UNIVERSITY OF CALIFORNIA
RIVERSIDE

Distributed Adaptive Energy Management Solutions for IoT Networks

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Elizabeth H. Liri

September 2023

Dissertation Committee:

Dr. K.K. Ramakrishnan, Chairperson
Dr. Srikanth Krishnamurthy
Dr. Nael Abu-Ghazaleh
Dr. Jiasi Chen
Dr. Koushik Kar

The Dissertation of Elizabeth H. Liri is approved:

_____

_____

_____

_____

_____

Committee Chairperson

University of California, Riverside

Acknowledgments

I wish to especially extend my heartfelt thanks, to my mentor and esteemed supervisor, Professor K. K. Ramakrishnan of the University of California Riverside, Riverside, California for his invaluable guidance and support throughout my academic journey. His commitment to excellence, research, and scholarly pursuits has been truly inspiring, and has helped shape me into the researcher I am today. Additionally, I wish to express my gratitude to Professor Koushik Kar of Rensselaer Polytechnic Institute, Troy, New York for his indispensable contributions to my research.

I deeply appreciate the diligent efforts of my dissertation committee and the continuous support provided to me by my colleagues and faculty at the University of California Riverside. I also extend thanks to HP Labs, Dr. Geoff Lyon, and Dr. Puneet Sharma for providing their expertise and the PERMIT sticks that enabled me to complete my research.

In a special way, I wish to express my heartfelt appreciation to my parents Mr. and Mrs. Henry and Rose Olule, and my siblings for their consistent support, affection, and valuable guidance. Their encouragement, wisdom, and empathy served as the bedrock of my life, and I am deeply thankful for everything they have done.

I express my special appreciation to my in-laws and extended family for their combined support, encouragement, and assistance that have been instrumental in my journey.

To my husband, Francis Xavier Liri, I wish to express my deep appreciation for his unwavering support and invaluable contributions to my academic journey. I feel incredibly fortunate to have such an exceptional life partner who has consistently stood by my side, offering strength and encouragement during every challenge. His steadfast support has

played a key role in helping me reach this significant milestone.

Last but not least, I thank my Lord Jesus Christ, who is forever faithful and has blessed my family and me with uncountable blessings. He has carried us through many challenging times during this academic journey and we know that when we place our trust in Him, He never fails us.

Large sections of the content of this dissertation are a reprint of the material as it appears (in full or in part) in the publications listed below. Prof. K.K. Ramakrishnan listed in the publications directed and supervised the research which forms the basis for this dissertation and additional co-authors provided technical expertise.

This research was supported by grants from the US National Science Foundation (CNS-1619441, CNS-1618344, CNS-1818971), a grant from Futurewei Inc., and US Department of Commerce NIST PSIAP award 70NANB17H188.

1. E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. K. Ramakrishnan, "Robustness of IoT Application Protocols to Network Impairments," in 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2018, pp. 97–103.

2. E. Liri, N. Orie, H. Siezar, K. Ramakrishnan, P. K. Singh, W. Cai, L. Gerday, K. Kar, G. Lyon, and P. Sharma, "Communication Energy and Protocol Considerations in IoT Deployments," in 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). IEEE, 2020, pp. 1–7.

3. E. Liri, K. K. Ramakrishnan, K. Kar, G. Lyon, and P. Sharma, "Invited paper: An Efficient Energy Management Solution for Renewable Energy-based IoT Devices," in Proceedings of the 24th International Conference on Distributed Computing and Networking, ser. ICDCN '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 20–27. [Online]. Available: https://doi.org/10.1145/3571306.3571387

4. E. Liri, K. Ramakrishnan, and K. Koushik, “A Renewable Energy-Aware Distributed Task Scheduler for Multi-sensor IoT Networks,” in Proceedings of the ACM SIGCOMM Workshop on Networked Sensing Systems for a Sustainable Society, 2022, pp. 26–32.

5. E. Liri, K. K. Ramakrishnan and K. Kar, ”Energy-Efficient Distributed Task Scheduling for Multi-Sensor IoT Networks,” in IEEE Network, vol. 37, no. 2, pp. 318-324, March/April 2023, doi: 10.1109/MNET.006.2200548.

To my Lord Jesus Christ for His Grace that has carried us thus far, His Mercies that are new every morning, and His Word that is ever True.

To my husband Xavier, my parents, and my siblings for all their support.

ABSTRACT OF THE DISSERTATION

Distributed Adaptive Energy Management Solutions for IoT Networks

by

Elizabeth H. Liri

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2023
Dr. K.K. Ramakrishnan, Chairperson

Multi-sensor IoT devices are used to monitor different environmental phenomena however these devices depend on batteries and renewable energy sources for power. Therefore, efficient energy management solutions are needed to maximize device lifetime while ensuring appropriate and sufficient data is captured to be processed by the IoT application. To address this problem at the device we use two strategies.

Our first strategy adapts the IoT device load to the available energy ensuring the device lifetime is extended while meeting the application information requirements. Our on-device solution called Predictive EneRgy Management for IoT (PERMIT) is a low-complexity Model Predictive Control (MPC) based approach that optimizes sensing tasks while ensuring battery charging is prioritized. Evaluations using our prototype IoT device and simulations demonstrate the effectiveness of PERMIT.

Our second strategy leverages cooperative sensing to further increase energy efficiency and minimize temporal overlap. Cooperative sensing allows multiple IoT devices to collaborate and coordinate their sensing operations to obtain the required data while min-

imizing energy use. Our Distributed Task Adaptation (DTA) algorithm empowers devices to modify their sensing task operations using information from neighboring devices. DTA works with our distributed Block Scheduler that minimizes overlap across all executions of a sensing task among neighboring devices by reframing the scheduling problem as a block placement problem and minimizing total block overlap every epoch. Experimental results demonstrate the effectiveness of DTA and the Block scheduler in saving energy and reducing temporal overlap among IoT devices while using tokens to share minimal information between IoT devices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-sensor IoT devices enable the monitoring of different phenomena using a single device. Often deployed over large areas, these devices have to depend on batteries and renewable energy sources for power. Devices may also experience varying network conditions such as loss, disruptions and delays which may affect their operations. Therefore, it is important to be able to understand the impact of such scenarios on the device energy. It is also important to be able to develop efficient energy management solutions that maximize device lifetime and information utility and which can adapt to changes in the environment.

Constrained Application Protocol (CoAP) and Message Queue Telemetry Transport (MQTT) are two IoT application layer protocols that are seeing increased attention and industry deployment. CoAP uses a request-response model and runs over UDP, while MQTT follows a publish-subscribe model running over TCP. For more constrained IoT devices, MQTT-Sensor Networks (MQTT-SN) provides a UDP-based transport between the sensor and an MQTT-SN gateway, while using TCP between that gateway and the MQTT

broker. Quick UDP Internet Connections (QUIC) is a new protocol and although it was not originally designed for IoT devices, some protocol features such as reduced connection establishment time may be useful in an IoT environment. Each of these protocols we consider seeks optimizations in features and implementation complexity based on application domains rather than having the full flexibility and adaptability of traditional transport protocols such as TCP.

In our first work [1] we investigate and analyze four protocols, namely, CoAP, MQTT, MQTT-SN and QUIC, to understand the overhead of obtaining data from an IoT device at a sink to potentially disseminate this data downstream. These constrained IoT devices often operate under challenging, varying network conditions, and it is important to understand the limitations of the protocols in such conditions. Thus, we evaluate the performance of these protocols under varying loss, delay, and disruption conditions to identify the most effective environment for their operation and understand the limitations of their dynamic range. Our experimental results show that with non-confirmable CoAP a more adaptive wait timer is required. In addition, a more streamlined QUIC protocol may be a potential alternative IoT protocol.

In our second work [2], we experimentally evaluate the impact of protocol features at the transport and application layers on the energy consumed in IoT devices. Our transport layer experiments with TCP and UDP over WiFi indicate that at distances less than 15m between the IoT device and gateway, the difference in energy used by the different transport protocols is not too significant, as long as the number of messages exchanged does not grow significantly. However, adding application layer features through additional

message exchanges comes with a significant energy cost. For example, adding DTLS to CoAP is 4x more expensive than using AES with CoAP. In addition, if services at one layer impose a burden on other layers, it may result in a significant increase in energy use. We conclude that in an IoT environment, it may be preferable to increase packet sizes rather than number of messages when feasible. Additionally, careful consideration should be given to adding features at a layer if it results in increased energy consumption at another layer.

Based on this research we then propose two strategies for IoT energy management at the device level. The first strategy adapts the IoT device task load based on the available energy and here we discuss our Predictive Energy Management Solution for IoT (PERMIT) which was initially published in [3] as our third work.Our smart energy management solution for IoT applications uses a Model Predictive Control (MPC) approach to optimize IoT energy use and maximize information utility by dynamically determining task values to be used by the IoT device's sensors. Our PERMIT solution uses the current device battery state, predicted available solar energy over the short-term, and task energy and utility models to meet the device energy goals while providing sufficient monitoring data to the IoT applications. Our centralized PERMIT solution runs on a sink and downloads the task parameters to the IoT devices. However, this incurs significant communication energy costs at the device since it has to regularly upload state information.

To avoid the need for executing the optimization at a centralized sink, we propose a distributed PERMIT solution, which uses an efficient MPC Approximation that is simple enough to be run on the IoT device itself. PERMIT decomposes the MPC optimization problem into two levels: an energy allocation problem across the time epochs, and

task-dependent sensor scheduling problem, and finds efficient algorithms for solving both problems. Experimental results show that both the centralized and distributed PERMIT solutions are able to adapt the task values based on the available energy. Results also show that the distributed PERMIT closely approximates the centralized PERMIT solution in sensing performance.

The second strategy leverages cooperative sensing where each device utilizes task information from its neighbors to adapt its own sensing operations and perform intelligent scheduling. For this, we propose a distributed scheduler to intelligently schedule and adapt sensing operations at each device based on information from its neighbors.

For intelligent scheduling, our Distributed Token and Tier-based Task scheduling protocol (DTTS) (proposed in [4] and in [5]), is an energy-efficient distributed scheduler for a network of multi-sensor IoT devices. DTTS, adapting on an epoch-by-epoch basis distributes task executions throughout an epoch to minimize temporal sensing overlap without exceeding task deadlines. The DTTS algorithm divides the monitoring period into a set of non-overlapping tiers, determines task start deadlines, and schedules tasks with shorter deadlines in earlier tiers in a distributed manner and with minimal information shared between IoT devices. When compared against a simple periodic scheduler, DTTS always schedules its task start times before the start deadline. The limitation of DTTS however is that it only considers the overlap of the start time of the first task execution. Thus we extend our algorithm and present our distributed Block scheduler that minimizes overlap across all executions of a sensing task among neighboring devices by reframing the scheduling problem as a block placement problem and minimizing total block overlap every epoch.

Experimental evaluation of our Block Scheduler demonstrate that it always gives results similar to the minimum temporal overlap obtained by an optimal scheduler under a variety of conditions.

Our Distributed Task Adaptation (DTA) algorithm empowers devices to modify their sensing task operations using extra information from neighboring devices. We partition the monitored area into a grid and establish a coverage threshold, representing the minimum number of measurements needed per grid point. Each device utilizes the sensing task information from its neighbors, which is conveyed in a token through two rounds of network traversal, to fine-tune its sensing task parameters, ensuring that the coverage threshold is met for all grid points within its scope. In the first token round, as the token progresses in a forward direction, each IoT device assesses the current coverage status of its grid point based on the information provided by its upstream neighbors within the token. It calculates the difference between the current coverage and the required coverage threshold and adjusts its task parameters to match the average difference. In the second token round, during the token's reverse journey carrying data from downstream neighbors upstream, our DTA algorithm further refines the task parameters to rectify any grid points that are over-covered or under-covered. Experimental results demonstrate the effectiveness of the DTA algorithm in saving energy. In small networks, energy savings of 5% on average per IoT device were observed under both cloudy and sunny weather conditions. Notably, for some devices with overlapping coverage areas, we achieved remarkable energy savings exceeding 30%. In larger IoT networks, our DTA algorithm exhibited average energy savings per IoT device of 3.34% on cloudy days and up to 38.53% on sunny days. These findings underscore its effectiveness

across diverse environmental conditions.

Chapters 2 and 3 discuss our IoT measurement work evaluating the performance of IoT protocols under network impairments and the impact of complexity at the transport and application layers on energy. Chapter 4 discusses our PERMIT IoT energy management solution that can operate on the IoT device and Chapter 5 discusses the DTTS scheduler. Chapter 6 presents our Distributed Task Adaptation Algorithm and Block Scheduler and we conclude this report in Chapter 7.

Chapter 2

Robustness of IoT Application Protocols to Network Impairments

2.1 Introduction

The Internet of Things (IoT) enables sensors and other devices to interact with each other, the environment, and communicate over the Internet [6]. In the past few years, there has been an exponential growth in the use of IoT devices in many fields, including vehicular related applications with smart cars and health with e-monitoring. IoT devices have the ability to sense, actuate, process and communicate over the Internet either directly via a cellular network or via a gateway using other radio technologies for Internet access. When designing protocols for an IoT environment, several factors should be considered including device and traffic characteristics, and the operating environment. We focus on constrained devices which have limited resources such as power, memory and bandwidth.

With the constrained devices, maximizing lifetime of the node is critical, and one way to achieve this is by use of energy efficient protocols for data management and transmission. In order for constrained devices to connect and communicate via the Internet, three core requirements need to be met by the communication stack [7]. It must be low-powered (for energy constrained IoT devices). It must be highly reliable, incorporating error detection, congestion and flow control schemes where necessary, and finally it must be IP enabled, allowing for integration into the Internet.

Application layer protocols for IoT seek to optimize for the environment by simplifying the protocol: having fixed parameters and avoiding the complexity and overhead of general purpose protocols. This may, however, limit the dynamic range of their applicability. General purpose protocols, such as TCP, add complexity by including functionalities, such as in-order delivery, congestion and flow control schemes, so as to be applicable in a wide variety of application scenarios. It is important to understand the tradeoff between application specific protocols vs. the use of a general purpose protocol in terms of performance, robustness and energy use: *What are the limitations of such application layer protocols when the network conditions are variable and impaired, which do not satisfy the assumptions that go into the optimized application layer protocols?*

Our work here is geared towards quantitatively understanding the application layer protocols for constrained IoT devices – How they may operate over networks that have impairments that arise from using low power (e.g. LoWPANs) or having excessive packet loss. Our quantitative comparison looks at a wide set of IoT protocols like CoAP, MQTT, MQTT-SN and QUIC under varying network conditions such as loss, communication disrup-

tion and high delay. We perform careful measurement experiments in both an emulated and physical wireless environment that emulates typical IoT environments, using open source implementations of the various IoT protocols, and Raspberry Pis for end-points.

2.2 Related Work

There are several surveys available which *qualitatively* compare various IoT protocols like CoAP and MQTT [8], [9]. However, there is not much work done on the quantitative comparison of IoT protocols. The authors in [8] present a survey of IoT by focusing on enabling technologies, protocols and applications. They provide a horizontal view of the IoT and look at providing a summary of the most relevant protocols used in IoT together with current application challenges. In this work, they discuss CoAP, MQTT and Advanced Message Queuing Protocol (AMQP) as some of the IoT protocols. Atzori et al. [9] describes the primary communication technologies for both wired and wireless along with the elements of wireless sensor networks (WSNs). The survey presented in [10] specifically discusses the IoT scenarios for specialized clinical wireless devices using 6LoWPAN/IEEE 802.15.4, Bluetooth and NFC for mHealth and eHealth applications. In [11], multiple issues with the development and deployment of IoT devices are discussed, and an IoT architecture is also presented. IoT applications utilizing multiple enabling technologies like RFID are discussed in [12]. Several surveys discuss various discuss challenges specifically for sensor networks, and these include [13].

Among quantitative studies, [14] presents the performance of CoAP and MQTT under a common middleware with different packet loss rates. However, [14] does not de-

scribe the effect of other network conditions like disruption, delay or protocol overhead on performance. In [15], comparison between the performance of CoAP's request-response mode and MQTT's resource-observe mode is presented with packet loss being considered as the only network condition. In [16], both protocols, CoAP and MQTT, are compared in a smartphone application environment. While the paper concludes that CoAP's bandwidth usage and round trip time are smaller than those of MQTT, the performance measurement is done only for a network with 20 % packet loss. To the best of our knowledge, our work is the only quantitative comparison for such a wide set of IoT protocols like CoAP, MQTT, MQTT-SN and QUIC under a wide range of network conditions. The performance comparison of QUIC against other IoT protocols is also a novel aspect of this study.

2.3 IoT Protocols

2.3.1 CoAP

The Constrained Application Protocol (CoAP) created by the IETF [17] is designed to interact with resource constrained devices, such as sensors, using a RESTful API. A key design goal with CoAP was to keep message overhead small and limit packet size. RFC 7252 [17] recommends that CoAP messages fit within a single IP datagram and assumes an IP MTU of 1280 bytes though more conservative limits may be used. Other features of CoAP include use of UDP as the underlying framing and binding protocol, asynchronous message exchanges and stateless HTTP mapping. CoAP seeks to keep parsing complexity low and requires two nodes for communication; a client (request originator) and a server (destination or origin server, usually the sensor or IoT device). CoAP uses four types of

messages for communication between nodes and these are *Confirmable*, *Non-confirmable*, *Reset* and *Acknowledgement* messages.

Both the client and server can utilize Confirmable messages independently although the specification recommends that they be matched. Every Confirmable message sent by an endpoint must be acknowledged by an ACK from the receiver. The ACK can be piggybacked with the data response if the data is immediately available, otherwise the server sends the ACK back first and later transmits the data response as a separate message. For Non-confirmable requests, no ACK is expected by the client. Furthermore, Reset messages are used as a response when the initial received message has an error or is missing context. The total number of packets for servicing a data request ranges from 2 (when both request and response messages are non-confirmable) - 4 (when both the request and response messages are confirmable, with separate ACKs).

Fig. 2.1 and 2.2 illustrate the CoAP call flow in the Confirmable mode under loss and delay. CoAP implements a simple stop and wait ARQ for reliability, including an exponential back-off for Confirmable messages and typically runs over UDP.

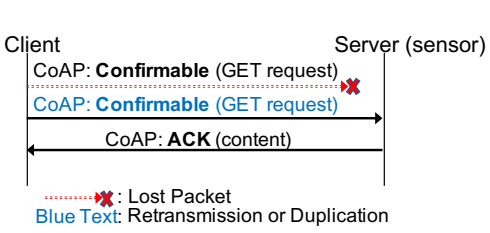

Figure 2.1: Call Flow: CoAP under Loss in Confirmable Mode.

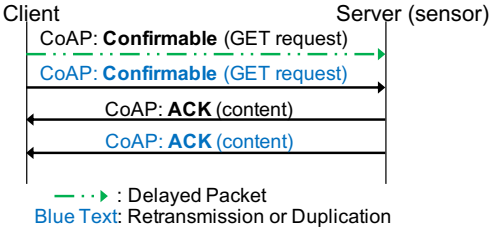

Figure 2.2: Call Flow: CoAP with Delays in Confirmable Mode.

2.3.2 MQTT

MQTT [18] is a publish/subscribe protocol implementing one-to-one, one-to-many and many-to-many connections between IoT devices. The MQTT protocol requires three major components: *subscribers*, *publishers* and *brokers*. Subscribers are devices that subscribe for content being generated by the publishers, which are generally sensor/IoT devices. The subscribed content is identified by topic names and brokers act as intermediary devices, managing the list of subscribed topics and forwarding information from the publishers to the subscribers. Using a broker is a key advantage of MQTT because it manages the subscriptions and also can handle retransmission of data to the subscriber when necessary. Another advantage is the publisher can send new content whenever it is available, thus decoupling the temporal relationship between a node's interest and the publication of the information.

MQTT provides three levels of Quality of Service or reliability: fire-and-forget (QoS 0), deliver at least once (QoS 1) and deliver exactly once (QoS 2). QoS 2 messages are useful in scenarios where neither duplication nor loss is acceptable. The difference between QoS 0 and QoS 1 is that with QoS 1, the PUBLISH message from the sender, which contains the sensor data, requires a PUBLISH ACK back from the recipient. In order to provide the required QoS guarantee, the client and server must store session state for the entire duration of the session or active network connection. For constrained devices, this may be a resource constraint, and failures will cause loss or corruption of the session state. MQTT is built ontop of TCP to get a lossless, ordered stream of bytes between the two nodes (i.e. from sensor to the broker or broker to the subscriber). Figs. 2.3 and 2.4 illustrate one MQTT call flow scenario under loss and delay. When sending data from a

sensor to the MQTT broker, a single data response may require more than 12 packets to be transferred between endpoints as shown in the figures.

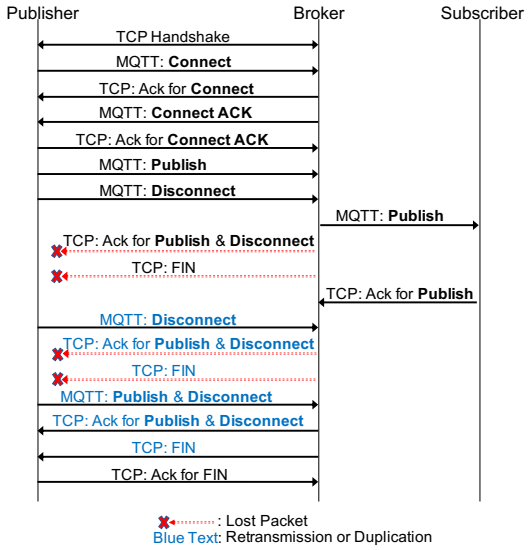

Figure 2.3: Call Flow: MQTT under loss condition with QoS = 1.

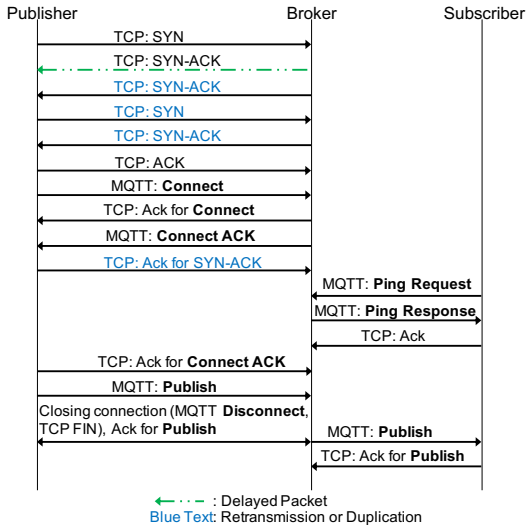

Figure 2.4: Call Flow: MQTT under delay condition with QoS = 1.

2.3.3 MQTT-SN

A variant of MQTT known as MQTT-Sensor Networks (MQTT-SN) [19] built on top of UDP seeks to reduce the overhead of TCP session maintenance associated with MQTT at the publisher. MQTT-SN was designed for constrained devices that communicate over a wireless channel. A new node called the MQTT-SN Gateway, is introduced to connect MQTT-SN clients to the MQTT broker. The MQTT-SN Gateway basically translates messages sent from the publishers using the MQTT-SN protocol into MQTT protocol messages used by the broker, and vice versa.

