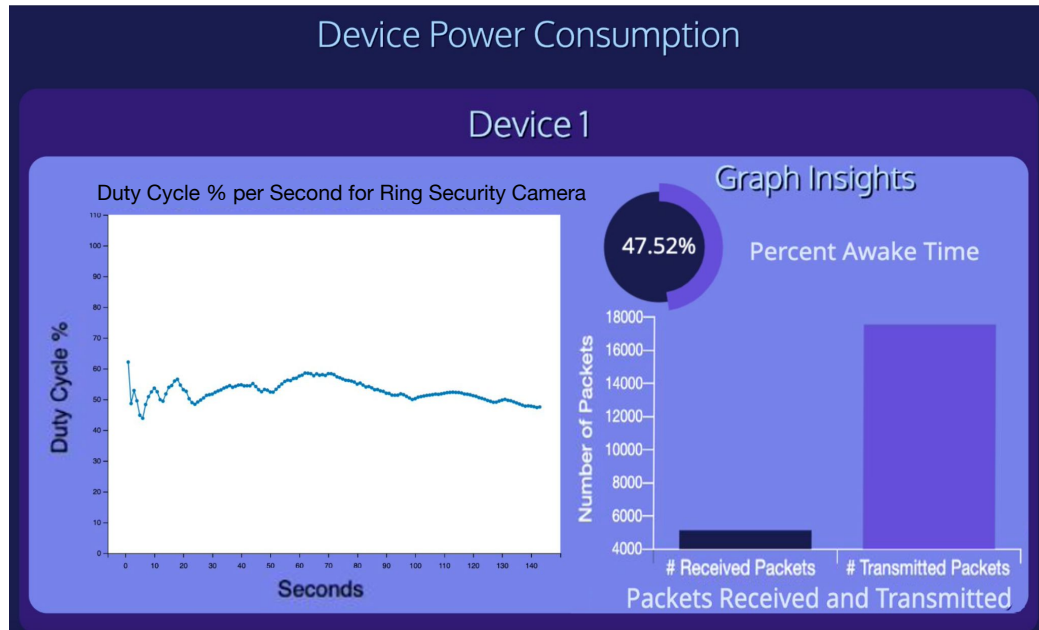


Fig. 3.4: EMT's web-based interface dashboard depicting the insights for device 1, a Ring Security Camera.

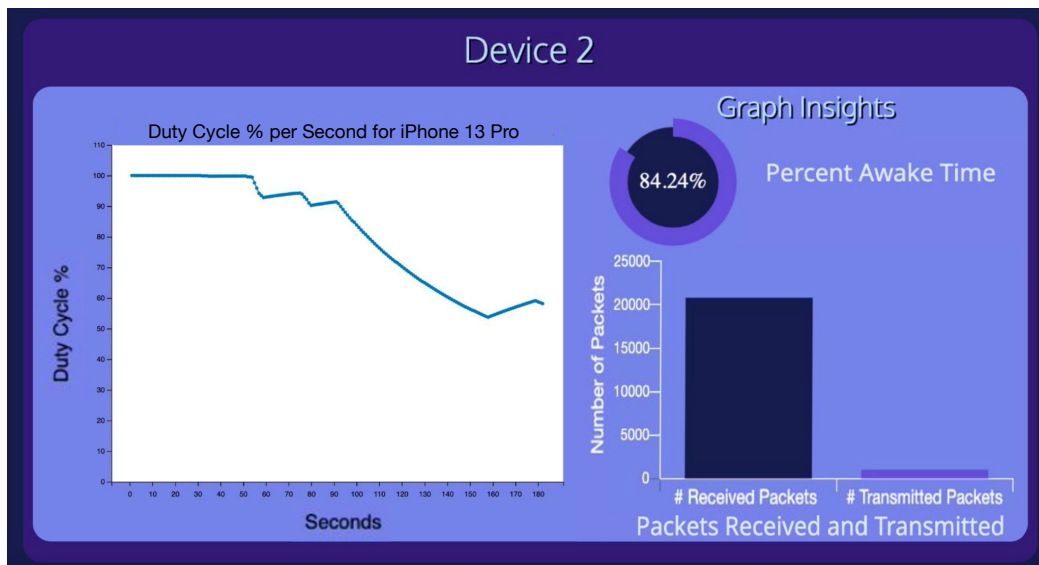


Fig. 3.5: EMT's web-based interface dashboard depicting the insights for device 2, an iPhone 13 Pro.

consumed and distributed among the devices, making it simple to figure out which devices uses the most energy.