MQTT-SN differs from MQTT in several ways. These include replacing the topic

name by a shorter topic ID, the use of predefined topic IDs and a gateway advertisement, discovery and publish protocol. The MQTT-SN gateway can either be a separate node or co-located with the MQTT broker. Another node called a forwarder may also be used to allow the MQTT-SN clients to communicate with the MQTT-gateway when the MQTT-SN gateway is not directly attached to the network. MQTT-SN, like CoAP, does not support message fragmentation and reassembly. The advantage of MQTT-SN from an IoT device perspective is that since it can be implemented over an unreliable transport, in terms of the number of messages sent out from the publisher, it has less overhead and is more efficient, running over an unreliable transport. Fig. 2.5 illustrates one MQTT-SN call flow scenario under loss conditions. In addition to the MQTT packets between the MQTT-SN gateway and the MQTT broker, MQTT-SN requires at least an additional 6 packet transfers between the publisher and the MQTT-SN gateway to service a single data response, thus requiring approximately 18 packets overall.

2.3.4 QUIC

QUIC, a transport protocol developed by Google, seeks to address several transport and application layer challenges that modern applications experience [20].The protocol runs over UDP, emulating the functionality of the set of TCP+TLS+HTTP protocols. Some advantages of QUIC include reduced time to establish a connection since it uses one round trip to complete a handshake compared with TCP+TLS which use 1-3 round trips and multiplexing with no head of line blocking compared to TCP+HTTP.

QUIC uses TCP-CUBIC's mechanisms for congestion control and includes addi-

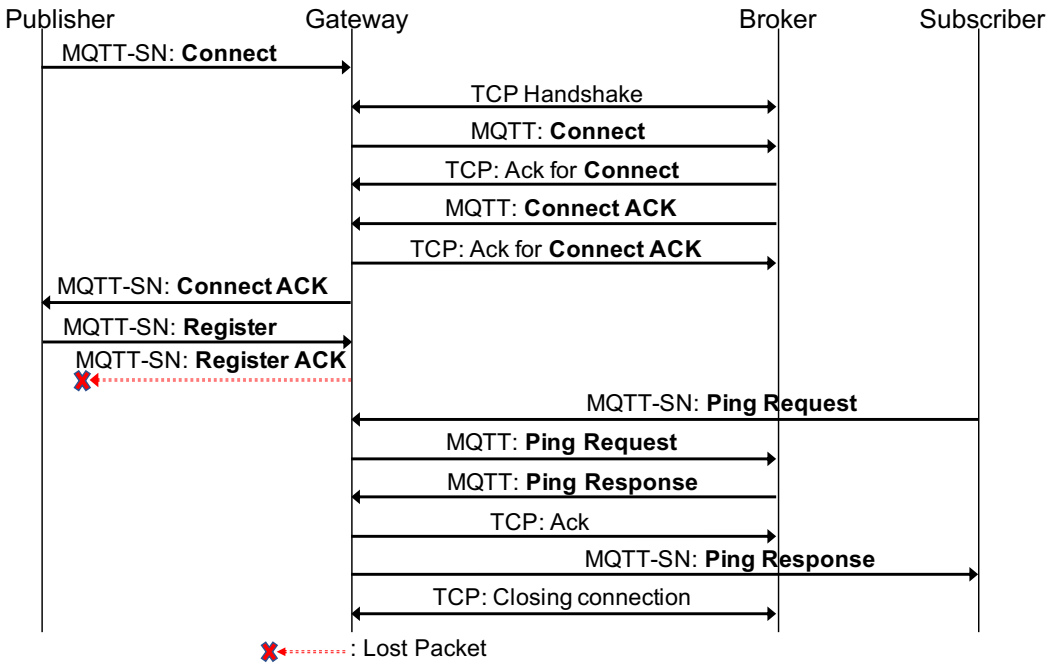

Figure 2.5: MQTT-SN Call Flow under Packet Loss.

tional information in the congestion control algorithm, in particular including sequence numbers for both the original and retransmitted packets. This allows the QUIC sender to identify retransmitted ACKs and avoid the retransmission ambiguity problem. QUIC ACKs also carry the delay between packet receipt and acknowledgement to more accurately estimate round trip time. It supports up to 256 NACK (Negative ACK) ranges to make QUIC more resilient to reordering. Another key advantage with QUIC is that it avoids head-of-line blocking which occurs when one lost packet in a stream prevents delivery of subsequent packets until the lost packet is delivered. In HTTP2, head of line blocking affects all streams using that TCP connection but with QUIC, packets lost for an individual stream only impact that stream. QUIC requires approximately 17 packets exchanged between the client and server to send data to a sink and most of the packet content is encrypted. Most

of these extra packets are for increased reliability, congestion and flow control. Since the QUIC protocol was initially designed for web transactions, these packets provide improved performance in that environment. However reducing the number of packets may result in a lighter weight version of QUIC, with improved performance, that is suitable for an IoT environment while providing adequate reliability and congestion control.

We postulated that although QUIC has not been used in the context of IoT, some of its features may be useful for IoT communication. A streamlined QUIC protocol may be effective in an IoT environment. For example, the Connection ID feature could be used to enable reliable communication with fewer RTTs than TCP. The use of multiple streams to an individual IoT device could address the head of line blocking with TCP.

2.4 Evaluation

2.4.1 Experiment Setup

The performance of CoAP, MQTT, MQTT-SN and QUIC was evaluated using the metrics of task completion time (which refers to the total duration of the experiment), and total packets transferred (by nearest node to the IoT device) during the experiment, to get an indication of the overhead with each protocol. A simple scenario of a client obtaining information from a sensor, is considered to allow us analyze the behavior at the level of an individual IoT device under different conditions (Fig. 2.6). The behaviour of a network of IoT devices can then be inferred from the results. For evaluation, the experiments were first performed in an emulated environment using VirtualBox VMs. A subset of the experiments were also run using Raspberry Pis [21] over a WiFi network to verify the same behavior in

Table 2.1: Experiment parameters.

Experiment	Value	Description
Base/Ideal	-	No loss, delay or disruptions
Packet Loss Range (%)	0-5, 0-10, 0-20	New value every 100 requests
Network Delay (s)	0-1, 0-2, 0-5	New value every 100 requests
Disruption duration (s)	20, 60	5 disruptions per experiment

an actual wireless environment. Typical IoT environments utilize wireless communication and incur loss, delay and disruptions due to external factors. Therefore the use of WiFi while also varying the loss, delay and disruption parameters allows us to emulate a typical IoT environment. Delay and loss were emulated using traffic control TC [22].

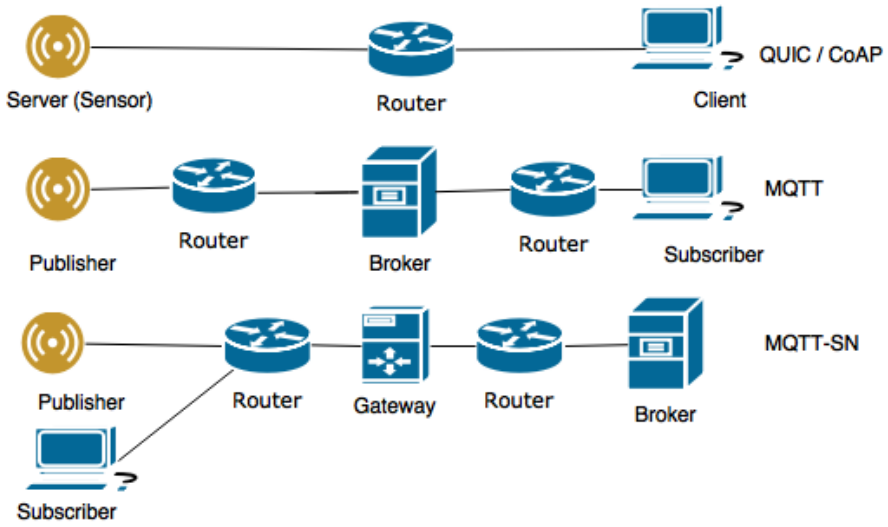

Figure 2.6: Application Scenarios

The CoAP server and client are based on a C implementation, namely Libcoap [23]. The MQTT Publisher/Subscriber/Broker are based on the Mosquitto Broker v3.1 software

[24]. MQTT-SN implementations leveraged the Eclipse's Paho Library [25] and we used a C implementation of QUIC developed by Google called proto-quic [26].

For each experiment, in the request-response protocols (CoAP and QUIC), the client sent 1,000 data requests and the server (IoT node) responded with 1,000 data responses. Each new data request was sent sequentially after receiving the response to the previous request. In the publish-subscribe protocols (MQTT and MQTT-SN), the publisher sent 1,000 data responses to the broker for a pre-subscribed topic.

A data request/response may contain multiple packet transfers between endpoints and the protocol's performance was studied under ideal conditions and then under varying network loss, delay and disruption conditions.

The parameters used are shown in Table 2.1. CoAP experiments were run for Confirmable and Non-confirmable modes, while MQTT and MQTT-SN used the QoS 0 and QoS 1 variants. In the case of MQTT/MQTT-SN, a new TCP connection was established for each data response and experiments were run multiple times for each scenario.

One aspect under investigation was the behaviour of the protocols within their timer limits and when these timers expired. Retransmission timers were used to evaluate this behavior. In order to evaluate all the protocols under identical conditions, we used the CoAP default timer values as a guide because it has the largest timer values that are also static. The two timers used were the ACK_TIMEOUT and the total retransmission time. In the delay case, the default ACK_TIMEOUT for the first CoAP Confirmable message retransmission is between 2s and 3s. Therefore we selected 1s, 2s and 5s as the upper bounds on delay. For a CoAP Confirmable message, the default time from first transmission

to last retransmission is 45s (i.e. MAX_TRANSMIT_SPAN) therefore we selected a short disruption (20s) within the MAX_TRANSMIT_SPAN and a long disruption 60s that exceeds the MAX_TRANSMIT_SPAN.

For the ideal case, there is no network loss or delay. For the experiments with network loss and delay, the instantaneous loss and delay values were changed every 100 requests and uniformly distributed in the range specified in Table 2.1. For network disruption experiment, five random network disconnections occurred for the given disruption duration in 2.1. Wireshark traces were analyzed to evaluate protocol performance.

2.4.2 Performance Analysis

CoAP

Task completion time for CoAP under delay and disruptions is shown in Figure 2.7 and loss in Figure 2.8. As the packet loss increases, task completion time increases. One significant observation (in both environments) is that the task completion time for the Non-confirmable CoAP is significantly longer than Confirmable CoAP. However the Raspberry Pi experiment results (RP) are lower than in the emulation experiment (SM). This is because the SM case for the loss experiment includes a higher script processing delay when changing the loss values every 100 requests. The cause of the high duration with non-confirmable CoAP is explained next.

When a Confirmable client request does not receive a response, it uses a back-off

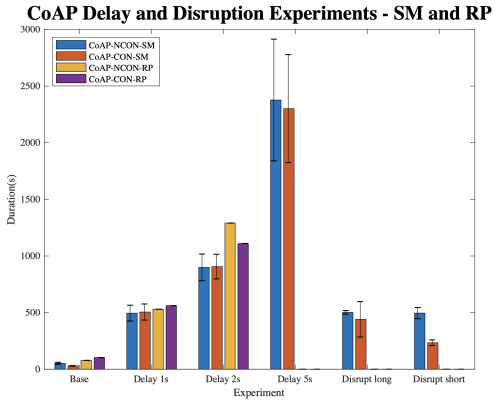

Figure 2.7: CoAP: Task Completion Time with Delay and Disruption

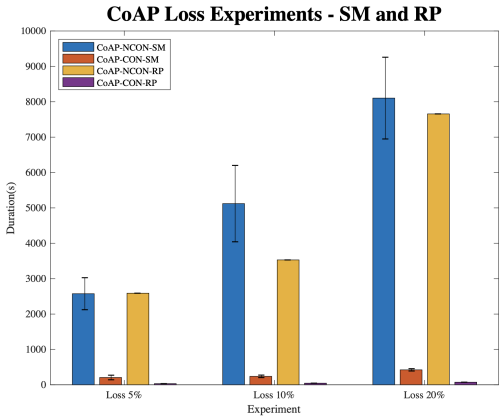

Figure 2.8: CoAP: Task Completion Time with Loss

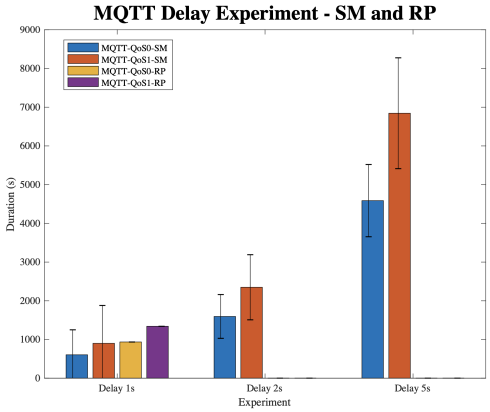

Figure 2.9: MQTT: Task Completion Time with Delay

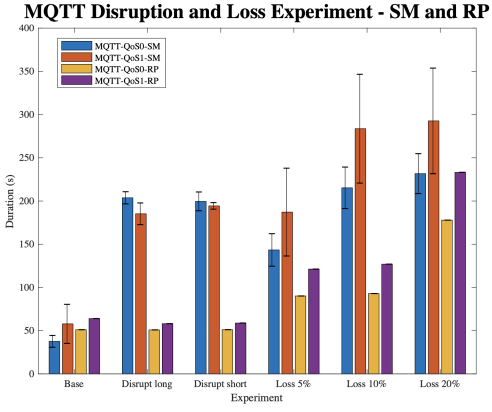

Figure 2.10: MQTT: Task Completion Time with Disruption and Loss

timer to generate subsequent retransmissions. From the loss experiments, generally the second or third retransmission was successful leading to the reduced task completion time when compared with non-confirmable case. However considering the worst case scenario when the maximum number of retransmissions (i.e. 4 by default) has been reached the client gives up on that data request and sends a new request for the next piece of information.

If a client does not receive an immediate response to a non-confirmable request, it does not retransmit the request; instead, it waits for 90s before sending the next request.

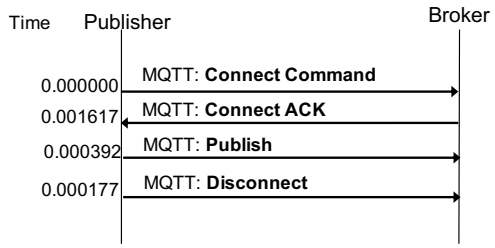

Figure 2.11: MQTT: Time between MQTT Control Commands QoS 0

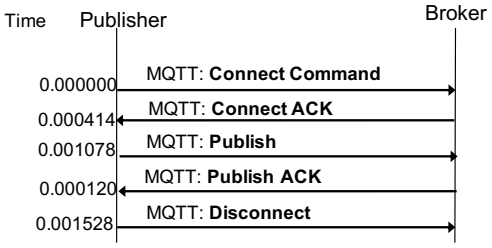

Figure 2.12: MQTT: Time between MQTT Control Commands QoS 1

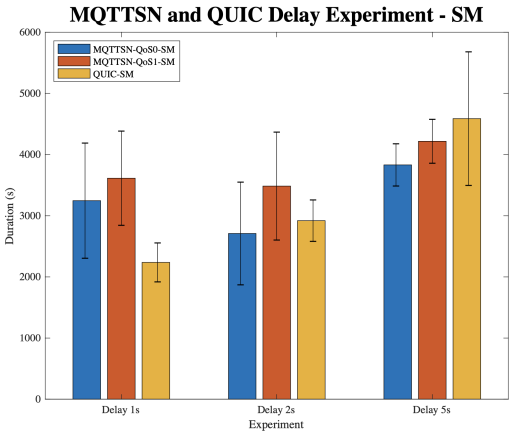

Figure 2.13: MQTT-SN and QUIC: Task Completion Time with Delay

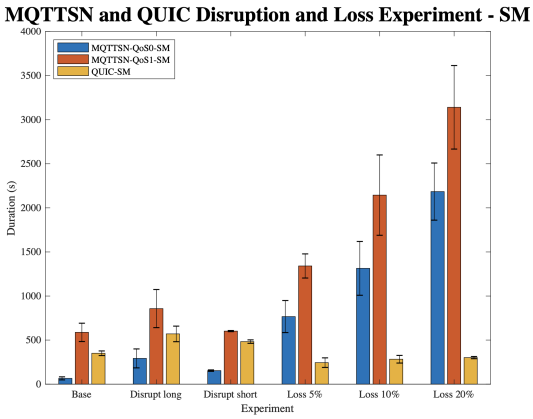

Figure 2.14: MQTT-SN: Task Completion Time with Loss and Disruption

From our experiments, confirmable requests had a maximum wait time of approximately 30s if a response was lost (due to retransmissions), versus the 90s wait time in the Non-confirmable case. This may be categorized as an unexpected, and potentially undesirable, effect since the Non-confirmable option would likely be chosen without the expectation of introducing such a significant performance penalty while reducing overhead. Further analysis indicated this was due to two factors: the NSTART parameter and the timer determining when to send the next request.

The NSTART parameter is the maximum number of simultaneous outstanding interactions at any time between a specific client and server, and has a default value of

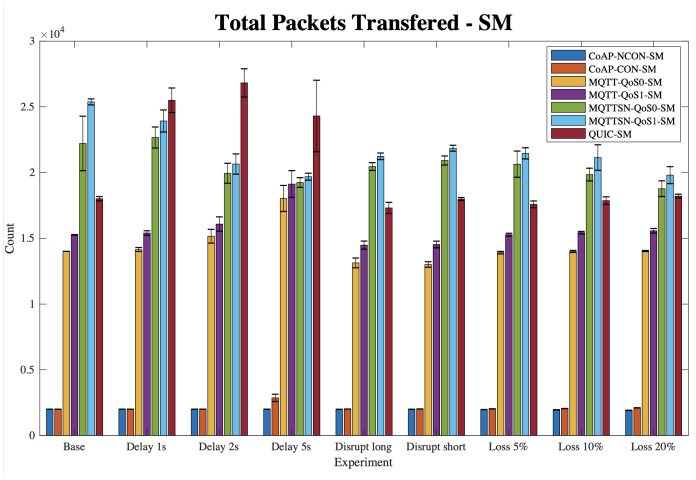

Figure 2.15: Total Packets Comparison

1 (for both Confirmable and Non-confirmable). An outstanding interaction is either an outstanding confirmable request which is still awaiting an ACK or a request still waiting for a response or ACK [27].

For the timer to send the next request for Non-confirmable messages, the RFC does not explicitly indicate which timer should be used to determine how long to wait or what algorithm to use to decide when to stop expecting a response. In the case of Confirmable requests, however, the MAX_TRANSMIT_WAIT timer determines the maximum time a sender should wait from the time of the first transmission of a Confirmable message to when it should give up waiting for either an Acknowledgement message or Reset message. Results from using libcoap indicate that the timer used had a value of 90s, corresponding to the MAX_TRANSMIT_WAIT timer. This points to a weakness of depending on non-adaptive application-layer timers, which have the potential of unintentionally introducing performance limitations.

In the case of delay experiments, the base time is the task completion time in the ideal case and requests are sent sequentially after receiving a response (due to NSTART=1). For the delay experiments, the new task completion time is approximately equal to the sum of base time + (delay * no of packets) + script processing time. The experimental results closely align with the expected result, with the network delay (delay * no of packets) contributing to over 90 % of the total task completion time. The remaining 10% is due to base time and script processing delay. The base time includes internal processing at the receiving node and propagation delay in the ideal case. However the script processing delay is the time needed to switch to new delay parameters using TC (Traffic Control). Compared to the emulation experiment, the switching delay is larger with the Raspberry Pi delay experiments. This caused the difference in duration and depends on the current delay value i.e. changing the delay value in the 1s delay experiment takes less time than changing values in the 5s delay experiment.

For the disruption experiment, with Confirmable requests, in the majority of the cases with the short disruption (20s), the network had recovered by the time the last retransmission was due. For long disruptions (60s), the maximum number of retries was exceeded while the network was still unavailable, so the client dropped that request and proceeded to the next request. Since this new request is sent while the network is still unavailable, it may also require retransmissions.

In the disruption experiment the Non-confirmable clients exhibit the same behavior as described for packet loss in the network. Specifically, when the client receives no response, it waits for the 90s timer to expire before sending the next request. Addressing

this challenge, requires an adaptive timer, or a well-defined 'cookbook' of how applications should set the timer for the Non-confirmable case.

MQTT

With MQTT, increase in loss and delay causes a corresponding increase in the completion time of the experiment due to retransmissions (Fig. 2.9 and Fig. 2.10). The duration is significantly longer with the delay experiments because a single MQTT transaction requires multiple messages to be transferred, and each of these messages is affected by the delay, leading to a cascading effect on the duration. Similar to the CoAP delay experiment, the delay script processing time was significantly longer in the Raspberry Pi (RP) delay experiment than the emulation experiment (SM). On the other hand, in the loss experiment, the script processing delay was more pronounced in the emulation experiment.

In MQTT and MQTT-SN, the difference between QoS 0 and QoS 1 requests is that QoS 1 requires an acknowledgement for any PUBLISH messages that are sent. With MQTT, changing from QoS 0 to QoS 1, leads to an average increase of 48% and 30% in the duration for the delay and loss experiments respectively. This is because in QoS 0, the PUBLISH and DISCONNECT control messages are sent back-to-back from the publisher, while for QoS 1, the publisher needs to wait for the PUBLISH ACK and complete internal processing after receiving it before it can send the DISCONNECT. Fig. 2.11 illustrates the call flow of just these control messages for QoS 0, while Fig. 2.12 is for QoS 1.

In the case of QoS 0, the DISCONNECT is sent after approximately 0.177 ms after the PUBLISH. However, in the case of QoS 1, the DISCONNECT is sent after 1.648ms (1.528+0.12ms), which is an approximately 10x increase when compared with QoS 0.

In terms of total packets transmitted during the experiments, there is only a small increase of less than 10 % when changing from QoS 0 to QoS 1 in the loss and delay experiments (see Fig. 2.15 which shows results of the emulation experiment). Thus, the cost of increased reliability by using QoS 1 is because of the additional round trip significantly increasing time to completion although both QoS 0 and 1 have about the same number of packets. This implies that time sensitive applications can benefit with QoS 0 (without reliability). For the delay experiments, since multiple messages are exchanged between the two nodes, the impact of delay is significantly higher, see. Fig. 2.9.

MQTT-SN

For MQTT-SN, increase in loss or delay results in a decrease in the total number of packets transferred. This is because, if packet loss occurs between the publisher and gateway (e.g., the CONNECT ACK from gateway to the publisher is lost), the rest of the transaction packets such as the PUBLISH and DISCONNECT messages will not be sent.