Figure 3.4 and 3.5 depict a dashboard which consists of device-specific insights

for each device that the user connects to their AP. Each device’s dashboard includes insights about the percentage awake time for the duration that the device was monitored, the percent duty cycle per second that the device was monitored, and the total number of packets that the device transmitted and received during the time frame that it was connected to the AP.

The interactive graph of a device’s energy consumption insights can be zoomed into and out of, to better interpret the insights over particular monitoring periods. Users can then analyze and interpret parameters that may effect the energy consumption, such as the number of devices concurrently connected or the distance between the device and the AP. This can also help a user interpret and detect abnormal energy consumption events. We will explore this analysis further in Chapter 4 in our testbed evaluation.

---

## CHAPTER 4

# Empirical Evaluation

In this section, we perform an empirical evaluation of EMT, which we accomplish using a testbed. We describe the components of the testbed, how we mimic real world traffic and network configurations, and then discuss our results.

### 4.1 Testbed Setup

The *Testbed* in Figure 4.1 shows the setup used for evaluating EMT. The AP is a single-board computer with an Intel i7 processor (4 cores) that has 512GB SSD and 16GB RAM. The AP is connected to a layer-2 switch through a wired connection, which is connected to the internet.

We use two groups of devices in order to accurately replicate a realistic network setup. In the first group, we use IoT devices including security cameras, smartphones, and wearable devices. The second group of devices is comprised solely of Raspberry Pis. In order to generate realistic network traffic from the Raspberry Pis, we run Internet Performance Working Group (iPerf) on the devices. By integrating both groups of devices in our testbed, we can emulate various types of real network traffic for testing purposes. As both device groups are simultaneously connected to the AP, it receives the data transmitted by both groups of devices. The AP then analyzes the performance of the connected devices by correlating their sleep and awake times with the amount of data it received.

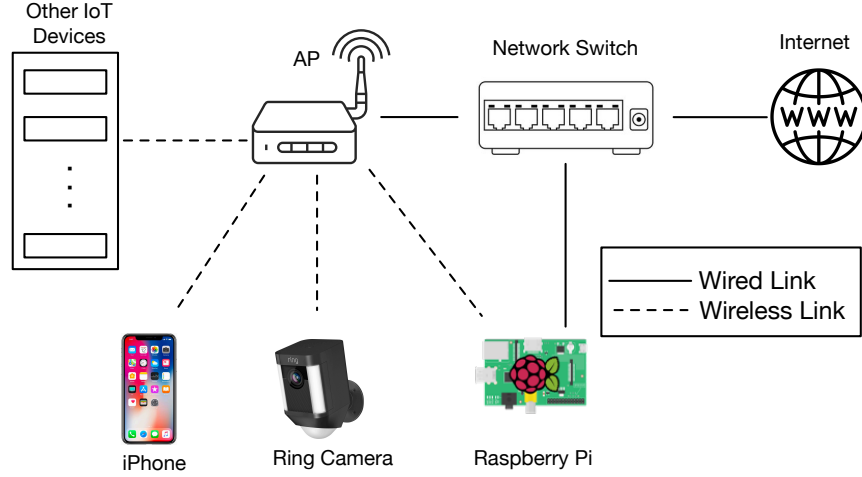


Fig. 4.1: The testbed setup for empirical evaluation of EMT.

We can obtain a metric for how long the device must remain awake in order to operate normally, since the testbed setup is a controlled environment. This metric relates to how much power and energy the device typically consumes. We can determine a normal range for the device’s duty cycle as we monitor it for longer periods of time, which serves as a more accurate benchmark of the power consumed. If irregularities are detected in the duty cycle, they are easily detected from EMT’s insights. A cause of these irregularities could be due to transmission interference as a result of the device’s location, or retransmission of data from a device because another device was monopolizing the network bandwidth. Essentially, if a device connected to the AP is having irregularities in its battery consumption, it would be depicted in the duty cycle that EMT calculated.