Task completion time increases with delay and loss and this is partly caused by timers (Keep Alive value in CONNECT message) which control how long to wait before sending the the next message. Comparing the change in task completion time for varying delays, there is an increase of 10-30% in the task completion time for QoS 1 compared to QoS 0. For loss and disruption, the increase in task completion time with QoS 1 is significantly higher, at over 40 %. The reason for this is similar to the MQTT case. With QoS 1, the publisher needs to wait for a Publish ACK before it can send the final disconnect to the gateway, causing a substantial increase in the time between those commands, increasing from about 100 microseconds to hundreds of milliseconds.

QUIC

In the case of delay, the task completion time with QUIC is comparable to MQTT-SN because a single QUIC transaction requires multiple messages to be exchanged between two end hosts. However in the case of delay, the total number of QUIC packets exchanged is significantly higher than when using any of the other protocols (see Fig. 2.15). In the disruption and loss cases, the total number of packets sent by QUIC is fewer than MQTT-SN with a smaller task completion time. Therefore in a lossy or disconnected environment, QUIC provides similar performance to MQTT-SN in terms of task completion time and packets transferred. QUIC, originally designed for resource rich nodes has significantly more packets transferred to get the same number of values from the sensor. However, reducing the packet transfers, by reducing congestion control and reliability information for example, may result in a lightweight version of QUIC, which may provide sufficient QoS guarantees and improved performance to be suitable in an IoT environment.

2.5 Conclusions

In this paper, we extensively analyzed the performance of four application layer protocols for constrained IoT devices over networks with impairments. We performed multiple experiments and evaluated the performance of CoAP, MQTT, MQTT-SN and QUIC in networks with delay, loss and disruptions. Results show that in terms of overhead, CoAP is the most efficient protocol because it only requires two messages for data transfer. Results also show that the cost of using additional reliability with MQTT QoS 1 has more of an impact on task completion time (over 30% difference) than overhead (10% difference).

For QUIC, in its current version, the performance of QUIC in lossy and disruptive environments is comparable to MQTT-SN performance. Therefore, a streamlined QUIC protocol may significantly improve performance and enable QUIC to be a potential request-response IoT protocol alternative to CoAP.

Another key finding from the experiments is that for IoT protocols that use a fire-and-forget paradigm, such as CoAP non-confirmable, MQTT QoS 0 and MQTT-SN QoS 0, the wait timers (keep alive for MQTT/MQTT-SN) play a crucial role in performance. These timers determine how long the node should wait before giving up on expecting a response and/or determine when the protocol can proceed to the next request. In the case of CoAP non-confirmable, we recommend an adaptive timer or a well-defined 'cookbook' of how applications should set the timer. Future work includes implementation of an adaptive timer for the non-confirmable CoAP as well as evaluating the performance of these protocols in networks of multiple heterogeneous and homogeneous IoT devices.

Chapter 3

Communication Energy and Protocol Considerations in IoT Deployments

3.1 Introduction

Edge devices deployed in an IoT environment are typically constrained by limited power, memory and compute capability. These devices are often powered by batteries, and deployed in environments where replacing batteries may expensive or infeasible. Since energy used up in communication makes up the bulk of the total energy cost, designing strategies for reducing the communication energy cost is essential for improving device lifetime. In the context of sensor networks, where energy usage is also a key consideration, there is a large body of prior work on optimizing communication energy cost [28]. Most of

the prior work in this context focus on optimizing individual layers of the communication stack – using rate and power adaptation at the MAC layer or adjusting sensing rates [29] or duty cycles [30] at the application layer, for example. Cross-layer approaches which are more complex and combine parameters from multiple layers have also been proposed [31]. However, similar issues have not been investigated comprehensively for IoT devices, and in the context of the emerging IoT communication protocols.

In this work, we experimentally investigate the impact of IoT protocol features at the transport and application layers on communication energy cost in IoT devices. IoT protocol features refer to the number and types of services provided; examples of such services include reliability and congestion control, encryption and authentication etc. At the transport layer, we re-look at TCP and UDP - which most application layer IoT protocols tend to build upon - in terms of their energy usage. Our consideration of TCP and UDP is also guided by the fact that these two protocols represent the two main alternatives in terms of transport protocol features provided to the application layer - while TCP provides end-to-end reliability, flow and congestion control and a host of other features, UDP does not provide any of these services. Our results show the strong dependence of the transport layer energy consumption on the distance between the IoT edge device and the IoT gateway[1]. Further, we make the two following interesting observations: (i) below a certain distance between the IoT device and gateway (<15m), there is not much difference between using simple and complex transports (e.g., UDP vs. TCP respectively); (ii) above this critical distance, an exponential increase in energy is observed though at a lower rate for the simpler transport protocol only if we consider data sent not data delivered in UDP case. In the

[1]The IoT gateway is the bridge between the IoT device and the Internet or external networks.

application layer, we consider the Constrained Application Protocol (CoAP) [17] in the confirmable mode, and show that adding encryption and authentication features at the application layer can have a significant effect on device communication energy especially if additional messages are involved. In particular, as compared to vanilla CoAP with no encryption or authentication, using CoAP+DTLS increases power consumption by 23.7% - or approximately 4x the additional energy incurred due to simple AES encryption.

The core contribution of our work is in the rigorous *experimental* evaluation of energy consumption in IoT devices (measured at Raspberry Pi devices) over WiFi, for different transport layer protocols, and security options at the application layer protocol. The results may help practitioners to determine which transport protocol to use in IoT deployments, depending on the distances between the IoT devices and the gateways. Further, it would help in the understanding of the additional energy costs associated with adding security features at the application layer. The results also indicate that practitioners and protocol designers interested in energy efficiency should consider protocol features (including cross layer interactions and impact on number of transmitted messages) when making IoT protocol implementation and deployment decisions.

3.2 Background and Related Work

3.2.1 Background

Three of the more popular IoT protocols used today include Constrained Application Protocol (CoAP) [17], Message Queue Telemetry Transport (MQTT) [18], and MQTT for Sensor Networks (MQTT-SN) [19]. CoAP, is a light-weight request/response protocol

designed for resource-constrained devices. Initially designed for a UDP transport, message overhead is kept small by specifying that the payload fit into one packet. CoAP uses a RESTFul API and is standardized in RFC 7252 [17] by the IETF. It uses four types of messages for interactions: *Confirmable*, *Non-confirmable*, *Reset* and *Acknowledgement* messages. CoAP implements reliability using Confirmable messages which use a simple stop-and-wait ARQ mechanism, and an exponential back-off timer for retransmissions. CoAP has also been integrated with TCP [32] to enable CoAP traffic in networks that do not forward UDP traffic. In this case, only TCP manages reliability and message duplication detection as well. For security, CoAP over TCP uses the Transport Layer Security protocol (TLS) (RFC 5246) [33] to provide privacy and data integrity between two communicating applications. CoAP over UDP uses the Datagram Transport Layer Security (DTLS) protocol [34] which is based on TLS and supports authentication, data integrity, confidentiality and automatic key management[35].

MQTT relies on a TCP transport and follows a publish-subscribe model to implement one-to-one, one-to-many and many-to-many connections between IoT devices. It uses a broker to interface between publishers who publish sensor data and subscribers who are interested in receiving this data. MQTT has three reliability levels (QoS levels) at the application layer, fire-and-forget, deliver at least once and deliver exactly once. MQTT-SN is a variant of MQTT that reduces the TCP overhead by using UDP. Comparison of how these protocols operate under network impairments can be found in [1].

3.2.2 Related Work

Mechanisms used to manage energy efficiency include using different optimization strategies at different layers of the communication stack. For example MAC layer strategies include using collision avoidance or output power control, network layer strategies include using efficient routing and data dissemination protocols [36] while application layer strategies include alternate operating schedules of the redundant nodes [37] and adjusting sensing rates [29]. However cross-layer strategies which are more complex and combine parameters from multiple layers have also been proposed [31] and[28] is a survey on some approaches.

Node placement is another strategy to optimize energy efficiency, and [28] discusses different approaches to determine the best node placement to maximize energy efficiency.

Prediction approaches seek to proactively manage device energy, typically by identifying power used by different components of the IoT device e.g., by experimentation, modelling it and using the model to predict future energy use. For example, in [38] the authors model the power used by the IoT communications, acquisition and processing systems. In [39] the authors present a power model (Power Pi) to derive possible power saving strategies for the Raspberry Pi (RP) when used as home gateways. [40] shows that a large fraction of energy is used by frames when they cross the protocol stack (OS, driver, NIC) and [41] extends this to show that this cross factor energy is device independent.

3.3 Evaluation

Protocol features loosely refers to the number and type of services at a protocol layer and these features can be added or removed at different layers. Communication

typically uses the most energy at an IoT device and therefore we consider this and features at the transport and application layers. Our focus is on services offered at these layers and we use the application layer IoT protocol, CoAP.

3.3.1 Transport Layer

At the transport layer, we use TCP and UDP to illustrate the impact of features with respect to reliability. Since TCP and UDP are the most widely used protocols, our insights will be applicable to the many IoT protocols that use them. These two protocols represent two extremes of reliability and have been studied extensively. For IoT protocols that implement their own reliability separately, e.g., CoAP over UDP, their implementation typically lies between the two extremes of full (TCP) and no (UDP) reliability. Therefore evaluating transport complexity using UDP and TCP may give a fair indication of the range of performance possible. Finally, reliable delivery is usually enforced through retransmissions. Therefore, retransmission rates and the mechanisms used for it have a large impact on communication energy. Thus, we use the different reliability mechanisms to compare how protocol features affect energy. We categorize the reliability as no reliability with UDP and full reliability with TCP.

3.3.2 Application Layer

We focus on the application layer because some IoT protocols implement additional services at the application e.g., reliability or security that affect the number/size of messages to be transmitted, and therefore, the communication energy. In order to demonstrate the impact of protocol features on the device energy we use CoAP as an example IoT protocol.

Table 3.1: CoAP Messages

CoAP Setup	Num Round-trips	Total Messages
Basic CoAP	1	2
CoAP with AES Encryption	1	2
CoAP+DTLS	4	8
CoAP+TCP/TLS	5	10

An alternative to CoAP is MQTT. However, since MQTT relies on TCP as the transport, our transport layer results with TCP provide guidance on the impact on MQTT as well. Our consideration of CoAP is also guided by the fact that it has been standardized by the IETF [17], and is finding increasing adoption.

There is also available software and a significant level of ongoing development for CoAP. There is also ongoing research on CoAP, either using it in experiments or proposing solutions to address current limitations of the protocol. Lastly, although CoAP was originally designed to work with UDP as the transport, there is ongoing effort to standardize CoAP over TCP [32] which allows us to compare the penalty incurred by an IoT device when it uses a more complex transport. Before discussing addition of security features we consider how many messages are typically needed for the basic CoAP (with UDP transport) and no security. If we use Confirmable CoAP messages, and include the ACK in response messages, then the average number of messages for a single request and response is 2 (the best case). AES encryption with CoAP, does not increase the number of messages sent, but may increase the message size slightly due to padding to align the data to the size of a block. On the other hand, adding DTLS to a CoAP message generally requires an additional 6 messages (3 additional round trips) for providing encryption, authentication and message integrity. Adding TCP/TLS to a CoAP message generally requires an additional

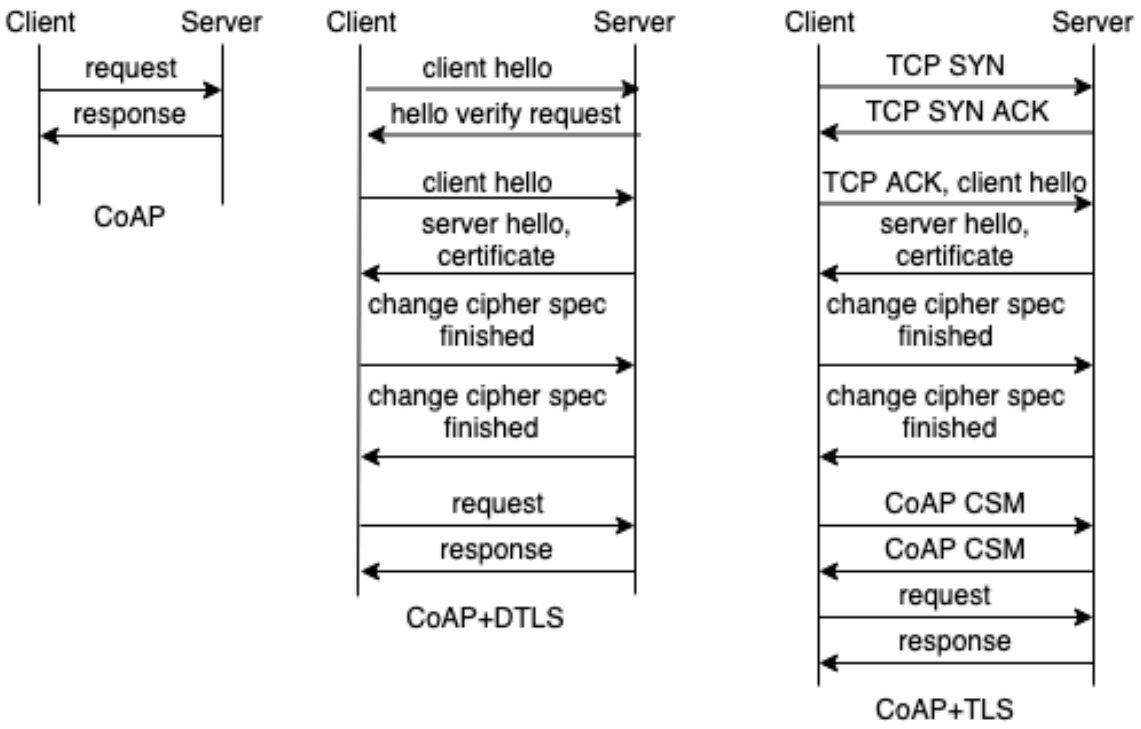

Figure 3.1: Call flows comparing CoAP, CoAP with DTLS and CoAP with TLS

3 round-trips for the TCP and TLS Handshakes and an additional CoAP round trip for the Capabilities and Settings message (CSM) exchange. All of this is prior to sending the CoAP request, thus incurring 4 additional round trips. Table 3.1 compares the number of messages in each case and Fig. 3.1 illustrates the differences. Adding features with increasing complexity which increases both the amount of data to be sent and the messages exchanged, has an impact on energy consumption. A balance needs to be found between the additional desirable features that require complexity and the energy cost of those features. This needs to be done from the perspective of both implementation and operation. For example in terms of implementation, AES provides only encryption and increases the packet size while DTLS requires additional messages but provides authentication as well. In

terms of operation, with CoAP+TCP/TLS, every new TCP connection will incur the extra messages required for setting up the encryption. Therefore if using TCP/TLS with CoAP, practitioners need to decide between utilizing keep alive messages (CoAP Ping and Pong messages) to maintain the TCP connection (thus keeping the IoT device active throughout, and not going to sleep) for a longer time versus having rather short TCP connections and incurring the setup cost of a new connection and TLS session. We perform experiments to understand the quantitative impact of additional features at both the transport and application layers.

3.3.3 Experimental Setup

Transport layer experiments compared energy required for UDP and TCP while application layer experiments compared CoAP with and without security mechanisms. The two types of experiments were performed with different devices, in slightly different environments and restricted to 1 client and 1 sink to compare energy use patterns and trends across devices. 2.4GHz WiFi was used and a power meter (AVHZY CT-2 USB Power Meter Load Tester Voltage Detector) connected to the client measured power consumption.

We recognize that there are a range of communication technologies available for IoT solutions including short range (e.g., RFID and Bluetooth), mid-range (e.g., WiFi and Zigbee), and long range (e.g., NB-IoT, LoRaWAN, SigFox etc). We used WiFi primarily for convenience. WiFi itself is also evolving, with alternatives, such as those being proposed by the WiFi Halow project to design a more energy efficient WiFi protocol for IoT devices based on the original WiFi protocol being a candidate for a range of IoT solutions. We believe that the experimental methodology used here for determining energy consumption

for the networking subsystem could be applied to other communication technologies as well.

IoT applications are evolving, with some requiring IoT devices having increased functionality, more compute and communication capability. Agriculture applications that were traditionally measuring humidity and temperature seek to now use image and video data to monitor plant growth. Smart home monitoring and surveillance also incorporate video or images in addition to motion sensors. The use of low-cost, relatively lower power consuming Raspberry PI devices with adequate compute capability are therefore likely to the type of IoT devices used in these applications. Nonetheless, the one common characteristic is that they are generally energy constrained, with many running on battery power. We consider such energy constrained IoT devices, that can see increased lifetime and functionality through energy management techniques.

Transport Layer Experiments

Figure 3.2: Transport layer experimental setup

Fig. 3.2 illustrates the experimental setup. Iperf3 was used to generate upload traffic from a client (RP3B+) to a server (MacBook Air) in an open field. The sink was connected to a Netgear Nighthawk AC1750 (Model: R6700v2) router (WiFi AP) via a short Ethernet cable. The distance between the client and AP was increased in steps from 0.9m to 61m and at each step separate experiments were run with TCP and UDP and

the client uploading 1, 3, 5, 10, and 10,000 packets. Each experiment was repeated 200 times, apart from the 10,000 packet case at distances greater than 15m, which was only repeated 30 times due to increased execution time. The primary metric of interest used was the energy to transmit 1 MB from the client to the server (energy per MB). A number of related metrics were also collected to help understand the root cause for the communication-related power consumption. IoT traffic is generally expected to be either a periodic or event based traffic pattern. A majority of our experiments assumed continuous packet transmission and ignored the sleep cycle in between. We run the experiments consecutively since we were only concerned with the upload energy. For this, Iperf3 was used, as it enabled bandwidth management and conveniently generates the statistics we needed for our experiment. However we verified our models validity and results by performing additional periodic traffic with an idle period between bursts of transmission, to be more representative of IoT traffic. These results are discussed in Section 3.3.4.

Application Layer Experiments

For the application layer experiments three types of CoAP experiments are considered, simple CoAP (with no security features), CoAP with Encryption only (AES 128), and CoAP+DTLS. The setup is similar to Fig. 3.2 except an RP3B is used as as sink instead of a laptop. Two different IoT devices with different power capabilities were used as the client i.e. ESP32-DevKitC and RP3B. Experiments at distances 3m, 15m and 50m were performed outdoors in an open field while the 0.3m experiments were done indoors. Lightweight Tiny DTLS was used, and in each experiment the CoAP client sent 100 Confirmable requests.

3.3.4 Transport Layer Results

The maximum loss free rate, device base power requirements and the iperf energy overhead were first determined by experimentation. The maximum loss free rate for uploading data from the IoT devices to the sink was set to 44Mbps. Using this rate avoids wasteful work on the transmitting node if packets are dropped at the receiver because of UDP not having flow control. The WiFi channel was fixed to channel 6 for all the experiments. In addition, the RP3B+ idle base power with no peripherals connected and with WiFi active was 2.024W (with WiFi disabled it was 1.815W). Correspondingly, for the RP3B it was 1.165W (with WiFi disabled 1.099W). The power difference between the two systems is because the RP3B+ has a higher CPU clock rate of 1.4GHz compared to 1.2GHz for RP3B and supports both 2.4GHz and 5GHz WiFi. The fixed energy overhead contributed by iperf3 was 0.0001Wh. One issue to note is that although iperf3 periodically sends back delivery status updates to the sender the energy cost of transmitting the status updates could not be accurately obtained, therefore a fixed value for the iperf overhead was used. The results presented are limited to the experiments that exchange 10,000 packets, to amortize a number of the fixed overheads introduced by iperf3.

Fig. 3.3 shows the change in energy per MB as distance to the AP is increased. The UDP plot first shows the energy per MB to transmit 1MB of data i.e., energy per MB of data transmitted. But some of that data can be lost (channel or at the receiver). The $UDPDelivered$ plot shows the energy per MB for data delivered data i.e., energy per MB according to data actually delivered (lost packets not included). TCP contains reliability mechanisms to ensure data sent is delivered, and therefore the TCP plot for data sent and

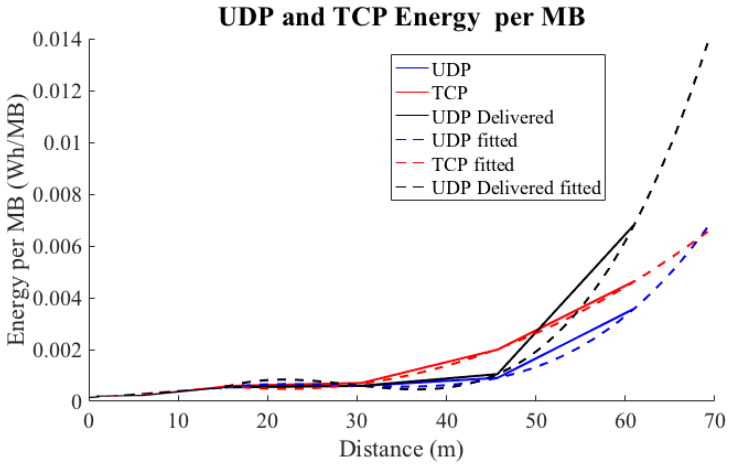

Figure 3.3: Energy per MB with TCP and UDP (for both transmitted and delivered data

delivered is the same and includes all data sent, including retransmissions. Although the signal strength decreases with distance (results not shown due to space), the energy cost per MB is fairly constant, initially, in all three plots at distances less than 15m and can be modeled as a linear relationship. However, as the distance grows beyond 15m, there is a significant increase in the energy required in all cases modeled by Equations 3.1, 3.2 and 3.3. In the equations, d is the distance (m) between the IoT device and AP and E_{UDP}, E_{TCP} and E_{UDP_D} are the energy per MB in each case. We consider the energy per MB cost from the perspective of data that was actually delivered by comparing TCP and $UDPDelivered$. At larger distances, the energy per MB is higher for UDP than TCP, because of UDP's packet loss which reduces the amount of data that actually reaches the sink. TCP's retransmissions effectively increase the amount of data that reaches the sink, thus effectively using the energy for delivering a given amount of data. We also consider the energy per MB cost from the perspective of all data sent from the IoT device, for TCP and UDP. At larger distances, TCP requires significantly more energy than UDP i.e., it is

more costly to send the same amount of data with TCP than UDP. A similar overall trend is seen with the experiments with less than 10,000 packets per experiment. The factors that are the underlying cause for the increase in transmission energy per MB as distance increases include retransmissions and link-layer ARQs.