## 4.2 Testbed Evaluation

As discussed in section 4.1, our testbed setup includes an access point with the wired ethernet connection to a network switch, and our IoT devices ready to be

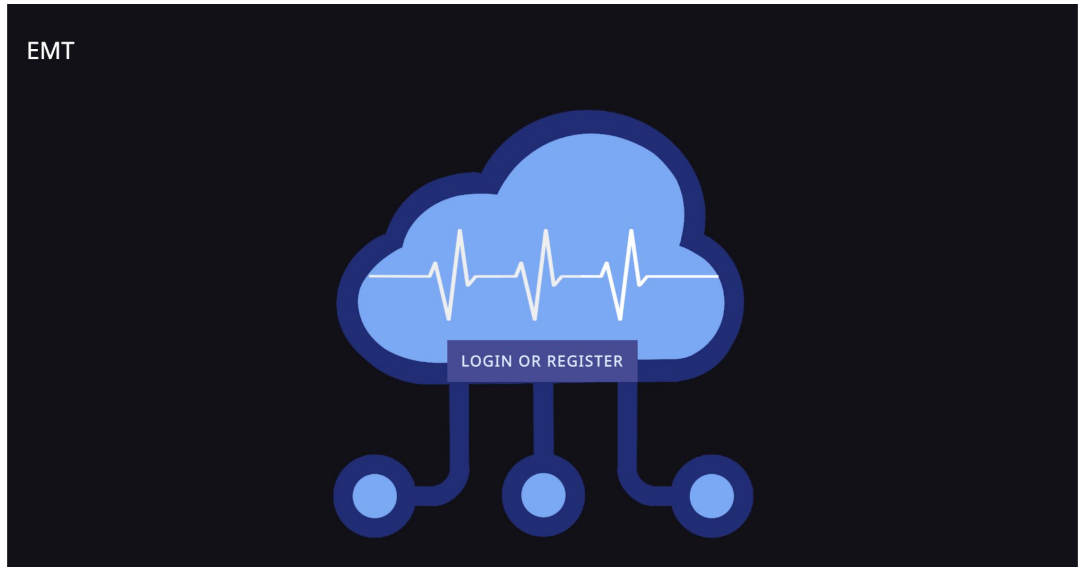


Fig. 4.2: EMT's landing page for the frontend dashboard.



Fig. 4.3: EMT's landing page with a high level overview of how EMT works.

wirelessly connected to the AP. When a user installs EMT on their AP, all they will need to do is connect their IoT devices to the WiFi network. Once we connect a device to the AP, EMT begins logging timestamps and tracking key insights about the device's energy consumption.

In a test case, we used the device for a short period of time. We tested out a bandwidth-intensive activity by streaming a video on YouTube, then turned the device off for about 15 seconds. We then turned it back on to check the news on