The free space path loss model (Friis Transmission Formula) [42] indicates that path loss is proportional to the square of the distance d between the transmitter and receiver, or is proportional to $log(d)$ if working in dBm. The Okumura-Hata model [43] shows a similar log relationship between path loss and distance. A transmitter needs to adjust transmit power to ensure the received signal power is always greater than the receiver sensitivity, since path losses reduce the power of the signal received at the receiver. From the log plots of path loss vs. distance in the Friis and Okumura-Hata models, we observe two regions, which is also seen in our results. In the first region, path loss increases quickly but since this loss does not exceed the receiver sensitivity, the additional transmission energy needed per MB is small. In the second region, the rate of increase in path loss (in dBM) has reduced but because of the log relationship and the fact that the receiver sensitivity has been exceeded, even a small change in path loss incurs much higher transmission energy. This is seen as the exponential increase in energy per MB beyond 15m in our results. Note that although the Okumura-Hata model is typically used in a cellular environment, and for $d \geq 1$km, our model at larger distances can also be modeled by a log relationship and would

therefore exhibit similar behavior to the Okumura-Hata model.

$$E_{UDP} = \begin{cases} 2.356 \times 10^{-05} d + 0.0001538, \text{ if } d \leq 15m \\ 9.883 \times 10^{-08} d^3 - 8.491 \times 10^{-06} d^2 \\ +0.0002313d - 0.001362, \text{ otherwise.} \end{cases} \tag{3.1}$$

$$E_{TCP} = \begin{cases} 2.809 \times 10^{-05} d + 0.0001386, \text{ if } d \leq 15m \\ 2.704 \times 10^{-06} d^2 - 0.0001181d + 0.00177, \text{ otherwise.} \end{cases} \tag{3.2}$$

$$E_{UDP_D} = \begin{cases} 2.446 \times 10^{-05} d + 0.0001515, \text{ if } d \leq 15m \\ 2.302 \times 10^{-07} d^3 - 2.018 \times 10^{-05} d^2 \\ +0.0005516d - 0.003979, \text{ otherwise.} \end{cases} \tag{3.3}$$

The first factor causing increase in transmission energy that we consider is the observed throughput. From the experiments, the receive rate remains high for small distances, but then drops drastically from approximately 40Kbps to 20Kbps above a distance threshold i.e. approximately 15m (results not shown due to space limitation). The drop in throughput results in longer transmission time to send the same data (i.e., 10,000 packets) and thus more energy is spent by the IoT device.

The second factor affecting the energy per MB, (in the case of TCP), are the transport layer retransmissions due to the residual packet loss after link layer ARQs. Fig. 3.4 shows the transport layer losses for UDP and retransmissions for TCP with increasing distance as the average percentage of the total packets sent that are lost or retransmitted per experiment (i.e., the percentage of packets lost with UDP or retransmitted with TCP when sending 10,000 packets). For both transports, the loss/retransmission rate is fairly constant

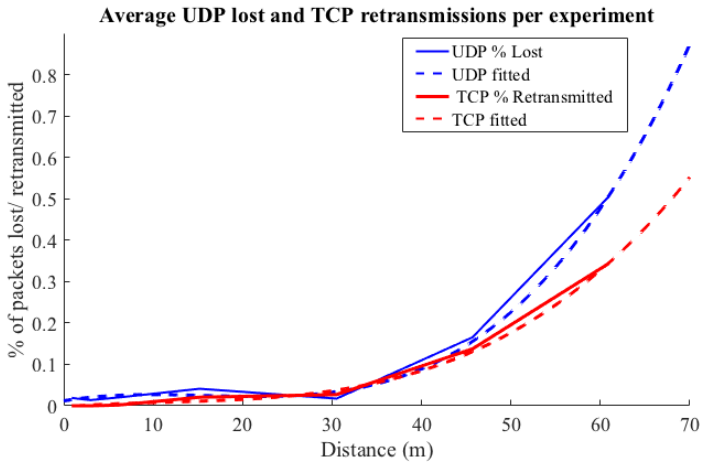

Figure 3.4: Average lost/retransmitted packets with distance (% of packets sent per experiment

and low up to around 30m before there is an exponential increase at longer distances. At 60m there is a 0.5% loss for UDP and 0.34% retransmission rate for TCP. This residual packet loss recovered by retransmissions at the TCP layer, although small, still requires additional energy compared to UDP.

The third factor affecting the energy per MB is the MAC layer ARQ that implements reliability at the MAC layer. Fig. 3.5 shows the link layer retransmission count (ARQ) as a percentage of the total data traffic sent from the RP3B+ to the sink and this increases for both TCP and UDP as distance increases. The ARQ percentage per experiment exceeds 10% at 6m and increases to approximately 24% for UDP and 31% for UDP at 60m. The values for TCP are larger than UDP, due to additional ARQs for the TCP retransmissions. Although the ARQ % increases from 3m onward, we only see a significant jump in energy per MB from 15m, when the ARQ is about 12% for UDP and 20% for TCP. Thus, only having a large % of ARQs (above a threshold) begins to have a significant im-

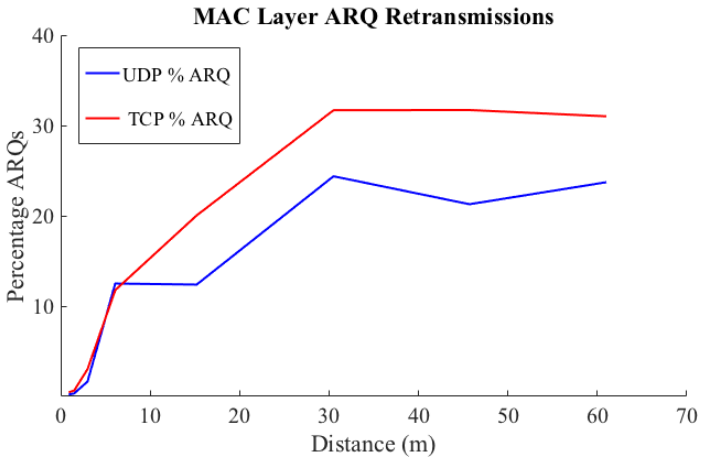

Figure 3.5: MAC Layer ARQ retransmissions as a % of total data traffic

pact on energy. At shorter distances (<15m) the energy per MB, bit rates and percentage loss/retransmission rates are similar for UDP/TCP and the differences in terms of the reliability mechanisms do not have too much of an impact on energy. The 15m distance is the maximum distance at which the energy cost is agnostic to the transport protocol selected and is dependent on the communication technology, device characteristics and application tolerance for loss. Below this maximum distance, the energy per MB with distance can potentially be modeled by a linear relationship. In this work we have selected 15m from Fig. 3.3 but depending on the application tolerance, 30m is also a possible distance. At distances larger than 15m, there is an exponential increase in energy per MB, but UDP consistently requires less energy than TCP because of the TCP retransmissions. For IoT applications that have some loss tolerance, UDP can be an attractive and energy efficient option at distances greater than 15m.

The high ARQ percentage for UDP (24%) and TCP (31%) at 61m indicates that the MAC layer is working hard to achieve reliability. For UDP this may be a waste of en-

ergy but if ARQ was not implemented, UDP would probably see a much higher, potentially unacceptable loss rate. In the case of TCP, the MAC ARQs help keep the residual loss rate low, but there is still duplication of functionality. For example, where TCP sends a retransmission and the MAC ARQ also has a retransmission for that retransmission, this increases energy consumption more. However, without ARQs, the high loss rate would result in very poor TCP performance. The effect of TCP retransmissions on ARQ retransmissions illustrates the cascading effect on energy across layers due to additional features. This is exacerbated in some protocols like MQTT which have retransmissions being generated at three layers: application, transport and MAC with retransmissions at the lower layer compounding the energy cost of retransmission from the higher layer. Some protocols like CoAP over TCP have tried to address this by having the transport layer handle reliability but the ARQ retransmission issue still remains. A possible solution to this is an adaptive reliability mechanism that reduces the duplication by TCP and ARQ retransmits and removes excessive ARQs for UDP. This is the focus of our future work.

IoT traffic is typically characterized by periodic transmissions and therefore we examined how valid our results with Iperf3, and the model derived from those measurements are, by performing additional IoT-like traffic which generates traffic periodically. Using Python scripts and TCP and UDP sockets, we transmitted data periodically from the RP3B+ to the sink with an idle period of 4s between successive transmissions. This was repeated multiple times and for data sizes 512KB and 1MB.

When using TCP traffic, our results show that with small amounts of data the overhead due to the data transfer protocol's setup and tear down as well as TCP connec-

tion setup and close is significant. These cause the energy per MB cost to increase very significantly. However, sending larger files, e.g., 1MB and above, this overhead is amortized and the energy per MB cost is similar to the values seen within our model derived from Iperf3 measurements. Table 3.2 compares the energy per MB values obtained from these experiments with the estimated energy cost from our model given in Equation (3.2) in the first row of the table. Since our focus is on IoT devices that are energy constrained but have more compute and communication capability than the traditional resource constrained sensors, data transmissions of 1MB and above are very possible especially when multimedia data is required. Therefore our model will prove useful in such scenarios. Nonetheless, with the same methodology, we will be able to derive a model for shorter transfers as well.

Results using UDP as the transport show a similar pattern, with the energy per MB cost for small data transfers, i.e., 512B being higher than our estimates based on Equation (3.1) in Row 1. This is due to the overhead for marshalling and packetizing the data. However, as the data size transferred increases, the measured energy per MB value increases, approximating our estimate since the added overheads are amortized. Based on the model we developed in our field measurements, at very small distances, the energy consumption for UDP is slightly higher than TCP. This is reflected in our estimates. However, the actual measurements for IoT-type traffic show that using TCP for data transfer consumes more energy. This is due to overheads for setting up the data transfer as well as the TCP connection setup and tear down for each transfer.

Table 3.2: Average energy per MB for data upload

Experiment	TCP (Wh/MB)	UDP (Wh/MB)
Estimated energy using model	0.000166	0.000176
512B	0.003248	0.001183
1MB	0.000115	0.000100

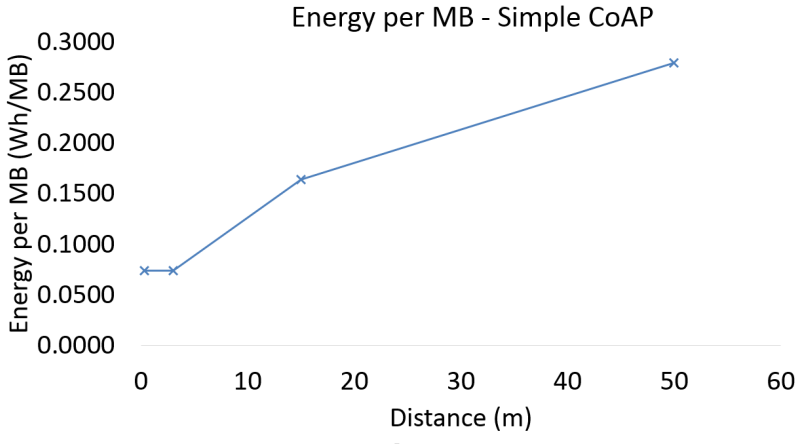

Figure 3.6: CoAP: Energy per MB vs Distance

3.3.5 Application Layer Results

For the CoAP experiments, we conduct experiments similar to the transport experiments described in Section 3.3.4. Note that delay and loss were implemented using Linux Traffic Control (TC) and these experiments were run in CoAP Confirmable mode. These experiments also showed a similar increase in energy per MB beyond 15m and with the increase in the distance from 0.3m to 50m, the energy per MB increases by 280.1 % (Fig. 3.6). However, because of constraints on experiments with distances beyond 15m and the larger step size, the data from the measurements are insufficient to show the exponential increase adequately. Fig. 3.7 shows the power consumption (in Watts) for CoAP under three different security settings: (i) no security (CoAP), (ii) encryption only (CoAP with

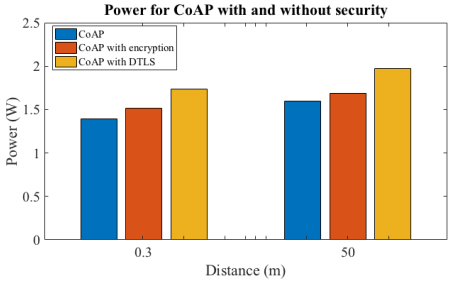

Figure 3.7: Power consumption at 0.3 and 50m

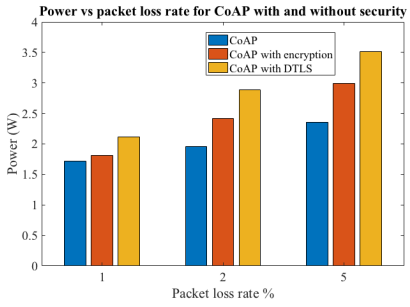

Figure 3.8: Power variation with loss

Encryption), and (iii) with DTLS (CoAP-DTLS). Addition of simple encryption increases the power consumption slightly (5.53 % over no encryption case). However, the use of DTLS increases the power consumption quite significantly (23.7 % over no encryption case) which is approximately 4x the cost of simple AES. This is due to the extra messaging overhead associated with DTLS, as illustrated in Fig. 3.1. Experiments performed comparing the energy used with CoAP (no security) and CoAP-DTLS show that the CoAP-DTLS overhead is approximately 2x CoAP (no security). We repeat the experiment with ESP32 as the client and observe similar results. From Fig. 3.7, we observe that the power consumption increases slightly with distance. For CoAP, this corresponds to an increase of 15.7 % in power consumption (Watts) as the distance increases from 0.3m to 50m. However, when measured in terms of energy per MB, the impact of distance is more pronounced, as seen earlier in Fig. 3.6.

Network impairments were emulated using Linux Traffic Control and Fig. 3.8 shows the impact of loss on power consumption when we use CoAP with and without security. Generally as the loss increases, all three CoAP configurations show an increase in required power due to additional transmissions required by the protocols. DTLS still re-

quires more power than simple encryption and experiments with delay also show a similar trend. Loss and delay exacerbate the cross layer energy effect. Therefore, one recommendation is that in cases where there are poor network conditions, complex protocols that require fewer messages to be transmitted or protocols that have less cross layer energy effect may be preferred in order to save energy.

3.4 Conclusion

In this paper, we experimentally investigate how increasing protocol features related to increased energy usage in IoT devices. Our transport layer experiments with TCP, UDP over WiFi indicates that distance between the IoT device and the gateway plays a critical role in this energy consumption: when this distance is less than 15m, the difference in energy used by TCP and UDP is not too significant. However, at larger distances, energy to send the same amount of data increases exponentially for both protocols, with UDP requiring less energy to send the same amount of data than TCP for the same distance. This result is important because it is indicative of the expected behaviour when these transports are used with other IoT application protocols, e.g., MQTT over TCP. This knowledge can then be used when making decisions regarding the desired IoT protocol and its features. Our experiments at the application layer includes studying the energy usage of CoAP with additional security (encryption and authentication) features. The results show that the energy cost of adding DTLS to CoAP can be quite significant.

Overall, our experiments show that adding features incurs significant energy, especially when these features require additional message exchanges and when features at one

layer impose a cross layer energy consumption in another. In an IoT environment, it may be preferable to increase packet sizes rather than number of messages. Careful consideration should be made when adding features at one layer to avoid the cross layer energy effect. This requires balancing the energy use against the benefit of a more complex protocol stack to optimize performance.

Chapter 4

Efficient and Lightweight Model-Predictive IoT Energy Management

4.1 Introduction

IoT applications are becoming increasingly complex, taking advantage of the better computing and communication capabilities in multi-sensor IoT devices. For example in modern agriculture, effective monitoring may require utilizing temperature and soil moisture sensors and a camera, all on a single device. The temperature and soil moisture readings together with the image and video data can then be used by the IoT applications to monitor plant growth, improve crop yields, control pests and detect intrusions, to ensure the safety and security of food sources. Sensors and IoT deployments for large-scale sensing have

attracted much attention; e.g., see [44] for a comprehensive survey of IoT-based agriculture solutions. However, large outdoor deployments increasingly depend on renewable energy sources [45], [46], [47] and batteries to power the IoT devices, rather than 'brown' energy. Efficient energy management is necessary to then maximize device operational period and effectiveness while meeting application requirements. This includes utilizing power prediction [48], scheduling tasks only when there is sufficient energy available [49], and adapting workloads to match currently available battery energy [50].

The sole dependence on renewable energy sources and rapid adaptation of sensor applications to energy availability have also been addressed in recent work. For example, FarmBeats [51] is an IoT platform for data-driven agriculture and animal husbandry that utilizes multiple sensors and has a weather-aware design. [52] use a neural network approach to predict the hourly soil moisture content to address irrigation challenges for farmers. [53] propose and evaluate a platform for city-scale sensing.

All of these approaches try to make longer-range predictions on energy availability based on weather and historical solar energy availability. However, the local conditions at an IoT device can be very dynamic. For example, shade due to obstructions, plant growth, or even a cloudy day can make the conditions at a particular device change substantially over the period of a day. Moreover, small Lithium-ion batteries age and their ability to sustain energy constantly changes over time, and the system needs to adapt to these conditions. These aspects continue to be a challenge.

Given the dynamic nature of the sensing environment, *Model Predictive Control (MPC)* is an appealing method for adaptive and predictive scheduling of the sensing tasks.

Over the past couple of decades, MPC has been successfully used for predictive optimization of industry processes and management of energy-constrained systems, in applications as diverse as chemical plant automation [54], building management systems [55], and wireless networks [56]. However, MPC is generally too computationally complex to be implemented in a lightweight sensor device. In this paper, we pose the predictive energy management problem in a renewable-energy powered IoT device as an MPC, but show that it can be solved using a low-complexity algorithm that is suitable for implementation in IoT devices. The solution we present – **PERMIT**, a **P**redictive **E**ne**R**gy **M**anagement solution for **I**o**T** – is designed to dynamically adapt individual sensing tasks executed by the IoT device, e.g., temperature sensing, image, and video capture, to meet application goals given the frequently changing energy availability. Towards addressing the complexity of solving the MPC optimization step, PERMIT utilizes certain features in the model structure and a well-justified approximation guided by charging characteristics of real batteries. For developing this approximation, PERMIT breaks down the complex MPC problem into two sub-problems: a time-dependent energy allocation problem, and a task-dependent sensor scheduling problem, which are shown to be solvable in time complexity that is cubic in the number of adaptation time-points (epochs) and the number of sensors on the multi-sensor IoT device. The solution method is developed as a combination of an incremental max-min fair allocation method and a recursive dynamic programming-like procedure, which is a novel contribution of our work. PERMIT integrates this efficient optimization method with a simple parabola-based adaptive solar prediction method based on battery state change that does not have to depend on weather data (unlike [51]). As batteries age, they in-

creasingly exhibit unpredictable behavior, especially when going below a threshold voltage. Thus long-term predictions, considered in previous work, may be inadequate to capture such changes. Due to frequent updates and adaptation of both the solar prediction and the energy optimization steps (feasible due to the low complexity of these methods in PERMIT), long-term predictions of solar energy and battery energy availability become unnecessary.

We demonstrate the feasibility of online operation of PERMIT on lightweight IoT devices by implementing and experimenting on a Raspberry Pi equipped with a camera, temperature, humidity, and soil moisture sensors, powered by a low-cost solar panel. We also run extensive simulations conducted using empirical energy usage models developed from real data traces collected from measurements and experiments with the physical device. The simulation results show that PERMIT adapts quickly to environmental changes by modifying the sensing task parameters towards maximizing information utility given the current energy limitations. Our evaluations demonstrate that PERMIT's low-complexity algorithm approximates the exact MPC solution very closely. PERMIT performs significantly better than Signpost [53], a comparable IoT energy management solution especially under conditions of concern - limited solar energy availability, changing battery conditions and behavior - by allocating energy more fairly across the different sensing tasks.

4.2 System Model and Optimization Formulation

IoT devices that rely on battery power and renewable energy need to manage their energy carefully. However, they also need to satisfy application information requirements by operating their sensors and executing sensing tasks as and when needed (e.g., throughout

the day and night). Therefore the IoT device has to strike a delicate balance between maximizing information utility (using considerable energy) and maximizing device lifetime (minimizing energy use). The performance of an IoT device powered by a renewable energy source (such as solar) needs to be optimized subject to three factors. First is the current battery energy level (battery state), which we estimate by reading the current battery voltage at the IoT device. The second factor is an estimate of the total amount of solar energy available to recharge the battery from now till the end of the prediction horizon. Third is the energy goal or minimum battery state required at the end of the prediction horizon. Since solar energy follows a diurnal pattern anyway, the prediction horizon ends at 06:00 am the next day (beginning at the current time). We seek to ensure that the battery energy at the end of the prediction horizon is no less than the minimum battery energy level allowed on the device.

Given these three factors, the system optimization goal is to determine the maximum sensing task parameter values that can be used throughout the prediction horizon subject to meeting the end-of-horizon battery constraint. If the battery energy and solar energy are low, the adaptive MPC approach reduces the task parameter values (sensing rates) to extend the device's lifetime. However, if high solar energy is predicted then higher task parameter values are used since the MPC forecasts the energy used will be reclaimed later in the day from the solar energy.

In this work, motivated by agricultural applications [44], we consider four tasks; video, image, temperature, and humidity with task parameters as: video duration t_v; image quality q (amount of image compression); and measurement frequency (n_t and n_h)

respectively. In Sec. 4.3.6 we empirically characterize the energy requirements for each task through experimental measurements, which is then utilized by Model Predictive Control (MPC) [57] to optimize the sensing operations during the day and ensure the battery is sufficient to operate throughout the night. At every epoch, MPC optimizes operations over a time horizon into the future (until 06:00 the next morning), considering the current state (battery energy) and predictions on future solar charging rates. Thus, our approach adapts to solar prediction errors, and the resulting changes in the battery energy estimates.

Consider an IoT device required to perform a set $\mathcal{K}$ of tasks (here we use video (v), image (i), temperature (t), and humidity (h), but this can be extended to include other sensors, e.g., soil moisture sensors) to be executed. Each task $k \in \mathcal{K}$ is characterized by a parameter: video duration t_v, image quality q, and the number of measurements per epoch, n_t and n_h, for temperature and humidity respectively. Each task is associated with a utility function U_k which is a function of task k's variable parameter, and assigned weight w_k. The parameters U_k and w_k provide a measure of the value of the information provided by the task, e.g., for image data, a high q value implies a better-quality image (less image compression), giving a higher information utility U_i (see Sec. 4.3.5); if the image data also has a higher weight w_i, this implies the image task is more important (see below in Sec. 4.3.6).

Each task k requires an amount of energy, E_k, to execute, which is a function of the task parameters, as described in Sec. 4.3.5. Each epoch n lasts 15 minutes. We impose a constraint that the battery level at the end of the night (06:00) must be no less than E_{min} (minimum IoT device battery level possible before shut down). The following formulation

represents the MPC optimization at any epoch n from $n_0 = 0$ at 06:00, to $N = 95$ at 05:45 next morning.

$$\max_{f_k(n) \geq f_{k,min}} \sum_{n=n_0}^{N} \sum_{k \in \mathcal{K}} w_k U_k(f_k(n)), \tag{4.1}$$

subject to

$$E(n) = \min\left[E(n-1) + R(n), E_{max}\right], \ \forall n, \tag{4.2}$$

$$R(n) = \min\left[S(n) - \sum_{k \in \mathcal{K}} E_k(f_k(n)) - E_{base}, R_{max}(n)\right], \ \forall n, \tag{4.3}$$

$$R_{max}(n) = \frac{IeE(n)v_{max}}{E_{max}}, \ \forall n, \tag{4.4}$$

$$E(n) \geq E_{min}, \ \forall n. \tag{4.5}$$

In the above, f_k represents the sensing optimization variables for each task k, $R(n)$ is the energy pushed into or taken out of the battery during epoch n, $E_k(f_k(n))$ is the total energy required by task k in epoch n, E_{max} and E_{min} are the maximum and minimum battery energy levels allowed at any time, $S(n)$ is the total predicted solar energy available for use during epoch n, and E_{base} is the energy required by the IoT device while idle (only taking power measurements) during one epoch. Further, $R_{max}(n)$ is the recharge limit of the battery (i.e., the max. energy that can be pushed into the battery in epoch n), v_{max} is the maximum battery voltage (4.2V), e is the epoch duration in seconds, I is the current flowing to the battery, and $f_{k,min}$ is the minimum sensing level that must be maintained for sensing task k.

The objective function in Eq. (6.3) maximizes the total information utility obtained through sensing, under the constraints in Eqs. (4.2)-(4.5). Eq. (4.2) represents evolution of the battery energy, where $R(n)$ is given by Eqs. (4.3)-(4.4). The term $S(n) -$

$\sum_k E_k(f_k(n)) - E_{base}$ represents the amount of energy that can be utilized for recharging the battery, once the energy consumed by the sensing tasks is accounted for. $R_{max}(n)$ depends on the energy level of the battery, as given by Eq. (4.4); The lower limit on the battery energy level is given by Eq. (4.5); the upper bound E_{max} is enforced by Eq. (4.2).

4.3 The PERMIT Framework

The MPC optimization posed in Eqs. (6.3)-(4.5) is a convex optimization problem. This implies that a generic convex optimization solver (possibly running at a remote server) could solve the MPC problem efficiently once the battery state is communicated to it by the sensor device. However, a more efficient algorithm for solving the MPC optimization problem would be required when it has to be carried out on the sensor device itself. Further, with standard methods for solving this MPC problem, such as the iterative primal-dual approach or dynamic programming, there will be a tradeoff between the computation time (or the number of iterations) and the degree of optimality. We develop an approach that can solve the problem of Eqs. (6.3)-(4.5) very efficiently (i.e., in time complexity that is cubic in number of epochs) to a very close degree of the optimum, with a slight approximation of the constraint in Eq. (4.4). This approach utilizes the structure of the MPC optimization problem to perform a bi-level decomposition.

4.3.1 Bi-level Decomposition

Let $f = (f_k(n), k \in \mathcal{K})$ denote the vector of all sensing variables over all epochs and let the set of constraints Eqs. (4.2)-(4.5) be represented compactly as $f \in F$. Note

that the sensing variables $f_k(n)$ in epoch n only appear in the energy constraints in the form $\sum_k E_k(f_k(n))$ $(= g(n)$, say). Let $g = (g(n), n = 1, \ldots, T)$, and $g_{min} = \sum_k E_k(f_{k,min})$. Then constraint $f \in F$ can be equivalently written as $g \in G$, where G is obtained from the constraint set F, by replacing the term $\sum_k E_k(f_k(n))$ in (4.3) by $g(n)$. We re-write the optimization problem in (6.3)-(4.5) as Eqs. (4.6)-(4.7):

$$\max_{g \in G, g(n) \geq g_{min}} \sum_{n=n_0}^{N} V(g(n)), \quad \text{where} \tag{4.6}$$

$$V(g(n)) = \max_{\substack{\sum_k E_k(f_k(n)) = g(n), \\ f_k(n) \geq f_{k,min}}} \sum_{k \in \mathcal{K}} w_k U_k(f_k(n)). \tag{4.7}$$

This decomposition allows us to develop the algorithm in two phases (levels): (i) *Energy allocation per epoch (EA)*, where Eq. (4.6) is solved and allocates the optimum energy $g^*(n)$ across the epochs n; (ii) *Sensor scheduling within an epoch (SS)*, where Eq. (4.7) is solved for each epoch n to divide up the energy $g^*(n)$ among the different sensing tasks $f_k(n)$ in that epoch. Note that SS (lower-level problems, one for each epoch) is embedded into the EA problem (higher-level problem, just one for the entire optimization period). For ease of understanding, we describe the SS phase first.

Sensor Scheduling Within an Epoch (SS)

Towards solving the SS problem efficiently, we observe that based on the task energy expressions obtained from our experiments (see Section 4.3.6), the function $E_k(f_k)$ can be expressed as $E_k(f_k) = E_k^0 + E_k^1 f_k$ (for appropriately chosen constants E_k^0 and E_k^1) when k corresponds to the video, temperature, and humidity sensing tasks. For the image sensing task, the dependence of $E_k(f_k)$ on the sensing parameter $f_k(= q)$ is quadratic however a linear fit also works quite well in practice (see Section 4.3.6). In our case, the

utility functions $U_k(f_k)$ have a simple structure of the form $1 - e^{-\alpha_k f_k}$, where constant α_k depends on the sensing task k (see Section 4.3.6). Further, note that the constraint set comprises a single linear constraint $\sum_k E_k^0 + E_k^1 f_k(n) = g(n)$, in addition to the lower bounds $f_k(n) \geq f_{k,min}, \forall k$. For any epoch n, when $g(n)$ is given, the optimal sensing variables $f_k(n)$ can be obtained by Algorithm 4.1, which has a complexity of $O(K^2)$ per epoch, for $K = |\mathcal{K}|$ sensing tasks. Algorithm 4.1 first calculates the solution that maximizes $\sum_k U_k(f_k(n))$ subject to the constraint $\sum_k E_k^0 + E_k^1 f_k(n) = g(n)$; this is given by line 10 of the algorithm. If any of the resulting $f_k(n)$ values are less than or equal to $f_{k,min}$, they are set to $f_{k,min}$, and the process repeats for the other sensors until all remaining sensors have $f_k(n) > f_{k,min}$.

To obtain the equation in line 10, we use Lagrange multipliers, set the partial derivatives of the Lagrangian to zero, and solve for f_k. The correctness of Algorithm 4.1 in computing the optimum solution in Eq. (4.7) follows easily, and is formally stated below.

Theorem T.1 *Algorithm 4.1 computes the optimum solution of the SS problem as expressed by Eq. (4.7).*

Proof. We first show that the solution maximizing $\sum_{k\in\mathcal{K}} w_k U_k(f_k) = \sum_{k\in\mathcal{K}} w_k(1-e^{-\alpha_k f_k})$ subject to constraints $g(n) = \sum_{k\in\mathcal{K}} E_k(f_k) = \sum_{k\in\mathcal{K}} E_k^0 + \sum_k E_k^1 f_k$ is given by the formula in line 10 of Algorithm 4.1. The correctness of Algorithm 1 follows directly from that result.

We have

$$g(n) = \sum_{k \in \mathcal{K}} E_k(f_k) \tag{4.8}$$

$$= \sum_{k \in \mathcal{K} \setminus \mathcal{K}'} (E_k^0 + E_k^1 f_{k,min}) + \sum_{k \in \mathcal{K}'} (E_k^0 + E_k^1 f_k)$$

$$= E' + \sum_{k \in \mathcal{K}'} E_k^1 f_k.$$

$$V = \sum_{k \in \mathcal{K}} w_k U_k(f_k) = \sum_{k \in \mathcal{K}'} w_k (1 - e^{-\alpha_k f_k}) + \text{constant}. \tag{4.9}$$

Let $C = 0$ denote the constraint in Eq. (4.3.1), i.e., $C = g(n) - (E' + \sum_{k \in \mathcal{K}'} E_k^1 f_k)$. If the Lagrange multiplier associated with the constraint in Eq. (4.3.1) is λ, then setting the partial derivatives of the Lagrangian $V + \lambda C$ to zero gives $\frac{\partial V}{\partial f_k} = w_k \alpha_k e^{-\alpha_k f_k} = \lambda E_k^1$, or

$$f_k = \frac{\ln(\frac{w_k \alpha_k}{E_k^1}) - \ln \lambda}{\alpha_k}, \; k \in \mathcal{K}'. \tag{4.10}$$

Solving for λ using f_k from Eq. (4.10) and the constraint $g(n) = E' + \sum_{k \in \mathcal{K}'} E_k^1 f_k$ from Eq. (4.3.1) gives

$$\ln \lambda = \frac{\sum_{k \in \mathcal{K}'} \frac{E_k^1}{\alpha_k} \ln(\frac{w_k \alpha_k}{E_k^1}) - (g(n) - E')}{\sum_{k \in \mathcal{K}'} \frac{E_k^1}{\alpha_k}}. \tag{4.11}$$

Replacing $\ln \lambda$ in Eq. (4.10) gives us the expression for f_k in line 10 of Algorithm 4.1, for for $k \in \mathcal{K}'$. It is easy to verify that for $k \in \mathcal{K} \setminus \mathcal{K}'$, the derivatives of the Lagrangian w.r.t. f_k negative, justifying $f_k = f_{k,min}$ for $k \in \mathcal{K} \setminus \mathcal{K}'$. Therefore the f_k values resulting from Algorithm 4.1 (when it completes), with the Lagrange multiplier computed as in Eq. (4.11), maximize the Lagrangian, and are also feasible. Therefore, they solve the SS problem expressed in Eq. (4.7) optimally. ■

4.3.2 Energy Allocation Between Epochs (EA)

The detailed development of an efficient algorithm for the EA problem and its correctness analysis are quite involved. We provide the development here, including the lemmas and corollaries that lead to the main result (Theorem T.2).

Developing an efficient solution to Eq. (4.6) relies on a critical but reasonable approximation. Note that while $R_{max}(n)$ varies linearly with $E(n)$ (as seen in Eq. (4.4)), the slope is nearly flat in the constant-current phase for charging the battery, as observed in prior work [58], [59], [60] and verified experimentally by us (see Fig. 4.2).

Algo 1: Sensor Scheduling in epoch n (SS)

Result: $f_k(n)$ that attains the maximum in (7), given $g(n)$.

Initialize: $\mathcal{K}' = \mathcal{K}$, *done* = FALSE;

if $g(n) \leq g_{min} = \sum_{k\in\mathcal{K}}(E_k^0 + E_k^1 f_{k,min})$ **then**

  $f_k(n) = f_{k,min}$; *done* = TRUE;

while *!done* **do**

  $E' = \sum_{k\in\mathcal{K}\setminus\mathcal{K}'}(E_k^0 + E_k^1 f_{k,min}) + \sum_{k\in\mathcal{K}'} E_k^0$;

  For $k \in \mathcal{K}'$, calculate

$$f_k(n) = \frac{1}{\alpha_k}\left[\ln\left(\frac{w_k\alpha_k}{E_k^1}\right) + \frac{g(n) - E' - \sum_{k\in\mathcal{K}'}\frac{E_k^1}{\alpha_k}\ln\left(\frac{w_k\alpha_k}{E_k^1}\right)}{\sum_{k\in\mathcal{K}'}\frac{E_k^1}{\alpha_k}}\right];$$

  $\mathcal{K}'_+ = \{k \in \mathcal{K}' : f_k(n) > f_{k,min}\}$;

  if $\mathcal{K}'_+ = \mathcal{K}'$ **then** *done* = TRUE; **else** $\mathcal{K}' = \mathcal{K}'_+$;

end

Figure 4.1: *Sensor Scheduling in epoch n (SS)].*

Therefore, we can approximate R_{max} to be a constant: corresponding to the lowest value in the constant-current phase (conservative estimate), for example. This results in a simpler reformulation of the problem, expressed below.

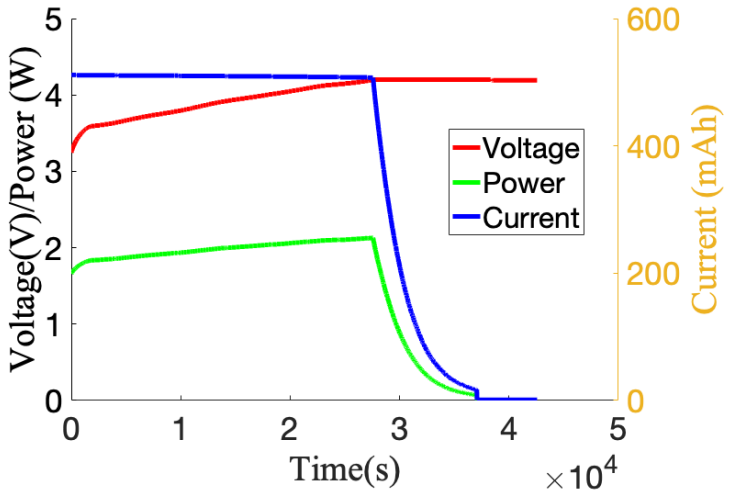

Figure 4.2: Li-ion Battery charging characteristics: When charging, the battery operates in three distinct regions: (a) pre-charging, (b) constant current, and (c) constant voltage regions. In phase (a), a small charging current is initially provided to the battery for a short time. In phase (b), a constant current is fed to the battery (as the voltage increases) until it reaches approximately 95% full. In phase (c), the battery is almost full, and charging progressively reduces to a trickle until the battery is full. The slope of the charging rate (power) is nearly flat with respect to the state of charge in phase (b), which is most of the charging period.

Lemma L.1 *If* $R(n) = R_{max}\ \forall n$, *then problem in (4.6)-(4.7) is equivalent to minimizing* $\sum_{n=n_0}^{N} V(g(n))$, *subject to* $g(n)$ *satisfying*

$$E_{min} \leq E(n_0 - 1) + \sum_{n'=n_0}^{n} \tilde{S}(n') - \sum_{n'=n_0}^{n} g(n') \leq E_{max}, \tag{4.12}$$

$$g(n) \geq \max[g_{min}, \tilde{S}(n) - R_{max}], \tag{4.13}$$

where $n \in [n_0, N]$ *and* $\tilde{S}(n) = S(n) - E_{base}$.

Proof. We show this result by analyzing the constraint set $g \in G$ and showing that it is equivalent to the constraints in Eqs. (4.12)-(4.13). Firstly note that the constraint $g \in G$ is derived from the constraints in Eqs. (4.2)-(4.5) by replacing $\sum_k E_k(f_k(n))$ in Eq. (4.3) by

$g(n)$. Since R_{max} is now a fixed value, Eq. (4.4) can be removed, and Eq. (4.3) becomes

$$\begin{aligned} R(n) &= \min\left[S(n) - g(n) - E_{base}, R_{max}\right], \forall n, \\ &= \min\left[\tilde{S}(n) - g(n), R_{max}\right], \ \forall n. \end{aligned} \tag{4.14}$$

We next show that imposing the constraint in Eq. (4.13) does not sacrifice optimality, and then use Eq. (4.13) and Eq. (4.3) to show that Eq. (4.12) can represent Eqs. (4.2)-(4.5).

Towards that end, we first observe that the battery energy $E(n)$ captures the state of the system, i.e., given the battery energy $E(n)$ at the end of epoch n, the objective function value derived from epochs $n+1$ onwards is independent of the history up to epoch n. Now, for the sake of contradiction, let us assume that Eq. (4.13) does not hold at optimality. Since $g(n) \geq g_{min} \forall n$, this implies that there exists an n such that $g_{min} \leq g(n) < \tilde{S}(n) - R_{max}$, or equivalently, $\tilde{S}(n) - g(n) > R_{max}$. Then we can increase $g(n)$ slightly without affecting $R(n)$ in Eq. (4.15). From (4.2), $E(n)$, we see that $E(n)$ (which depends on $g(n)$ only through $R(n)$) also remains unaffected. This implies that the objective function value at epoch n can be increased without affecting that obtained from future epochs. This is a contradiction, establishing that at optimality, $g(n) \geq \tilde{S}(n) - R_{max}$ holds for all n.

Now note that Eq. (4.13) implies that $R(n)$ in Eq. (4.15) can be written as $R(n) = \tilde{S}(n) - g(n)$. Utilizing this in Eq. (4.2), we get $E(n) = \min\left[E(n-1) + \tilde{S}(n) - g(n), E_{max}\right], \ \forall n$. Now, if $E(n-1) + \tilde{S}(n) - g(n) > E_{max}$, we can increase $g(n)$ slightly without altering $E(n)$. This would improve the objective function value derived from epoch n without affecting that obtained from future epochs, thereby contradicting the optimality of $g(n)$. Therefore, in any optimal solution, $E(n-1) + \tilde{S}(n) - g(n) \leq E_{max}$, implying that as far as the optimal

solution is concerned, Eq. (4.2) can be written as $E(n) = E(n-1) + \tilde{S}(n) - g(n) \leq E_{max}$. Iterating over n_0 to n, we get

$$E(n) = E(n_0 - 1) + \sum_{n'=n_0}^{n} \tilde{S}(n') - \sum_{n'=n_0}^{n} g(n'). \tag{4.15}$$

Imposing the upper and lower bounds (E_{max} and E_{min}, respectively) on $E(n)$ (as in Eqs. (4.2) and (4.5)) gives Eq. (4.12). This concludes the proof of the lemma. ■

Let the constraint set implied by Eqs. (4.12)-(4.13) be denoted by $\tilde{G}$. Lemma L.1 implies when $R(n) = R_{max}\ \forall n$, as far as optimality is concerned, constraints $g \in G$ and $g \in \tilde{G}$ are equivalent.

Under the above approximation, we have another important property: the solution of Eq. (4.6) is independent of the exact nature of the function V, as long as it is an increasing, strictly concave function with $V(0) = 0$. This fact allows us to solve the EA problem without solving the SS problem first (or as a sub-problem of the EA problem). Further, the solution to the EA problem (for all such strictly convex utility functions, including the V function in our case) turns out to be a max-min fair solution, but under additional constraints [61], i.e., it tries to maximize the minimum energy allocated to the different time slots n (lexicographically), subject to constraints $g \in \tilde{G}$. To formally express this, we define the notion of a *Constrained Max-min Fair (CMF)* solution. We then show that when $R(n) = R_{max}\ \forall n$, any CMF solution provides the optimal $g(n)$ values of the problem in Eqs. (4.6)-(4.7) is CMF, and vice versa. (Lemma L.3). Finally, we show that Algorithm 4.3 computes a CMF solution (Theorem T.2).

Definition D.1 *The vector of energy allocations across the epochs, g is said to be Constrained Max-min Fair (CMF) if g is feasible (i.e., $g \in \tilde{G}$), and increasing the components*

of g (while maintaining feasibility) would cause an equivalent amount of reduction in the other components of equal or lower value.

This definition of CMF is more restrictive than that of the traditional definition of max-min fairness [62] in that an increase in any component of g, say $g(n)$, by an amount δ would cause an *equivalent amount* of decrease (δ) in the value of other component(s) n' satisfying $g(n') \leq g(n)$. The following fact is also implicit in the definition: if multiple components of a CMF vector g are increased, the corresponding amounts of reduction in the other components (of equal or lesser value) are non-overlapping. As an example, for a CMF vector g, supposed two components $g(n_1)$ and $g(n_2)$ are increased by amounts δ_1 and δ_2 respectively. Then to maintain feasibility, some other component(s) n' must decrease by a total amount of $\delta_1 + \delta_2$, of which δ_1 amount of reduction must be in component(s) satisfying $g(n') \leq g(n_1)$ and δ_2 amount of reduction must be in component(s) satisfying $g(n') \leq g(n_2)$. This stronger version of max-min fairness is needed for showing that any CMF solution is optimal for our problem (see Lemma L.3 below).

Let OPT denote any allocation vector g^* that minimizes $\sum_{n=n_0}^{N} V(g(n)$ subject to $g \in \tilde{G}$ (i.e., g satisfying Eqs. (4.12)-(4.13)). Before proceeding further, we make the following simple observation.

Lemma L.2 *In both OPT and CMF, $E(N) = E_{min}$.*

Proof. For the OPT case, the above result follows from the fact that the objective function value $\sum_{n=n_0}^{N} V(g(n))$ does not depend on $E(N)$; therefore if $E(N) > E_{min}$, we can improve the objective function value by reducing $E(N)$ to E_{min} and using that energy to increase $g(N)$. The result for the CMF case follows similarly. ■

From Lemma L.2 and Eq. (4.15), we obtain the following Corollary.

Corollary C.1 *In both OPT and CMF,* $\sum_{n'=n_0}^{N} g(n') = \sum_{n'=n_0}^{N} \tilde{S}(n') + E(n_0 - 1) - E_{min}$.

The result that Corollary C.1 states is very intuitive: since in the optimal solution no energy will be wasted, the total energy used in sensing, $\sum_{n'=n_0}^{N} g(n')$, equals the total solar energy obtained (discounted by the base energy used to keep each sensor powered on, E_{base}), plus the difference between the initial ($E(n_0 - 1)$) and final ($E(N) = E_{min}$) battery energy states. Next we state the following result establishing the connection between the optimal solutions of our problem and CMF energy allocations.

Lemma L.3 *An energy allocation vector g is CMF if and only if it is an OPT.*

Proof. Firstly, note that the function V is monotonically increasing and strictly concave while satisfying $V(0) = 0$. For the sake of contradiction, let us assume that the claim in the lemma is not true. Let $\hat{g}$ denote a CMF solution, and $g^* \neq \hat{g}$ be an OPT. Let S be the set of epochs for which $g^*(n) > \hat{g}(n)$. Since $\hat{g}(n)$ is CMF, from Definition D.1 it follows that as we increase the $\hat{g}(n)$ values to $g^*(n)$ for $n \in S$, there is an equivalent amount of decrease in other components (epochs) of equal or less value. From Corollary C.1, it follows that as we go from $\hat{g}(n)$ to $g^*(n)$, these increments must match the decrements. Since the objective function V is monotonically increasing, strictly concave and independent of n, the gain in the objective function thus obtained is offset by the loss from it due to the corresponding decrements. Therefore $\sum_n V(g^*(n)) \leq \sum_n V(\hat{g}(n))$, which cannot happen if g^* is OPT and $\hat{g}$ is not an OPT. Thus we arrive at a contradiction. Since V is strictly concave, g^* is unique. Therefore, this implies that $\hat{g}$ and g^* are both unique, and must coincide. ■

The result in Lemma L.3 is intuitive: since the function V is strictly concave and independent of n, the objective function $\sum_n V(g(n))$ is maximized when the $g(n)$ values are equalized as much as possible without wasting energy, which by definition corresponds to the CMF solution.

Lemma L.3, and the fact that the CMF solution does not depend on the exact form of the utility function V, results in the following corollary.

Corollary C.2 *If $R(n) = R_{max}\ \forall n$, then the solution to the problem in (4.6)-(4.7) is independent of the exact form of the function V, as long as it is an increasing and strictly concave function with $V(0) = 0$.*

Before we state the main result on the correctness of Algorithm 4.3, we make a few observations. Note that the approximate EA problem can be solved using dynamic programming but at high state and space complexity. This is because the state (variable) in this case, the battery energy $E(n)$ varies over a continuous space, even though the time steps (epochs) are discrete (N). On the other hand, algorithms for the max-min fair allocation problem [62] do not work in this case due to the existence of both lower and upper bounds on $\sum_{n'=n_0}^{n} g(n')$ in Eq. (4.12). In particular, the lower bounds on $\sum_{n'=n_0}^{n} g(n')$, due to the right-hand constraints in (4.12) (involving E_{max}), pose a technical challenge in applying the max-min fair allocation solutions to this problem. However, the basic principle of the step-wise bottleneck-based algorithm for computing max-min fair rates still applies, and an efficient bottleneck-based algorithm can be developed, exploiting the nested nature of the constraints in Eq. (4.12). We do so by combining the bottleneck-based algorithm principle with a recursive dynamic programming-like approach, but pivoting it at $\tilde{E}(n)$, the maximum

level of the battery state at any epoch n, as given by line 2 of Algorithm 4.3. Pivoting it at $\tilde{E}(n)$ avoids violation of the upper bound on the battery energy level (E_{max}); the algorithm increases the $g(n)$ values gradually (and as equally as possible subject to constraints in Eq. (4.13), ensuring that the lower bounds on the battery energy level (E_{min}) is also satisfied at all times. This is the intuition behind Algorithm 4.3, which computes a CMF solution (and therefore computes an OPT), and is captured in the proof of Theorem T.2, stated below. Theorem T.2 argues that Algorithm 4.3 computes a CMF solution; and therefore by Lemma L.3 it also computes an OPT.