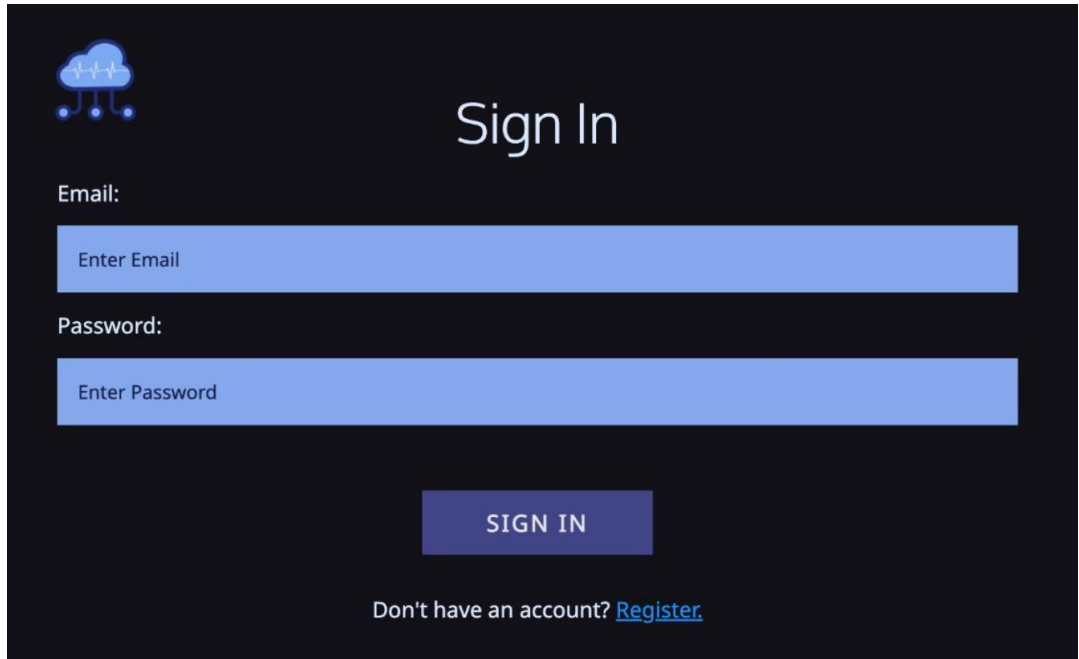


Fig. 4.4: Login page

the browser. Finally, we disconnected the device from the AP by turning off the WiFi. EMT's backend generated data insights for the user to interpret their energy consumption. These insights and statistics were then populated on EMT's frontend dashboard. Upon launch, the frontend includes some information about EMT and why it is important, such as in Figure 4.3. When the user opens EMT's frontend as seen in Figure 4.2, they are asked to either sign up or login into their EMT account. Figure 4.4 depicts the login page. Once a user logs in, they are taken to their personalized dashboard.

As described in Section 3.3.2, we retrieve timestamps and the device's data transfer information by processing the logs collected while the device was connected to the AP. We examine these timestamps to determine three main insights about the device's data transmission and energy consumption, which we then display to the user on the frontend dashboard, as in Figure 3.4 and 3.5. These insights include the percentage awake time, the duty cycle percentage per second, and the number of



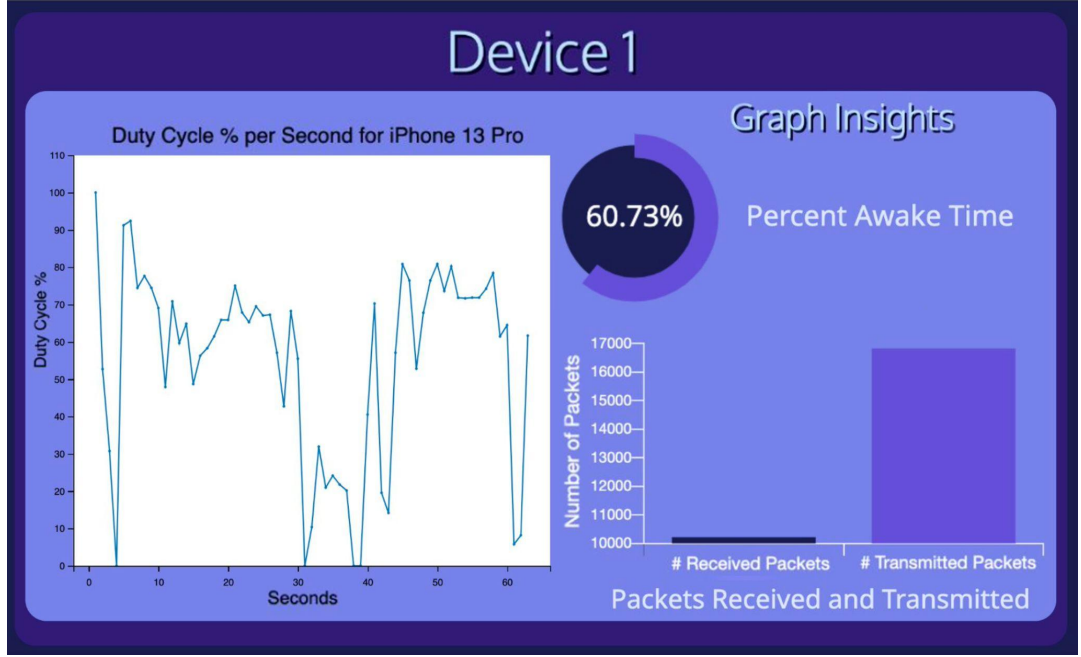


Fig. 4.5: EMT’s dashboard of insights for the iPhone 13 Pro used in the experiment

packets received and transmitted. As depicted in Figure 4.5, the dashboard includes these insights for all of the user’s currently connected devices, and displays these three main insights in an intuitive manner.

### 4.3 Percentage Awake Time

The logs provide us with exact timestamps of when the device enters and leaves sleep mode, and enters and leaves awake mode. So, by collecting this information throughout the course of a device’s connection to the AP, we can deduce the total time that a device spent awake and asleep during the experiment. By taking the ratio of the time it spent awake to the time it spent asleep, we are able to calculate the percentage of time it spent consuming power. This is because when it is in awake mode, it is receptive to incoming data packets and also transmits data to the AP. So, this ratio can be correlated to the device’s energy efficiency. We label

this value the *Percentage Awake Time*, and it is one of the three main insights we provide to the user through the dashboard.

From our test case, we calculated the Percentage Awake Time for the iPhone 13 Pro to be 60.73%. This means that for the duration of our experiment, the iPhone spent 60.73% of its time awake and consuming power, and 39.27% of its time asleep and conserving power. We displayed this statistic to the user, as in Figure 4.7.

## 4.4 Duty Cycle

While the Percentage Awake Time provides a broad overview of how much time a device spends awake and asleep, it's equally necessary to keep track of how much energy it uses and calculate the duty cycle percentage over shorter time intervals. EMT provides the user with a duty cycle per second graph, as seen in Figure 4.6, of their device's transition from asleep to awake modes every second. This interval of time offers a user with a second-by-second breakdown of their device's energy patterns, that is then correlated to the power consumed per second.

The duty cycle per second graph shows a detailed breakdown of how the device switches between asleep and awake modes every second. It helps the user understand how much energy was consumed per second. We calculate it by keeping track of a device's awake time and then dividing it by the sum of the total awake and total sleep times. This graph helps a user identify the random irregularities that could occur while a device is connected to the access point.

The graph in Figure 4.6 is the duty cycle that was recorded from our test case. From seconds 10 to 30, we were playing the YouTube video. The duty cycle shows that the device was awake, and that it was receiving and transmitting much data