Theorem T.2 *Assuming that constraints (4.12)-(4.13) are feasible, the energy allocation vector g computed by Algorithm 4.3 is CMF.*

Proof. We prove by induction over n from n_0 to N. Towards that, we first define the problem $\mathbf{P}_k(E)$ as one that computes the max-min fair rates $g^k = (g(n'), n' \in [n_0, k])$, subject to

$$E_{min} \leq E(n_0 - 1) + \sum_{n'=n_0}^{n} \tilde{S}(n') - \sum_{n'=n_0}^{n} g(n') \tag{4.16}$$

$$\leq E_{max}, \quad n \in [n_0, k-1],$$

$$E(n_0 - 1) + \sum_{n'=n_0}^{k} \tilde{S}(n') - \sum_{n'=n_0}^{k} g(n') = E, \tag{4.17}$$

$$g(n) \geq \max[g_{min}, \tilde{S}(n) - R_{max}], \quad n \in [n_0, k]. \tag{4.18}$$

From the above definition of $\mathbf{P}_k(E)$, and Lemma L.2, solving $\mathbf{P}_N(\tilde{E}(N))$, computes OPT (which by Lemma L.3 is a CMF solution); recall from Algorithm 4.3 that $\tilde{E}(N) = E_{min}$.

Next we show that Algorithm 4.3 correctly computes $\mathbf{P}_N(\tilde{E}(N))$, by induction over $k = n_0, \cdots, N$. A challenge in this induction step is that while the assumption that

the constraints in Eqs. (4.12)-(4.13) are feasible implies that the constraints in Eqs. (4.16)-(4.18) are feasible for $k = N$, it does not imply the feasibility of the constraints in Eqs. (4.16)-(4.18) for any $k \in [n_0, N-1]$. In other words, the problem $\mathbf{P}_k(\tilde{E}(k))$ may not be feasible for $n_0 \leq k < N$. To address this issue, we define the set of feasible epochs F, as those epochs k for which $\mathbf{P}_k(\tilde{E}(k))$ is feasible, i.e., $F = \{k \in [n_0, N] : \text{constraints (4.16)} - \text{(4.18) are feasible for } k\}$. Clearly F is non-null since by assumption, constraints (4.16)-(4.18) are feasible for $k = N$. Intuitively, a feasible epoch k is one for there is enough solar supply in epochs up to (and including) epoch k so that the battery energy can reach $\tilde{E}(k)$ (which is E_{max}, except for $k = N$, for which $\tilde{E}(N) = E_{min}$) at the end of epoch k, while satisfying the battery upper and lower bound constraints (4.16) for epochs $n = [n_0, k-1]$ (Eq. (4.16)) and the lower bounds on $g(n)$ (Eq. (4.18)). Clearly, if $E(n_0 - 1)$ (initial battery state) is low, we can expect that the first few epochs may be infeasible. The problem $\mathbf{P}_k(\tilde{E}(k))$ may become feasible at some intermediate epochs k, and then finally at $k = N$ as well.

We next proceed to show by induction that the **for** loop in lines 4-25 of Algorithm 4.3, when executed up to a feasible epoch $n = k$, computes a solution to $\mathbf{P}_k(\tilde{E}(k))$.

Base step: Let k_1 be the first epoch in F, i.e., the first epoch k for which $\mathbf{P}_k(\tilde{E}(k))$ is feasible. For $n \in [n_0, k_1 - 1]$, the condition in the while loop in line 5 is not satisfied, and therefore the entire while loop is skipped; this implies that when approaching the for loop with $n = k$, the $g(n)$ values for $n \in [n_0, k_1 - 1]$ remain at their lower bounds, $\max[g_{min}, \tilde{S}(n) - R_{max}]$. Some of these $g(n)$ values for $n \in [n_0, k_1 - 1]$, along with the $g(k_1)$ value, may get updated within the while loop for $n = k_1$, which we analyze next.

Note that in Algorithm 4.3, the energy allocations get updated only in lines 17 and 21. It is then easy to see that the energy allocation values for each epoch either increases or remains the same through the run of the algorithm. For $n = k_1$, for each run of the while loop, in line 16, either g is set to g_n or one of the g_m values (for $m = [n_b + 1, n - 1]$). In the former case, the condition of the while loop (line 5) is not satisfied in the next round, and the for loop with $n = k_1$ concludes. In the latter case, the bottleneck epoch, n_b, is updated; after that the $g(m)$ values of any epoch $m \leq n_b$ do not change for the rest of the algorithm run. Therefore in general, within the for loop with $n = k_1$, the while loop runs multiple times, and the bottleneck epoch gets updated in each round as $n_{b_1}, n_{b_2}, \cdots, n_{b_l}$, except for the last round. In the first round, when the bottleneck epoch gets updated to n_{b_1}, all epochs $m \in [n_0, n_{b_1}]$ are assigned the same $g(m)$ value $g = g_{n_{b_1}}$, subject to their respective lower bounds. These $g(m)$ values do not further change through the run of the algorithm. Since the battery energy $E(n_{b_1}) = E_{min}$ at this point, increasing any of these $g(m)$ values, say for $m = m'$, must be accompanied with decreasing some of the other $g(m)$ values for $m \in [n_0, n_{b_1}], m \neq m'$, so that $E(n_{b_1})$ does not fall below E_{min}. Now consider the epochs $m \in [n_{b_1}, n_{b_2}]$. These epochs all "bottleneck" at epoch n_{b_2}. Increasing any one of these $g(m)$ values, say $m = m'$ would require decreasing some other $g(m)$ values for $m \leq n_{b_2}$. For the epochs $m \in [n_{b_1}, n_{b_2}]$, the $g(m)$ values are the same as $g(m')$. Also since the $g(m)$ values for $m \in [n_{b_1}, n_{b_2}]$ were already assigned the same value as those of epochs $m \in [n_0, n_{b_1}]$ when those epochs bottlenecked at epoch n_{b_1}, and the $g(m)$ values are non-decreasing over the run of the algorithm, the $g(m)$ values of epochs $m \in [n_0, n_{b_1}]$ are less than or equal to $g(m')$. Therefore, as increase in $g(m')$ must be accompanied with decreasing a $g(m)$ value

that is equal to or less than $g(m')$, in order to ensure that the battery energy at epoch n_{b_2} does not fall below E_{min}.

Finally, consider the epochs $m \in [n_{b_l}, k_1]$ that "bottleneck" at the last round, against the constraint in line 6 ($n = k_1$). Similar to the above cases, we can argue that if the $g(m')$ value any one of these epochs m' is increased, that must be accompanied with decreasing a $g(m)$ value that is equal to or less than $g(m')$ in order to ensure that the battery energy at epoch $n = k_1$ does not fall below $\tilde{E}(k_1)$. From the definition of the CMF solution, we can then conclude that the energy allocation vector $g^{k_1} = (g(n'), n' \in [n_0, k_1])$ computed by Algorithm 4.3 at the end of epoch k_1 is a CMF solution of $\mathbf{P}_{k_1}(\tilde{E}(k_1))$, *provided* g^{k_1} satisfies all problem constraints.

To conclude the proof of the base step, therefore, lets us confirm that all the constraints of $\mathbf{P}_{k_1}(\tilde{E}(k_1))$ are satisfied by the solution computed so far, g^{k_1}. The lower bound constraints in Eq. (4.18) are satisfied by the initialization step itself (line 3); note that Algorithm 4.3 ensures that the $g(m)$ values do not fall below these lower bounds any time during the run of the algorithm. The last run of the while loop ensures that the battery energy constraint at epoch $n = k_1$, as given by Eq. (4.17), is satisfied. The battery energy lower bounds (E_{min}) given by the left-hand inequalities in Eq. (4.16), are always satisfied by the way the g value (which the energy allocations of the non-bottlenecked epochs are incremented to) is chosen in line 16 of the algorithm. The satisfaction of the battery energy upper bounds (E_{max}) given by the left-hand inequalities in Eq. (4.16) is however less obvious, since it is not explicitly imposed by the algorithm. For the sake of contradiction, suppose that the battery energy $E(n')$ for some $n' = [n_0, k_1 - 1]$ exceeds E_{max} in g^{k_1}. Then

it is easy to argue that the problem $\mathbf{P}_{n'}(\tilde{E}(n'))$ would be feasible: if the battery energy can be made to reach a value greater than E_{max}, it could also be made to reach a value $\tilde{E}(n') \leq E_{max}$. However, since $k = k_1$ is the first epoch for which $\mathbf{P}_k(\tilde{E}(k))$ is feasible, that is no possible. This implies that the upper bound constraint on the battery energy is also satisfied for $n' = [n_0, k_1 - 1]$. This concludes proof of the base step.

Induction step: The proof of the induction step follows along similar lines as that of the base step, and we only highlight the main differences. Let the claim of the theorem hold for the ith feasible epoch, k_i. We want to show that it holds for the $(i+1)$th feasible epoch, k_{i+1} as well.

As in the base step, when $n = k_{i+1}$, the while loop may run multiple times. In all of these runs except the last one, a battery energy lower bound (E_{min}) is reached at some intermediate epoch. In the last run, the battery energy constraint at $n = k_{i+1}$, $\tilde{E}(k_{i+1})$, is met. Note that for $n = [k_i, k_{i+1} - 1]$, the problem $\mathbf{P}_n(\tilde{E}(n))$ is infeasible; therefore the $g(n)$ values do not get updated for those runs of the for loop in lines 4-25. As the for loop is run for $n = k_{i+1}$, consider the first epoch $n'_{b_1} \in [k_i, k_{i+1}]$ that bottlenecks, i.e., for which the battery energy reaches E_{min} while executing the while loop in lines 5-24. Let n'_{b_0} be the bottleneck epoch when the algorithm enters the for loop for $n = k_{i+1}$. Then, when epoch n'_{b_1} bottlenecks, the $g(m)$ values of all epochs $m = [n'_{b_0} + 1, n'_{b_1}]$ are set as equal as possible, subject to their respective lower bounds. The bottleneck epoch n_b is then updated to n'_{b_1}; therefore, the $g(m)$ values of these epochs do not change any further in the rest of the run of the algorithm. If the $g(m)$ values of any of these epochs, say m' is increased, then to maintain feasibility (i.e., prevent $E(n'_{b_1})$ from dropping below E_{min}), one of the other

$g(m)$ values (which have the same value as $g(m')$), must be decreased. This establishes the CMF property for these $g(m)$ values that get determined at this stage of the algorithm. For the epochs that get bottlenecked similarly in the other runs of the while loop (except the last one), a similar property holds, and can be shown in a way similar to that in the base step. For the last run of the while loop, the remaining epochs get bottlnecked against the constraint that the battery energy at epoch k_{i+1} needs to be $\tilde{E}(k_{i+1})$ (which is E_{max}, except when $k_{i+1} = N$, for which it is E_{min}); a similar CMF property holds for these epochs, and can be shown in the same way as we did in the base step.

The satisfaction of the constraints for the epochs whose energy allocation change in this step, i.e., epochs $n'_{b_0} + 1$ to k_{i+1}, can be shown in a manner similar to the base step. The lower bound constraints are always satisfied since the $g(m)$ values only increase over the run of the algorithm. As argued before, the algorithm ensures that the battery energy lower bound constraints (E_{min}) are always satisfied. The upper bounds on the battery energy (E_{max}) are not explicitly imposed. However, note that at the beginning of run of iteration $n = k_{i+1}$, $E(m) \leq E_{max}$ for all $m = [n'_{b_0} + 1, k_i]$, and these $g(m)$ values only increase in this iteration. Therefore, the $E(m)$ values for $m = [n'_{b_0} + 1, k_i]$ remain less than or equal to E_{max} after this as well. The argument for all the $E(n)$ values remaining less than or equal to E_{max} for epochs $m \in [k_i + 1, k_{i+1}]$ remain the same as in the base step. Therefore, given that the energy allocation vector g^{k_i} computed by Algorithm 4.3 after running the for loop for a feasible epoch $n = k_i$ is CMF for $\mathbf{P}_{k_i}(\tilde{E}(k_i))$, the $g^{k_{i+1}}$ computed at the end of the epoch $n = k_{i+1}$ is CMF for $\mathbf{P}_{k_{i+1}}(\tilde{E}(k_{i+1}))$.

Since the last epoch $n = N$ is feasible, the result in Theorem T.2 follows by

induction over the feasible epochs, which end in epoch N. ■

Algo 2: Energy Allocation between epochs (EA)

Result: $g(n)$ that maximizes $\sum_{n=n_0}^{N} V(g(n))$, s. t. (8)-(9).

Define: $\tilde{E}(n) = E_{max}$ for $n < N$, and $\tilde{E}(N) = E_{min}$;
Initialize: $n_b = n_0 - 1$, $g(n) = \max[g_{min}, \tilde{S}(n) - R_{max}]\ \forall n$;
for $n \in [n_0, N]$ **do**
  while $E(n_0 - 1) + \sum_{n'=n_0}^{N} \tilde{S}(n') - \tilde{E}(n) > \sum_{n'=n_0}^{N} g(n')$ **do**
    Find g_n satisfying:
    $E(n_0 - 1) + \sum_{n'=n_0}^{n} \tilde{S}(n') - \sum_{n'=n_0}^{n_b} g(n') - \sum_{n'=n_b+1}^{n} \max(g(n'), g_n) = \tilde{E}(n)$;
    if no solution for g_n **then** $g_n = g(n)$;
    for $m \in [n_b + 1, n - 1]$ **do**
      Find g_m satisfying:
      $E(n_0 - 1) + \sum_{n'=n_0}^{m} \tilde{S}(n') - \sum_{n'=n_0}^{n_b} g(n') - \sum_{n'=n_b+1}^{m} \max(g(n'), g_m) = E_{min}$;
      if no solution for g_m **then** $g_m = g(m)$;
    end
    $g = \min(g_n, \min_{m \in [n_b+1, n-1]} g_m)$;
    $g(n) = \max(g(n), g)$;
    for $m \in [n_b + 1, n - 1]$ **do**
      if $E(n_0 - 1) + \sum_{n'=n_0}^{m} \tilde{S}(n') - \sum_{n'=n_0}^{n_b} g(n') - \sum_{n'=n_b+1}^{m} \max(g(n'), g) \le E_{min}$ **then**
        $n_b = m$;
      $g(m) = \max(g(m), g)$;
    end
  end
end

Figure 4.3: *Energy allocation between epochs (EA).*

4.3.3 Complexity Comparison

Note that the only complex step in Algorithm 4.3 is that of finding g_m (and finding g_n, equivalently); g_m being a scalar, the value of g_m satisfying $E(n_0 + 1) + \sum_{n'=n_0}^{m} \tilde{S}(n') - \sum_{n'=n_0}^{n_b} g(n') - \sum_{n'=n_b+1}^{m} \max(g(n'), g_m) = E_{min}$ can be found in $O(N)$ time. Since this computation must be run up to N^2 times in the worst case, the EA algorithm takes $O(N^3)$

time. Once Algorithm 2 is completed and the $g(n)$ values are calculated, Algorithm 1 runs takes $O(K^2)$ time to run, for each epoch n. Therefore, the SS phase runs in $O(NK^2)$ time. Considering both the SS and EA phases, the total time complexity of Algorithm 4.3 is $O(N^3 + NK^2)$. Since the number of sensing tasks (K) will generally be small and less than the number of epochs (N), the running time is dominated by $O(N^3)$ which is cubic in the number of epochs. In general, even for MPC optimization problems with a convex structure, the solution time complexity depends on the degree of approximation desired. For example, with gradient-based algorithms, $O(1/\epsilon)$ time is typically required to obtain a degree of sub-optimality ϵ. Typically, solvers like Gurobi are needed to solve convex optimization problems quickly. However, these solvers have a large memory footprint which may not be easily accommodated on IoT devices.

Under the approximation $R(n) = R_{\max} \forall n$, Lemma L.1 shows that the constraint set can be expressed as *nested* upper and lower constraints. Separable convex optimization with such nested upper and lower constraints, which have applications in production planning, portfolio management and wireless communications, allow a specialized class of algorithms due to their structure. These algorithms, however, still have a tradeoff between the quality of solution and complexity [63]: they produce ϵ-approximate solutions in running time that depends on ϵ as $\log((NB)/\epsilon)$, where constant (resource bound) B depends on the constraint set parameters. It is worth noting that the constraints in Eq. (4.12) are of *ascending* type, in the sense described in [64]. Separable convex optimization under ascending constraints allow ϵ-approximate algorithms at complexity that depends on ϵ as $\log(B/(N\epsilon))$ [64]. However, these algorithms only work with one-sided constraints and still

have a tradeoff between degree of approximation and running-time complexity, as is typically the case for convex optimization algorithms. The development of our strictly polynomial (cubic) time algorithm exploits the nested *and* the ascending nature of the constraints in Eq. (4.12), and their connection to max-min fair resource allocation.

An alternative way to solve the MPC problem posed in Eqs. (6.3)-(4.5) is by using a dynamic programming based approach like [49]. However, these approaches will suffer from the curse of dimensionality due to the continuous nature of the state-action space, as argued earlier. The type of constraints in our problem also makes the application of dynamic programming more complex. In fact, we can consider the Energy Allocation algorithm (Algorithm 4.3) to be a dynamic programming approach where the energy conservation properties, and a well-justified approximation guided by battery charging characteristics, are used to avoid the dimensionality issue and thus solve the constrained dynamic programming problem at low complexity.

Machine learning (ML) based approaches have been used to solve similar optimization problems as well, such as the neural network (NN) based approach utilized in [52]. In that case, the time complexity depends on various factors including the type of algorithm, number of layers, neurons per layer, learning rate, etc. This makes it difficult to directly compare it with the MPC approach or PERMIT in terms of complexity. When the problem is difficult to model or its parameters are difficult to estimate, the use of NN or other ML-based approaches can be justified. However, these ML-based algorithms require extensive training over different solar profiles and thus would be difficult to deploy quickly or implement on an IoT device with limited memory and computation power. Our prob-

lem is not difficult to model and its parameters can be estimated. Thus, compared to the alternatives, PERMIT which has lower complexity and implementation requirements is a practical and efficient solution for managing IoT device energy.

4.3.4 Solar Energy Prediction

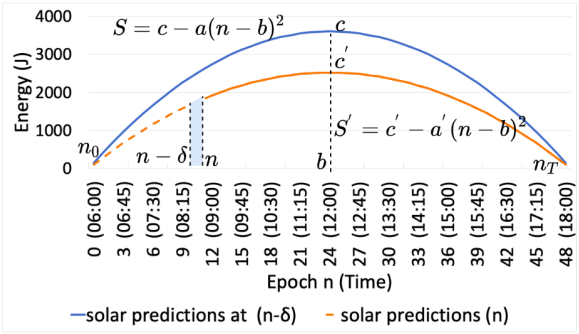

Figure 4.4: Solar energy prediction.

Due to the frequent correction by our algorithms, we only need to use the current solar prediction data from the current time till the next battery state check. Therefore our solar prediction algorithm seeks to adapt to near-term energy availability rather than primarily depending on historical data. We propose a simple Parametric Parabolic Prediction that assumes the daily solar energy follows a parabolic curve as illustrated in Fig. 4.4.

The curve is represented by $S = c - a(n-b)^2$ where S is the solar energy at a given epoch n, peak energy c is at epoch $n = b$ (e.g., mid-day) and a is an appropriately chosen constant. New solar curves are calculated every epoch with fixed start and end points, i.e., at sunrise and sunset, respectively when the solar energy is zero. Thus, the midpoint of all curves occurs at midday (i.e. $n = b$) though the values of a and c will differ between curves.

Fig. 4.4 shows the predicted solar curves S and S' at epoch $n-\delta$ and n respectively, where δ is the number of epochs over which we monitor the battery energy history and use it to estimate the amount of solar energy available during this epoch. We refer to these estimation windows (epochs) as slices e.g. if only the previous epoch's data is used in the prediction the slice $\delta = 1$. As new values for a and c are calculated, the new curve S' may be adjusted up or down. We assume the predicted battery energy at the beginning of epoch n, $E(n)$ is the sum of the battery energy at epoch $n-\delta$, $E(n-\delta)$ plus the available solar energy during the same time (i.e. $\int_{n-\delta}^{n} S\,dn$) minus the total energy used for tasks from epochs $n-\delta$ to $n-1$ i.e. $\sum_{n-\delta}^{n-1} F(n')$.

If the prediction is accurate, then the difference between $E'(n)$ the actual measured battery energy at epoch n and predicted battery energy $E(n)$, estimated at $n-\delta$ using the solar prediction data S, $(E'(n)-E(n))$ is small. Let $F(n')$ denote the total amount of energy used by the solar-powered IoT device (which we call a PERMIT Stick – see Section 4.3.5) in epoch $n' \in [n-\delta, n]$ to run tasks and power the device. Then $E'(n)$, the measured battery energy level at the beginning of epoch n, is related to the battery energy level at the beginning of epoch $(n-\delta)$ by $H(n-\delta)$ in Eq. (4.19), where $\int_{n-\delta}^{n} S'\,dn'$ is the total solar input energy from $n-\delta$ to n, $\sum_{n-\delta}^{n-1} F(n')$ is the total energy used for tasks as well as the energy consumed when idle from $n-\delta$ to n and $H(n-\delta)$ is the battery energy level at the start of epoch $(n-\delta)$. If we use a measured battery energy value at $n-\delta$ then $H(n-\delta) = E'(n-\delta)$, otherwise $H(n-\delta) = E(n-\delta)$ (predicted battery energy).

$$E'(n) = H(n-\delta) + \int_{n-\delta}^{n} S'\,dn' - \sum_{n-\delta}^{n-1} F(n'). \tag{4.19}$$

From the definition of the parabolic curve S',

$$\int_{n-\delta}^{n} S'\, dn' = c'\delta - \left(\frac{a'(n-b)^3}{3} - \frac{a'(n-\delta-b)^3}{3} \right). \tag{4.20}$$

$$c' = a'b^2; \quad a' = \frac{E'(n) - H(n-\delta) + \sum_{n-\delta}^{n-1} F(n)}{\delta b^2 - \left(\frac{(n-b)^3}{3} - \frac{(n-\delta-b)^3}{3} \right)}. \tag{4.21}$$

The curve midpoint ($n = b$) remains fixed and need not be re-estimated. Further, at $n = 0$, the solar energy is zero; therefore $c' - a'(0-b)^2 = 0$. We therefore estimate a', c' parameters of the S' curve using Eq.(4.21).

Sometimes the difference $E'(n) - H(n-\delta)$ is negative; for example, if the battery is discharging. Thus Eq. (4.21) gives negative values leading to an inverted solar prediction curve, which is not meaningful. In these cases, a' and c' are recalculated by approximating $S(n)$ with the measured solar energy during the previous epoch, $S(n-1)$ and thus calculate a' using $a' = \frac{S(n-1)}{b^2-(n-b)^2}$ and c' using Eq. (4.21).

Through experimentation and tuning, the initial values for a and c at 06:00 are set as $c = 122.4722$ and $a = 0.2126$. Note these values are only used in the first epoch before our algorithms adapt to the change in battery and re-estimate the solar curve. Some other simple solar prediction mechanisms that account for short-term weather changes include Exponential Weighted Moving Average (EWMA) [65], Weather Conditioned Moving Average (WCMA) [66], Pro energy [67] and [68]. PERMIT has a simple mechanism like [65], [66] for short-term predictions like [67] and it continually adapts to weather changes without using significant history like [66], [65] or matrix of profiles like [67].

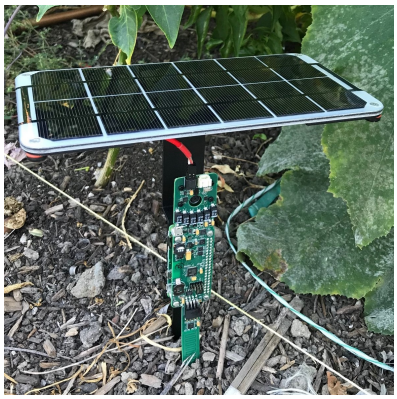

Figure 4.5: PERMIT Stick Hardware.

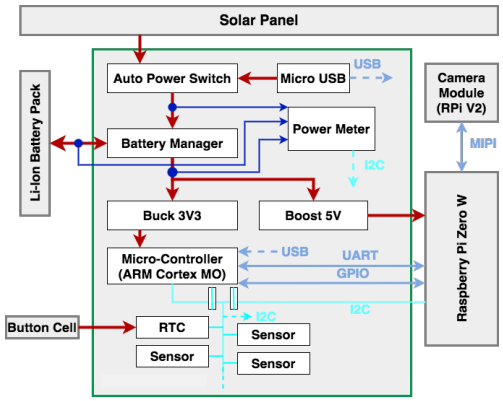

Figure 4.6: PERMIT Stick Block Diagram.

4.3.5 PERMIT Hardware Design

The design goal of the PERMIT Stick (Fig. 4.5) is to create a portable computational unit capable of monitoring and managing its energy generation and storage subsystems, while being able to modify the operation of its sensors and applications in response to measured and predicted energy fluctuations. The hardware design takes advantage of, and works alongside, a popular embedded computer - the Raspberry Pi Zero W (RPZ) [69] which has a WiFi connection supporting 802.11 b/g/n. Our design includes an energy capture component (6V 6W Adafruit solar panel [70] with 19%+ efficiency, dimensions 220mm x 175mm (8.7"x6.9")) and energy storage subsystem (3.7V 4400mAh Li-ion battery pack) which includes a mechanism to power the RPZ from either solar energy or a Li-Ion battery [71] auxiliary energy store.

The main function block of the PERMIT Stick and major external components, are in Fig. 4.6, with the main power path highlighted in red. Energy is captured at the top (via solar panel or USB power source) and an auto-sensing switch selects the appropriate

power source for the battery manager which controls the charging process in response to changes in system load and available energy. For example, if the input solar energy is too low to support the load then energy for the load is supplied from the battery. Completing the hardware features (lower portion of Fig. 4.6), is an embedded micro-controller and an array of sensors that can operate independently and monitor various environmental conditions e.g. temperature, humidity, air pressure soil moisture etc., and it also includes a camera for video and image data.

There are a couple of constraints with the hardware design. Primarily driven by cost, the PERMIT Stick omits a real-time clock and lacks true sleep operations (where the device can enter and exit low-power sleep-states on demand). Therefore we disabled the micro-controller's ability to manage the computer's power state, opting to keep the device powered on and acting as the sensor bus master at all times. Secondly, since the device is solar powered and supplies power to both the battery and the load (RPZ), the maximum energy captured at the solar panel is the sum of both the current energy required by the battery and load. Our design mitigates this limitation.

4.3.6 Task Characterization and Prioritization

Total device energy used is the sum of the base (idle) energy to keep the device operational and any additional energy for each task (characterized by its variable parameter). Energy models are determined by performing multiple experiments and fitting a suitable curve, and averaging the results.

Base Energy Estimation: Since the PERMIT Stick does not go into sleep mode and the base operating system is always running, it consumes power even when not executing

any task. The total base energy cost includes 'idle energy' (588.615J/epoch) and power measurements every minute (0.915J/epoch) and is high i.e. E_{base}=588.615J/epoch.

Video Task: The video task uploads one streaming video of duration t_v (5-30s) to the sink each epoch with a 10Mbps video bitrate, 1920×1080 pixel resolution, and 30 frames per second (fps). Total energy E_v(J) and utility U_v are given by:

$$E_v = 0.713t_v - 0.86363; \quad U_v = (1 - e^{-\frac{t_v}{5}}). \tag{4.22}$$

Image Task: This task captures and uploads one image of quality q and a fixed resolution of 2592×1944 pixels per epoch to the sink. Lower quality (q, ranges from 10-75) indicates higher image compression. Eq. (4.23) shows the utility U_i and the total energy, E_i which is the sum of computation energy (for image generation, $E_{i,comp} = 0.004255q + 0.4903$) and communication energy (for uploading data, $E_{i,comm} = s_i E_u$). E_u is the upload TCP transmission energy in Joules/MB [1] and s_i is the image size (MB).

$$E_i = (-517.9 \times 10^{-6} q^2) + (0.004255 + 0.08585E_u)q + (0.4903 - 0.1923E_u); \quad U_i = (1 - e^{-\frac{q}{10}}) \tag{4.23}$$

Appendix A includes figures illustrating the relationship between computation energy required to generate the image and image quality and the linear relationship between the video duration and measured energy consumption. A higher quality means less compression and hence a larger image size. Image size is given by $s = -517.9 \times 10^{-6} q^2 + 0.08585q - 0.1923$ and the quadratic nature of this curve is carried through to the total energy equation.

For PERMIT, we use a linear relationship between image size and quality and thus model the communication energy as a linear equation. Thus, E_i in Eq. (4.23) becomes $E_i =$

$(0.004255 + 0.04111E_u)q + (0.4903 - 0.4331E_u)$. In all cases, E_u is given by $E_u(J/MB) = 3600(1.301 \times 10^{-8}b^3 - 8.479 \times 10^{-7}b^2 + 1.573 \times 10^{-5}b + 1.8686 \times 10^{-5})$ where b is the upload bitrate. Fig. 4.13 in Section 4.4 shows a small impact of this change.

Temperature, Humidity and Soil Moisture Tasks: These tasks take environmental measurements around the device. Multiple measurements may be made per epoch and data is uploaded right after each measurement. Total energy cost per task type per epoch is the sum of the computation energy $E_{y,base}$ and communication energy ($E_u \times s_{y,base}$) times the number of measurements per epoch n_y (Eq. (4.24)). $y = t$ for temperature, $y = h$ for humidity, $y = m$ for soil moisture and $E_u = E_{u-tcp}$ and $s_{y,base}$ is the corresponding file size. From experiments, $E_{t,base}$= 0.00574J, $s_{t,base}$= 275B, $E_{h,base}$= 0.00861J and $s_{h,base}$= 55B, $E_{m,base}$=0.971J and $s_{m,base}$=908B.

$$E_y = n_y(E_{y,base} + E_u s_{y,base}); \quad U_y = 1 - e^{-n_y}. \tag{4.24}$$

For task utility, we use a simple concave utility function, ensuring the local maximum found by setting the Lagrangian derivative to zero is also the global maximum. Utility increases with the task value and flattens out beyond a certain value of each task key variable [72]. We use a utility function $U_k = 1 - e^{-\alpha_k f_k}$, where f_k is the variable task parameter, α_k is a task-dependent utility constant determined through tuning such that even the lowest value of f_k can give a reasonable utility (see Appendix A).

Task Prioritization: When an IoT device has a limited amount of energy available, it is important to use it judiciously across its various sensors, prioritizing the most valuable data for a given context, as distinct from the quality of the data provided. We use the AHP Process [73] to rate the different types of data the PERMIT Stick provides so that

'high-value' data is appropriately prioritized. A detailed explanation of the calculation is in Appendix B.

4.3.7 Battery Energy Management

We verified that the PERMIT Stick battery followed the typical characteristic Li-ion charging/discharging curves which contain regions where the battery energy depends linearly on the battery voltage (i.e., when charging, this occurs in the constant charging region). We set E_{max} (maximum battery energy) at the end of the constant-charging region, where the voltage is $v_{max} = 4.2V$ and E_{min} (minimum battery energy) at the end of the linear discharging region, where the voltage is $v_{min} = 3.0V$. We then estimate the battery energy level at any given time by combining voltage translation [74] and Coulomb counting [75]. Voltage translation uses the current voltage v_n to estimate the current battery level $E(n)$ then the Coulomb counting approach modifies the battery energy level by considering the energy going in and out of the battery, as measured by the PERMIT Stick. For the outdoor experiments, we only use voltage translation. However, we use both voltage translation and Coulomb counting for the simulations. From the linear energy voltage relationship in the constant charging region, the current battery energy level at epoch, n is given by $E(n) = E_{max}(v_n - v_{min})/(v_{max} - v_{min})$.

4.4 Evaluation

We evaluate the MPC approach and PERMIT using both simulations and live experiments. In the live experiments, the PERMIT Sticks were deployed outdoors over

several different days. We also compare the task energy management efficacy of PERMIT and Signpost which is an urban IoT monitoring solution that contains similar components to our PERMIT solution.

4.4.1 Evaluation using Simulation

Solar and battery measurements were taken with multiple PERMIT Sticks under different sunlight conditions. Using these solar profiles and battery measurements the MPC approach and PERMIT algorithms were executed every epoch to simulate their performance under the measured solar and battery energy conditions. Data was uploaded to a sink over a 10Mbps wireless Wi-Fi network.

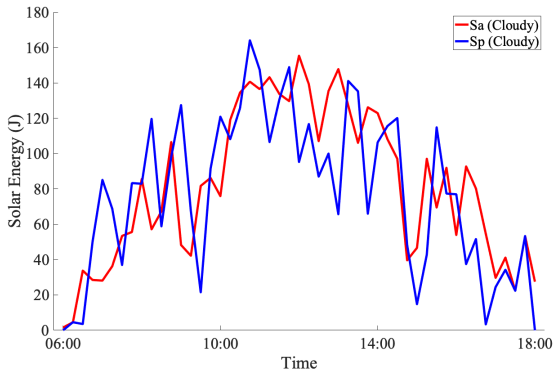

Figure 4.7: Measured and predicted solar energy.

Solar Profiles and Prediction

We identified several profiles that generally follow the typical solar energy parabolic curve, with minor differences. We present and discuss the MPC approach performance for

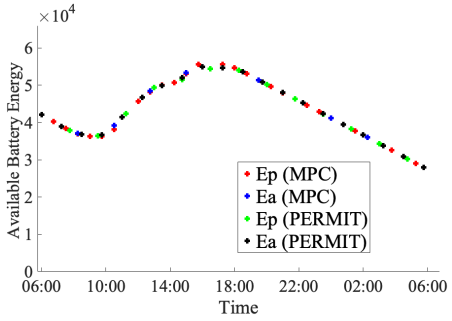

Figure 4.8: Comparing PERMIT and MPC approach on a Partly sunny day - Predicted battery

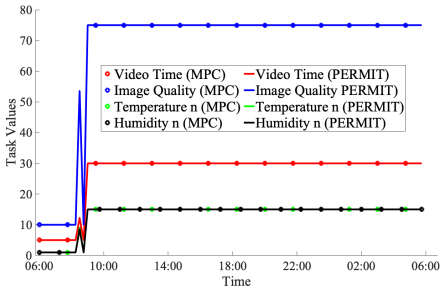

Figure 4.9: Comparing PERMIT and MPC approach on a Partly sunny day - Task parameters

two profiles: a 'Cloudy' day with low solar energy and no direct sunlight hitting the solar panel; and a 'Partly Sunny' day with high solar energy in the morning and afternoon, separated by a cloudy period with reduced solar energy. We evaluated our solar algorithm and on cloudy days comparing total predicted and measured solar energy we have an average prediction error of 16.5%. Fig. 4.7 shows measured S_a and predicted S_p solar energy on the cloudy day. Large prediction errors are not uncommon, for example, Signpost [53] reports an error of 22% on cloudy days however, since the MPC approach and PERMIT effectively use the prediction only for an epoch and continually adapt, it is not as dependent on the prediction accuracy as other approaches.

MPC approach Performance Evaluation

In this section, we use E_p as the predicted battery energy level from our MPC approach optimization (calculated from Eqs. (6.3-4.5)), and E_a as the measured battery energy from experimental measurements.

Partly Sunny Day: Figs. 4.8 and 4.9 show the battery and task results for the

partly sunny case. The solar energy during the day was enough to substantially charge the battery before and although the battery didn't get fully charged, it had sufficient energy to operate overnight.

The MPC approach and PERMIT optimize the power consumption of the PERMIT Stick by trying to use a single appropriate task value per sensor for the entire prediction horizon. The figure shows that PERMIT and MPC approach select similar task values. In this 'Partly Sunny Day' case, PERMIT and the the MPC approach lowered the task values before 09:00 because the solar prediction algorithm predicted lower solar availability for the rest of the day, based on the battery energy changes at that instant. However, from 09:00, since higher solar was actually available (thus battery energy increased), the prediction is quickly corrected allowing PERMIT and the MPC approach to select maximum task parameter values for the rest of the day and allowing the PERMIT Stick to operate overnight.

Cloudy Day: Fig. 4.10 shows the measured/predicted energy selected by PERMIT and MPC approach. The battery drained throughout the day due to low solar energy. The somewhat low starting battery level causes both algorithms to make progressive corrections modifying task values at each epoch to extend the battery life.

Fig. 4.11 shows that PERMIT selects very similar task values as the MPC approach. Task values vary more during the day (compared to the night). The available solar energy also charges the battery, resulting in revised solar predictions. Because of this adaptation, the battery energy is sufficient to keep the PERMIT Stick operating till 06:00 the next morning even for the Cloudy Day scenario, demonstrating its utility in managing

energy use to extend the lifetime of the IoT device.

Fig. 4.12 shows the task values when executing PERMIT with the measured solar data (not the predicted solar values) on the Cloudy day. We note that the task values for both the predicted and actual measured solar profiles have the same trend (compare Figs. 4.11 and 4.12). Variations in the solar prediction may overestimate the solar energy, resulting in periods where PERMIT selects higher task values than what are usable with the actual available solar energy (see graph from time 06:00-16:00). On days with low solar energy available, this may be a problem if it causes the PERMIT Stick to use more energy than appropriate (by using higher task values) resulting in an energy deficit later during the night. However, PERMIT corrects this quickly by using lower task values ensuring the battery lasts overnight.

Varying the Task Weights and Impact of Approximation Changes

Fig. 4.12 also compares the task values selected when using weights $w1 = [1, 1, 1, 1]$ and $w2 = [0.57, 2.09, 1, 1]$ for the video, image, temperature, and humidity tasks, respectively, with PERMIT. The figure shows that giving a task a higher priority ensures more energy is allocated to it. Using another cloudy day profile, Fig. 4.13 confirms that the approximation changes discussed in Section 4.3.6 do not affect the results significantly enough to justify the more complex offline MPC optimization implementation. The figure shows the task values selected using the full MPC approach (MPC) and the MPC with the approximations(Approx).

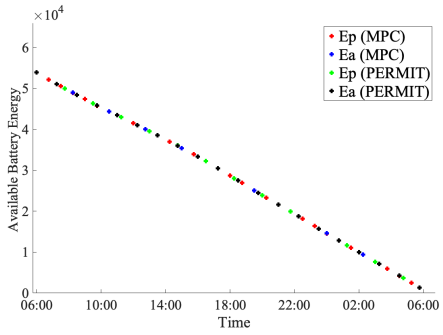

Figure 4.10: Comparing PERMIT and MPC approach on a Cloudy day - Predicted battery

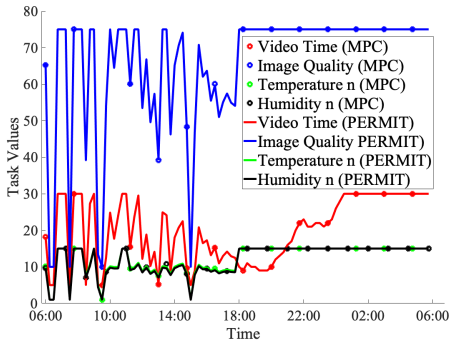

Figure 4.11: Comparing PERMIT and MPC approach on a Cloudy day - Task parameters

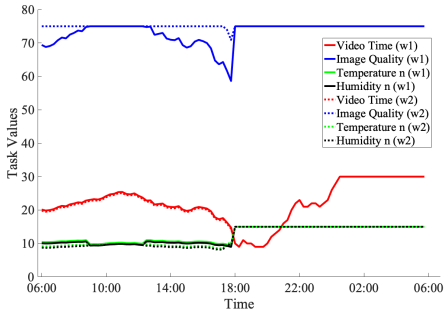

Figure 4.12: PERMIT Task values(vary weights)

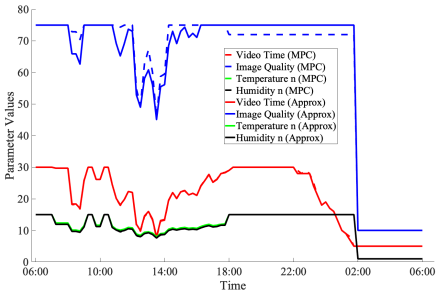

Figure 4.13: MPC approach with Cloudy profile task parameters with and without approximations.

Varying the Base Energy (E_{base})

Since we use a standard off-the-shelf compute platform (RPZ), one challenge is the high idle energy required which dominates the system's energy consumption. We vary the E_{base} (sum of idle energy and energy used to take periodic power measurements) to better illustrate how the PERMIT algorithm adapts if there is a larger fraction of energy available to the sensors (and with lower base energy).

Using the second cloudy day measured solar values and the PERMIT algorithm, we compare the task values with 95%, 100%, and 105% of the current base energy E_{base} (see

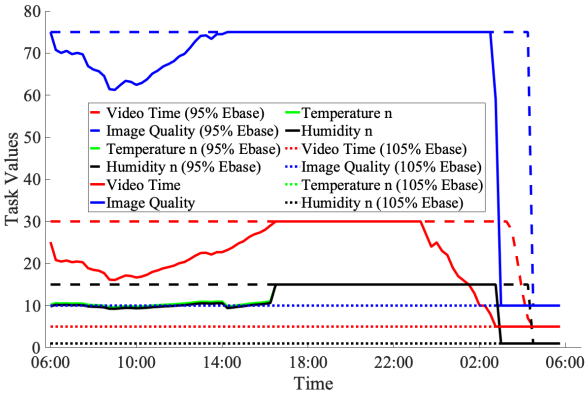

Figure 4.14: PERMIT varying Ebase.

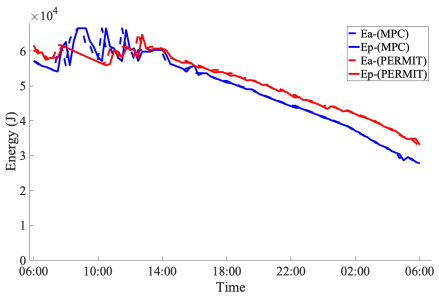

Figure 4.15: MPC approach and PERMIT predicted battery.

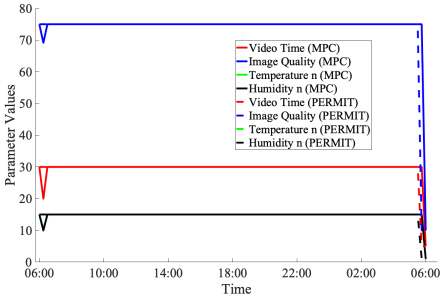

Figure 4.16: MPC approach and PERMIT task values.

Fig. 4.14). At 95% of E_{base} the extra 5% energy is now available for tasks and thus the device is able to use maximum task parameters for most of the day. At a higher, 105% of E_{base} energy consumption, the device is only able to use minimum task values throughout the day since the base energy uses an additional 5% energy. At 100% of E_{base}, PERMIT adapts for a reasonable period into the night before dropping to minimum parameters. These results show the base energy (E_{base}) has a large impact on the performance of PERMIT. However, PERMIT is still able to adapt, providing the best possible parameter values based on E_{base}, the available solar, and current battery state.

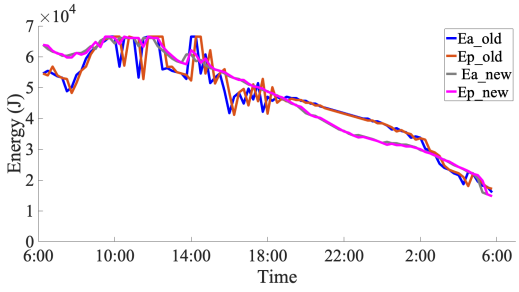

Figure 4.17: Comparing MPC approach performance with old & new batteries - Predicted battery

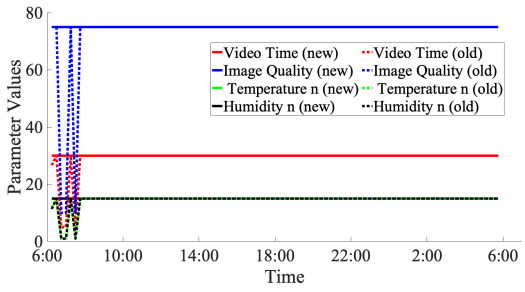

Figure 4.18: Comparing MPC approach performance with old & new batteries - Task parameters

4.4.2 Outdoor Experimental Evaluation with PERMIT Hardware

Comparing the MPC approach and PERMIT

For this evaluation, two PERMIT Sticks running the MPC approach and PERMIT were placed in an outdoor environment for a number of days over a period of several months. We show results for one sunny day with high solar energy in Fig. 4.15 which compares E_a and E_p with the MPC approach (running the exact MPC algorithm) and PERMIT (running the MPC approximation algorithm). The values for E_a and E_p are fairly close in both cases. The changes in the solar energy available in the morning are the main cause of variations in the battery levels. The battery level prediction error rate is approximately 1.66% and 1.14% for the MPC approach and PERMIT respectively. Since the initial battery energy level was high in both cases, and solar energy was also sufficient to charge the battery during the day, most of the solar energy was used for operating the sensors and supplying the PERMIT Stick idle power. The battery, therefore, lasted throughout the night and until the next morning in both cases.