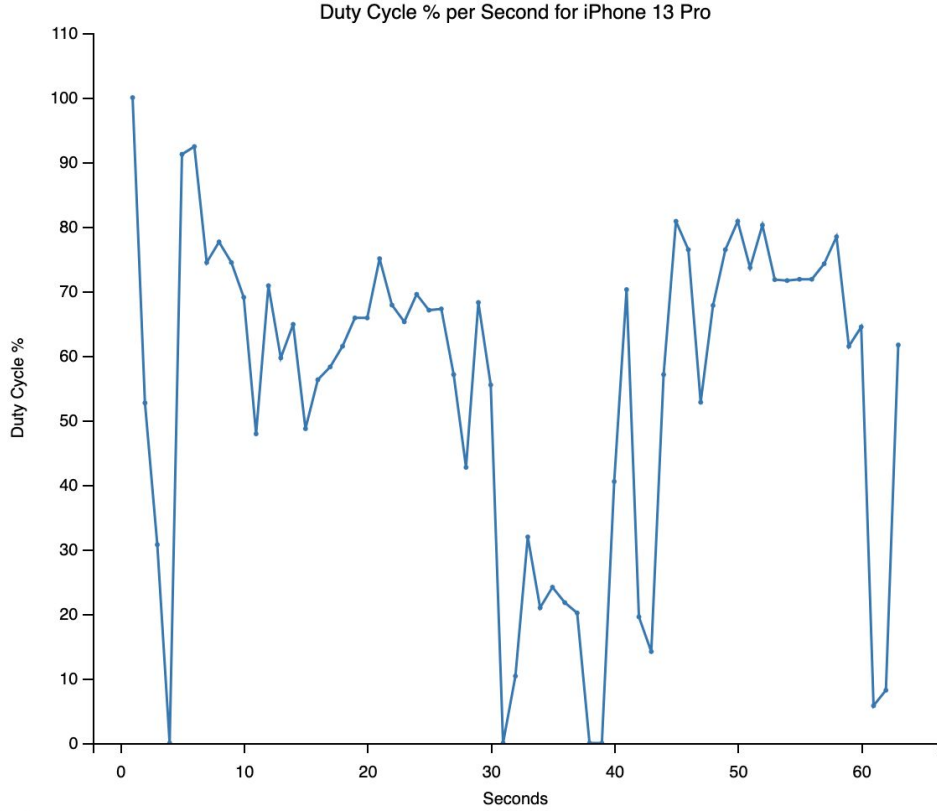


Fig. 4.6: The duty cycle graph for the iPhone 13 used in our experiment.

during this period. From seconds 30 to 45, the device was turned off, but the duty cycle is not completely at 0. For the duty cycle to be at 0, it would mean that the device enters complete sleep mode for 15 seconds and does not receive or transmit any data at all. As in Figure 4.6, the phone was locked, not powered off, but was still receiving notifications and performing functions in the background, which is why the duty cycle per second is lower than it is while the phone is actively being used, but not entirely at 0. At 45 seconds, the phone was turned on again to open the New York Times and perform Internet browsing. This is where the duty cycle shoots up again, as the user is actively using their device for browsing.

If any irregularities were to occur, such as interference from other devices on the same network, it would be reflected in this graph. Additionally, if the device's

duty cycle was not in accordance with its usage, the user can deduce this from the graph. For example, if the duty cycle had been higher during seconds 30 to 45 when the device was locked, rather than when it was being used, the user could sense that some unexpected energy consumption behavior is ongoing in the device's background operations. From here, the user would be able to take further steps to prevent unexpected battery consumption issues.

This graph serves as a tool for the user to continuously monitor the duty cycle of the device for every second that it spends connected to the network. As explained in Section 3.3, sudden or unexpected increases in the duty cycle percentage indicates that the device spends more time in awake mode than normal. This increased awake time corresponds directly to increased power consumption, and if the duty cycle does not quickly return to its regular range, the user can conclude that the device's power would be depleting much faster than normal. So, the user can find the exact timestamp in the graph when the duty cycle percentage became abnormal, and correlate the graph and the timestamp with that of other devices in that network, to deduce when and how the network irregularity began. This arms the user with the data to make informed decisions about their network configurations in order to prevent the unexpected battery depletion of their connected devices.

## **4.5 Number of Packets Transmitted and Received**

We recorded the timestamps at which the device received packets, transmitted packets, missed packets, and polled the AP in order to determine and analyze the amount of data exchanged between the iPhone 13 Pro and the AP. From this data, we calculated the total number of packets transmitted and received by the device during the experiment. The packets transmitted refers to the number of packets

sent to the AP from the device, while the packets received refers to the number of packets sent from the AP to the device. These are referred to as uplink and downlink packets, respectively.

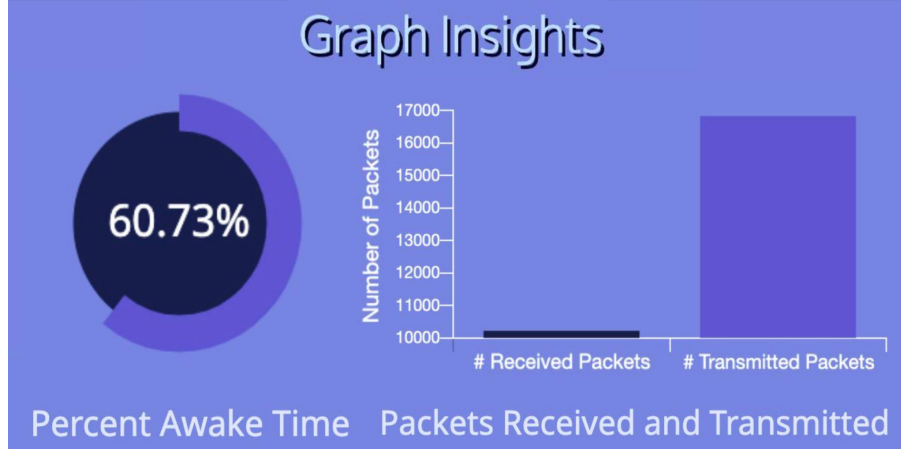


Fig. 4.7: The percentage awake time and packet transmission information for the iPhone 13, our test case device.

From our test case as seen in Figure 4.7, The insights from the iPhone 13 Pro device show that there are 170% more uplink packets than there are downlink packets. These packet transmission insights inform the user about the device's normal or average packet transmission and reception rates. If a device suddenly transmits or receives a significantly larger number of packets than usual, then it could be mapped to greater network traffic or congestion. This can also be correlated to other devices on the same network that could be occupying a larger portion of the channel utilization.

The large variance between the number of downlink and uplink packets reveals that the iPhone 13 Pro transmitted more data than it received over the course of this experiment. One of the causes for the large disparity between the number of downlink and uplink packets could be due to network interference or traffic congestion, requiring the device to retransmit its packets, and thus consume more energy. By monitoring the number of uplink and downlink packets, the user can remain

informed of any irregularities in the network and be aware of how this impacts their devices' energy consumption.

---

## CHAPTER 5

# Conclusion and Future Work

With the world running on wireless communication, the increasing battery-based energy consumption of wireless devices has resulted in alarming environmental consequences. Our goal for EMT is that our tool will enable users to learn more about how energy consumption and constant battery depletion results in the inefficient use of IoT devices. EMT prevents failures and costly downtime, and helps avoid disasters like the St. Jude Medical catastrophe that we discussed in Chapter 1. EMT adapts to the user’s network configuration and informs them of specific ways to manage their energy usage. This helps users implement more sustainable energy consumption practices, and helps them better configure their networks this way.

In this thesis, we recognized that batteries are unreliable sources of energy for WiFi IoT devices, and emphasized the need for an energy management platform. We discussed existing solutions to this problem and how they are not scalable to large networks, and usually require additional hardware components. We proposed a new platform, EMT, to solve this problem. EMT, is an Energy Management Tool for IoT devices. EMT is a novel, low-cost, scalable, and entirely software-based energy monitoring tool that aims to perform battery usage monitoring, by tracking and storing the data transfer information of a user’s devices. Using the device’s data transfer information, EMT performs a comprehensive analysis on these logs to determine the device’s energy efficiency. It displays these insights to the user in the form of an interactive and intuitive web-based interface. This low-cost and viable

solution will help users monitor the energy consumption of their devices and make changes to their network configurations based on EMT's insights.

As a future development, EMT will offer real-time, catered recommendations to the user based on their unique network configuration. We plan to leverage concepts from fog computing technology, and the AP will serve as a fog node to perform analysis on the collected data. We will apply data analysis techniques on the data we gather, and utilize machine learning algorithms to predict a device's energy consumption over time and anticipate when it will run out of energy. So, EMT will be able to predict changes in a device's duty cycle and offer better recommendations to the user. This low-cost and viable solution will help users monitor the energy consumption of their devices and improve network deployment and design.



---

# Bibliography

- [1] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, “Empiot: An energy measurement platform for wireless iot devices,” *Journal of Network and Computer Applications*, vol. 121, pp. 135–148, 2018.
- [2] M. Kuykendal, “Battery-Powered Medical Devices: Their Failure Modes and Mitigation Strategies,” in *Exponent*, 2018. [Online]. Available: <https://www.exponent.com/knowledge/alerts/2018/11/battery-powered-medical-devices>
- [3] C. Kyrkou, C. Laoudias, T. Theocharides, C. G. Panayiotou, and M. Polycarpou, “Adaptive energy-oriented multitask allocation in smart camera networks,” *IEEE Embedded Systems Letters*, vol. 8, no. 2, pp. 37–40, 2016.
- [4] C.-C. Li, V. K. Ramanna, D. Webber, C. Hunter, T. Hack, and B. Dezfouli, “Sensifi: A wireless sensing system for ultrahigh-rate applications,” *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2025–2043, 2022.
- [5] Y. He, R. Yuan, X. Ma, and J. Li, “Analysis of the impact of background traffic on the performance of 802.11 power saving mechanism,” *IEEE Communications Letters*, vol. 13, no. 3, pp. 164–166, 2009.
- [6] J. Sheth, C. Miremadi, A. Dezfouli, and B. Dezfouli, “Eaps: Edge-assisted predictive sleep scheduling for 802.11 iot stations,” *IEEE Systems Journal*, pp. 1–12, 2021.

- [7] J. Sheth and B. Dezfouli, “Enhancing the energy-efficiency and timeliness of iot communication in wifi networks,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9085–9097, 2019.
- [8] M. Kopikare, R. Banerjea, P. A. Lambert, and R. Fanfelle, “Power save mode for access points,” U.S. Patent No. US8917645B2, 23 Dec., 2014.
- [9] D. Camps-Mur, M. D. Gomony, X. Pérez-Costa, and S. Sallent-Ribes, “Leveraging 802.11n frame aggregation to enhance qos and power consumption in wi-fi networks,” *Comput. Netw.*, vol. 56, no. 12, p. 2896–2911, aug 2012. [Online]. Available: <https://doi.org/10.1016/j.comnet.2012.05.004>
- [10] L. M. Feeney and M. Nilsson, “Investigating the energy consumption of a wireless network interface in an ad hoc networking environment,” in *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’01)*, 2001, pp. 1548–1557.
- [11] A. Milenkovic, M. Milenkovic, E. Jovanov, D. Hite, and D. Raskovic, “An environment for runtime power monitoring of wireless sensor network platforms,” in *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory (SSST’05)*, 2005, pp. 406–410.
- [12] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, “Napman: Network-assisted power management for wifi devices,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 91–106.
- [13] J. Sheth and B. Dezfouli, “Enhancing the energy-efficiency and timeliness of iot communication in wifi networks,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9085–9097, 2019.

- [14] P. Gawande and R. A., “Performance analysis of an improved adaptive power saving technique for ieee 802.11 ac systems,” in *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, 2016, pp. 953–957.
- [15] T. H. Lim and S. H. Rhee, “An adaptive power management scheme for wlans using reinforcement learning,” in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 412–415.
- [16] B. Peck and D. Qiao, “A practical psm scheme for varying server delay,” *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 303–314, 2014.
- [17] “HostAPD.” [Online]. Available: <https://wiki.gentoo.org/wiki/Hostapd>
- [18] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. Vasquez Bernal, “Creating complex network services with ebpf: Experience and lessons learned,” *IEEE*.
- [19] J. Sheth, V. Ramanna, and B. Dezfouli, “Flip: A framework for leveraging ebpf to augment wifi access points and investigate network performance,” *Association for Computing Machinery*, 2021.






# Senior\_Design\_Nidusha\_Aastha\_Sreya

Final Audit Report

2022-06-06

Created:	2022-06-06
By:	Darcy Yaley (dyaley@scu.edu)
Status:	Signed
Transaction ID:	CBJCHBCAABAA4mxiamlN47aAq0vezvWL85B-0kqlJOeG

## "Senior\_Design\_Nidusha\_Aastha\_Sreya" History

-  Document created by Darcy Yaley (dyaley@scu.edu)  
2022-06-06 - 4:05:15 PM GMT
-  Document emailed to N. Ling (nling@scu.edu) for signature  
2022-06-06 - 4:05:52 PM GMT
-  Email viewed by N. Ling (nling@scu.edu)  
2022-06-06 - 4:05:57 PM GMT
-  Document e-signed by N. Ling (nling@scu.edu)  
Signature Date: 2022-06-06 - 5:00:09 PM GMT - Time Source: server
-  Agreement completed.  
2022-06-06 - 5:00:09 PM GMT