The task values that were selected by both the MPC approach and PERMIT are shown in Fig. 4.16. Maximum task values were used almost all of the time due to the high

battery and solar energy. The exceptions were at 06:15 when MPC approach selected lower task values for all the tasks because the PERMIT Stick had a slightly lower battery and therefore required adaptation during that epoch. A second change is seen at 06:00 the next morning. In this case, because a new 24-hour period is beginning, the algorithms seek to optimize for the next 24-hour period. Since the current battery level is fairly low at that instant and the solar prediction algorithms also predict low solar (no solar overnight), both algorithms switch to using minimum task parameters for this epoch.

Adaptation to Aging Batteries

Our the MPC approach and PERMIT algorithms adapt to the available solar energy and battery state. However as batteries age, their performance degrades in an unpredictable manner. We compare the performance of the MPC approach with old and new batteries. The new batteries have seen limited use and the old batteries are three years old and have been used regularly for experiments.

We ran the MPC algorithm over a 24-hour period on two PERMIT Sticks (with old and new batteries) next to each other on a sunny day. Fig. 4.17 compares the predicted and measured battery levels. The prediction with newer batteries has a smoother curve. The age of the old batteries causes more voltage spikes affecting battery prediction significantly, especially when recharging. In addition, since the old battery can no longer charge fully, once the device is removed from the power supply, the voltage drops significantly (compare the old and new results at 06:00 where the new battery maintains the charge). Fig. 4.18 compares task values selected. Since this was a fairly sunny day, both could run with max. tasks for the full 24-hour period though the old battery did adapt at the beginning of the

day. These results show that the MPC approach easily adapts, even with an old battery. This is especially useful on days with little solar energy, when adaptation is needed to maximize energy use and information utility.

4.4.3 Comparing PERMIT with Signpost

Signpost [53] is a recent example of a platform for city-scale sensing. Their device is mounted on traffic signposts, provides common services such as processing, storage, power, and communications, and has pluggable slots for sensors and radios. Just like PERMIT, their components include solar prediction, energy management, and independent modular sensor components. The Signpost solar prediction model uses data from the National Renewable Energy Laboratory (NREL) MTS2 dataset [76]. Their prediction uses weekly average measurements based on geography (longitude and latitude), past history, and solar panel size and efficiency [77]. Signpost's energy allocation policy has each sensor using a virtual battery with a maximum capacity. Energy is deducted for sensing and communication. The maximum virtual battery capacity (10% of the physical battery size) is chosen as a general policy to allow applications to run while their energy reduces in a predictable manner. Harvested solar energy is allocated to the virtual batteries based on a 'leaky bucket' like allocation. Once the virtual battery for a sensor is empty that module is shut down until it has sufficient energy to restart.

Solar Prediction: For a fair comparison, we adapted the Signpost algorithm [53] to use a more fine-grained approach than the original coarse-grained model which averaged measurements over very large coverage areas and periods. We used their algorithm but averaged the 2021 NSRDB dataset values for the local weather station at our University

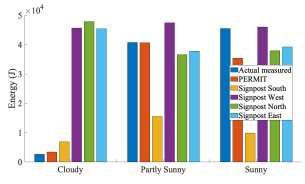

Figure 4.19: Comparing PERMIT and Signpost - Total energy

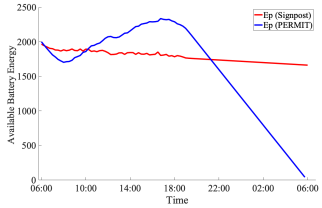

Figure 4.20: Comparing PERMIT and Signpost - Predicted battery

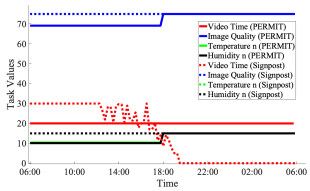

Figure 4.21: Comparing PERMIT and Signpost - Task parameters

and for our experiment days only. The PERMIT Stick solar panel size and efficiency were used, ignoring static power so we compare only the algorithms performance. Fig. 4.19 shows the total predicted harvested energy with PERMIT and Signpost panels facing the north, south, east and west over several days. For Cloudy days, PERMIT values are much closer to the measured value than Signpost because PERMIT adapts to the local device conditions like shade or obstacles obstructing the device's solar panel, thus providing unique solar predictions reflecting the environment around each PERMIT Stick. Therefore PERMIT's solar prediction is as accurate as Signpost (if not better), and serves as a good basis for the comparison of their energy management approaches. **Energy Management Comparison:** To compare the PERMIT and Signpost energy management policies via simulation, we used similar configuration parameters where possible, to reduce the impact of the different device hardware limitations. Both algorithms used the sensor tasks available in the PERMIT Stick (since we have their energy models), the PERMIT battery capacity, and energy for Wi-Fi communication. We used 15 min. epochs and Signpost's idle energy of 16mW (14.4J per epoch) to remove the dominating influence of the PERMIT Stick's high idle energy. For Signpost we used an adaptive application similar to that in [53], maximizing

task values based on virtual battery available energy. PERMIT categorizes device energy costs into 3 classes: idle energy, task communication energy, and task computation energy. Any additional energy costs, e.g., for storage services, are covered by one of these categories. Therefore using the PERMIT energy equations for both the PERMIT and Signpost algorithms allows us to have a fair performance comparison. We compared the algorithms using the Cloudy day profile's measured solar data, assuming a low starting battery state (approx. 2,000J). We used the measured solar profile data with both the PERMIT and Signpost algorithms. Based on the solar data and the initial battery state, both algorithms predict, on a per-epoch basis, the battery state changes throughout the day.

Fig. 4.21 shows that PERMIT selects the best task values based on the available energy to keep all tasks running throughout the night till 06:00 the next day and these task values can be adapted as needed. For instance with PERMIT a very small reduction in video quality around 17:45 allowed reallocation of energy to the other tasks, increasing their sensing values. In contrast, with Signpost some tasks are stopped much earlier, e.g., video task stops due to lack of energy, while other tasks have excess energy available, e.g., temperature and humidity tasks. With Signpost, because all tasks have the same 'virtual battery' size and since the video task uses the most energy, its virtual battery depletes fast. However, the harvested energy allocated to all the tasks using a leaky bucket method is insufficient to fully refill the video virtual battery since this is a cloudy day with low solar energy (see Fig. 4.21 from 13:00 where any harvested solar energy allocated to the video virtual battery is immediately used to set the highest sensing value possible with that amount of energy.) The video virtual battery empties at approximately 19:00 and the video

task shuts down. Fig. 4.20 compares the battery state at every epoch with PERMIT and Signpost. In both cases, the battery lasts up to 06:00 the next morning. But, PERMIT was able to run all tasks throughout the night (giving higher information utility) and thus has a lower battery, while Signpost's video task stops running at 19:00 and so has a higher battery charge left at 06:00, but a lower information utility.

Signpost's virtual battery 'sizing' and 'leaky bucket' allocations ignore the application's actual energy requirements. This can be inefficient and can result in energy-hungry applications being starved.

Therefore, in scenarios with limited solar and battery energy, PERMIT performs better, since it adapts the task values to available energy, maximizing information utility and also meeting the energy goals.

4.5 Related Work

IoT Device Energy Management: Signpost [53] is a recent example of a platform for city-scale sensing and has similar components to PERMIT. We compare Signpost and PERMIT in detail in Sec. 4.4.3. We focus on saving energy by adapting the IoT tasks and [78] divides such approaches into dynamic voltage and frequency scaling (DVFS) [79], duty cycling[50] and task decomposition/composition [80] approaches. [50] aims at network energy-neutral operation (i.e., the battery energy is the same at the end of a period) by adapting the device duty cycle, and recommends devices select workloads aligned with the current energy availability. In [80] energy intensive IoT tasks are decomposed into smaller sub-tasks which are executed depending on the available energy. For example a tempera-

ture sensing task is decomposed into sensing, computation and communication sub-tasks. In [49] a multiversion task approach is used with dynamic programming to schedule the best IoT task version based on energy availability but is computationally expensive. [48] uses predictive power management to match IoT device power consumption (load) and average energy generation rate. PERMIT manages available energy by adapting workloads, as suggested in [50] and implemented in [48, 49]. Compared to [49], PERMIT employs a simpler, much more cost-effective MPC-based strategy and does not require intricate circuitry like [79] for tracking voltage/frequency changes. PERMIT and [50] adjust device duty cycles but PERMIT offers a more granular control by adjusting individual task parameters in each duty cycle, as opposed to modifying only the overall duty cycle time as in [50]. Unlike [49], PERMIT does not require multiple task versions, it only modifies a single parameter per task. Also, by not decomposing tasks like [80], PERMIT reduces the risk of dealing with stale or missing data when the energy supply is insufficient to complete certain sub-tasks.

IoT Deployments: Farm Beats [51] improves farm productivity with an IoT-based platform, by integrating data from sensors and drones. They use an IoT base station which incorporates a weather aware solar prediction mechanism (performed once a day) and an energy management solution that adjusts the duty cycle of different base station components. Connectivity challenges in rural areas are addressed by using a TV White Space channel for data uploads from the IoT base station. [81] is another deployment that provides a similar solution as [51] but use LoRA instead of WiFi for the sensor-to-base station connectivity, and incorporates a sun tracker to maximize energy harvesting. CLAD [82] present a cross-layer and adaptive duty cycle protocol to reduce network latency in an agri-

cultural environment. Using a multi-hop 6LoWPAN network, each sensor's duty cycle is inversely proportional to the number of child nodes. Other examples of IoT deployments are: improving mushroom farming with Zigbee [83]; PotatoNet [84] which describes experiences of an outdoor deployment; [52] which uses a neural network approach to predict the hourly soil moisture content and determine an irrigation schedule; and a closed hydroponic system utilizing edge and cloud computing [85].

Unlike FarmBeats, PERMIT distributes solar prediction and energy management to the IoT devices and does not rely on external weather data. It also dynamically adapts to weather and environmental changes at each device throughout the day. Unlike CLAD, PERMIT does not require status updates from the sensor nodes to the sink. It is also simple enough to run on the local IoT device (a Raspberry Pi) unlike [52] which needs to be run on the more powerful compute engine at the sink. PERMIT also does not need additional hardware to track the sun, like [81].

Model Predictive Control and IoT: Model Predictive Control (MPC) [57] predicts the future behavior of a system over a finite time window by using a model for the system environment. MPC has been used for energy management in a variety of IoT-based systems e.g., optimizing indoor thermal comfort and power consumption in [55], [86] and estimating channel gains with radio frequency energy harvesting in [56]. PERMIT uses MPC to optimize power consumption similar to [55] and [86] but in an outdoor environment with no reliable (wired) energy supply, i.e., it uses solar energy harvesting. It also uses MPC for task optimization unlike [56].

4.6 Conclusion

We presented and evaluated PERMIT, a predictive energy management solution for IoT-based applications that utilize renewable energy sources and batteries. PERMIT uses an MPC-based optimization to continually adapt the values of sensing tasks in an IoT device, to maximize the utility of sensing, subject to energy availability and battery capacity limits. To overcome the onerous computation requirements of the MPC approach on small IoT devices deployed in the field, PERMIT uses a well-justified approximation of the battery charging characteristic and structures embedded in the modeling equations to solve the MPC problem at very low complexity. PERMIT relies on an equivalent decomposition of the MPC optimization problem into two sub-problems: a task-level sensor parameter selection; and an epoch-level energy allocation. We provide computationally efficient algorithms for each sub-problem and prove their correctness. The complex energy allocation problem (which represents a constrained dynamic programming question) is solved by using a novel combination of bottleneck-based max-min fair allocation and recursive dynamic programming-like procedures that overcomes the challenges of having a continuous state-space represented by the battery energy level. PERMIT is practically implementable on a Raspberry PI for online operation and has similar accuracy as the MPC approach.

Experimental results (from simulations and live outdoor experiments under different weather conditions) with both the exact MPC approach and PERMIT show they successfully adapt task values depending on the short-term available energy. Compared to Signpost, a current state-of-the-art platform, PERMIT performs better, particularly under limited solar and battery energy conditions, which is when adaptation is highly desirable.

By intelligently selecting task values during low-solar days PERMIT almost doubles the operational period of complex tasks compared with Signpost. PERMIT also adapts to the changing behavior of aging Li-ion batteries. The PERMIT framework can be utilized for outdoor monitoring applications such as smart agriculture and early wildfire warning systems, where an IoT device may be powered by a renewable power source such as a low-cost solar panel along with a relatively small battery.

Chapter 5

Energy-Efficient Distributed Task Scheduling for Multi-Sensor IoT Networks

5.1 Introduction

Many IoT devices are resource-constrained in terms of energy but still need to regularly send sensing data to the relevant IoT applications. Therefore, to maximize IoT device operating lifetime and meet IoT application sensing data requirements, energy-efficient solutions like cooperative monitoring are required. Cooperative monitoring saves energy by minimizing redundant data sent from neighboring IoT devices with overlapping coverage areas. Our strategy to implement cooperative monitoring is to develop a scheduling mechanism minimizing temporal sensing overlap. Therefore, we present our Distributed Token

and Tier-based task Scheduler (DTTS), an energy-efficient distributed task scheduler for multi-sensor IoT devices.

Multi-sensor IoT devices utilize multiple sensors to monitor different environmental phenomena in a variety of applications like agriculture, smart cities, and disaster management including forest fires and hurricanes. Analyzing multiple data types from multi-sensor IoT devices requires more complex IoT applications but improves situational awareness, helping provide targeted responses to the monitored phenomena. For example, in agriculture, multi-sensor IoT devices can measure soil moisture, temperature, and capture images to observe crop health. Precision agriculture applications use this data to provide a response that maximizes the yield e.g., by adjusting the irrigation cycle.

Deploying IoT devices over large areas poses several challenges. First, most IoT devices are resource-constrained with limited memory, energy, and computing power. They typically rely on batteries and renewable energy sources, since brown power is not always accessible or cost-effective to provide. Therefore, careful energy management strategies are needed to ensure the IoT devices operate as long as possible (preferably always) while providing the data required by the IoT applications [45]. Secondly, the IoT device deployment pattern may have overlapping coverage areas. Therefore, it is important to avoid data redundancy when IoT devices monitoring the same location simultaneously transmit their sensor data (i.e. temporal sensing overlap) and thus avoid wasting precious energy. It can be very worthwhile to use cooperative monitoring when deploying many IoT devices. Cooperative monitoring reduces duplication of sensing tasks between neighboring IoT devices and helps better utilize the total available energy across all the IoT devices, by minimizing

temporal sensing overlap.

Cooperative sensing for energy management is used in [87] which presents a distributed multi-sensor cooperative scheduling model for target tracking based on the partially observable Markov decision process. Another example is [88], which presents an IoT network cooperative power minimization scheme. Nodes receive task requests with estimated task execution times and schedule the tasks by scaling the CPU core's operating frequency ensuring task completion within the estimated time.

Implementing cooperative monitoring is a scheduling problem. A centralized scheduler may not be desirable in an IoT environment, from the point of view of resiliency and the potential need to frequently communicate every IoT device's state information to the central scheduler. However, setting up an energy-efficient distributed scheduler is challenging. First, IoT devices are resource-constrained and therefore can only store and process a limited amount of their neighbors' scheduling/state information. Next, since IoT devices are usually energy-constrained they typically limit communication to conserve energy. Therefore, schedulers requiring significant inter-device communication to share state may not be energy efficient. Lastly, as IoT devices go to sleep or become inactive, the network topology changes must be communicated to an IoT device, thus consuming energy. These challenges make designing a distributed task scheduling algorithm more complex than a centralized scheme like Earliest Deadline First, where all necessary information (deadlines) of all nodes is known in advance.

Our main contribution in this work is our Distributed Token and Tier-based task Scheduler (DTTS), a simple energy-efficient distributed scheduler for an IoT network. Our

DTTS scheduler works with multi-sensor or single-sensor IoT devices and we refer to the monitoring period (duty cycle) as an epoch. Also, each IoT device sensor has a *start deadline*. This is the latest time (from the start of the epoch) that the sensor must begin its sensing activity (task) in order to have all measurements completed within the epoch. To minimize temporal sensing overlap, our algorithm divides each epoch into a set of non-overlapping intervals called tiers. Then, in a distributed manner and using tokens to share minimal information between the IoT devices, our DTTS algorithm schedules tasks with earlier start deadlines in the earlier tiers and tasks with later start deadlines in later tiers. Comparing DTTS against a simple periodic scheduler shows that DTTS always schedules each task before its start deadline expires.

Some examples of distributed task schedulers are in [87, 88, 49, 89, 90]. In [49], the authors use an energy neutrality constraint and dynamic programming to find a task schedule that maximizes the Quality of Service. The Lazy Scheduling Algorithm (LSA)[89] determines whether all task deadlines can be met before creating a schedule and uses task energy requirements, task deadlines, and current battery capacity of rechargeable IoT devices to make a scheduling decision. Jarvsis [90] is a distributed task scheduler that uses a hierarchy of control tasks operating in the Cloud/Fog to control robots and IoT devices on the ground.

The DTTS algorithm is based on [4] and is simple to execute unlike the dynamic programming approach in [49] or Markov decision process in [87]. DTTS does not require all task deadlines in advance like [89]. In DTTS each IoT device independently schedules its task execution. It is different from [90], where nodes higher in the hierarchy control task

execution of IoT nodes lower in the hierarchy. In [88], task execution is triggered by requests from another node. While both DTTS and [88] are distributed and consider deadlines when scheduling, *Local* [88] requires each IoT device to also keep track of neighbor's deadlines. DTTS instead uses tokens for inter-device communication of minimal information, with independent decision-making at each IoT device.

5.2 System Design

5.2.1 Design Concepts

A task is when an IoT device performs a sensing activity using a particular sensor. For example, an IoT device with a camera and temperature sensors has image, video, and temperature tasks. A task can be executed multiple times per epoch. This number is referred to as the task parameter value and can change every epoch. For example, if the temperature task parameter value is 5, the temperature task is executed 5 times per epoch.

Assuming task executions at a device are uniformly distributed in the epoch, the time interval between two consecutive task executions is the task *period*. Therefore, there is a time limit (from the start of the epoch) to start the first task execution so the last task execution can still be completed within the epoch. This time limit is the task's *start deadline* (i.e. task maximum start time) and its value is determined by subtracting the task *execution time* from the *period*.

Our algorithm sets the task *start time*, i.e., when the IoT device first executes the task in the epoch, to any value from 0 up to the task *start deadline*. Once the IoT device knows its *start time*, it automatically executes the task every *period* during the epoch until

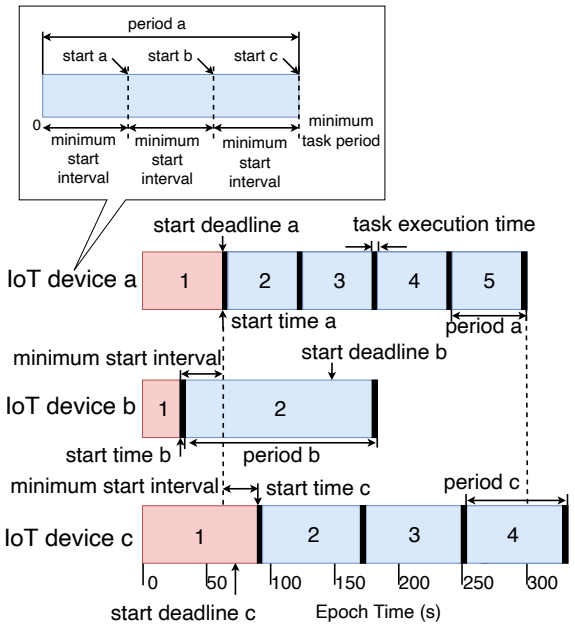

Figure 5.1: Figure illustrating design concepts.

it reaches the required task parameter value (number of executions). For each task, we also record the *minimum* and *maximum task periods* within the IoT network.

Lastly, to minimize the temporal overlap we set a *minimum start interval* between the *start time* of the same task on any two IoT devices e.g. if the *minimum start interval* is 15s and the *start time* for an IoT temperature task is 60s, then the *start time* for the temperature task at the next IoT device must be 75s or later. Fig. 5.1 illustrates these key concepts using a temperature task at three IoT devices a, b, and c. The task execution time is the same in all cases (shown by the thick black vertical lines) and the epoch duration is five minutes. The task parameter values for a, b, and c are 5, 2, and 4 giving periods 60s, 150s, and 75s, respectively. The task *start time* is shown by the width of the red rectangle and takes any value from 0 to *start deadline*. Note that IoT devices with different *periods*

have different *start deadlines*. In each case, the interval between consecutive task executions at a device is the task *period* (shown by the width of the blue rectangles). The *minimum task period* is *period a* and the *maximum task period* is *period b*.

With *a*, the *start time* is equal to the *start deadline* and the task is executed five times within the epoch. With *b*, the *start time* is less than the *start deadline* so the last time the task is executed is much earlier in the epoch. With *c*, the *start time* exceeds the *start deadline* so when the task is executed for the last time, it is not within the epoch which is unacceptable.

The call-out above *a* in the figure shows how the *minimum start interval* is determined. Since the *minimum task period* is 60s we assume the temperature task on all IoT devices must be scheduled from 0-60s. Evenly distributing the task *start times* from all 3 IoT devices within 60s means the interval between each *start time* is 20s i.e. the *minimum start interval*. Therefore, the interval between the task *start time* at two different IoT devices must be at least the *minimum start interval* (see start times at *a*, *b*, and *c*).

5.2.2 IoT Monitoring Environment

Our design is considered in the context of an IoT monitoring solution we developed, and the important characteristics in terms of tasks and monitoring environment requirements.

Predictive Energy Management for IoT

One key aspect required for our DTTS scheduling algorithm is knowing how many times a task must be executed per epoch i.e. the task parameter value. DTTS relies on

existing algorithms to determine these values per epoch at the IoT device and one example algorithm is PERMIT.

We use PERMIT to provide the task parameter values for several reasons. First, it is a solution we have built that includes a practical low-cost and multi-sensor IoT device, therefore we can perform on-device experiments to test the scheduler's performance. Next, PERMIT includes two task adaptation algorithms of which one is distributed and can be executed quickly on the IoT device itself. Therefore, together with our DTTS scheduler, this provides a distributed energy management solution. Finally, it adapts the sensor task parameter values every epoch based on the available energy and thus responds quickly to energy changes in the device/environment.

PERMIT runs on the PERMIT Stick hardware unit which is a prototype IoT device. The key hardware components include a solar panel, lithium-ion battery, a Raspberry Pi Zero W with a camera, and an embedded microcontroller with multiple sensors. The PERMIT Stick can run 5 tasks: temperature, humidity, soil moisture, image, and video, and the energy cost for each task was modeled based on a single variable parameter e.g. the temperature task uses number of task executions per epoch.

For each PERMIT Stick, the PERMIT algorithm uses a heuristic optimization, total available energy (from the battery and solar source), and system models to determine the appropriate parameter values for each task every epoch. PERMIT maximizes the information utility but also ensures that there is sufficient battery energy for the PERMIT Stick to operate overnight till the battery can be recharged the next day. In deployments over large areas with multiple PERMIT Sticks, the devices may experience different local

conditions that affect the amount of solar energy their solar panel receives, for example, there may be obstructions like clouds, trees, or buildings. The PERMIT algorithm caters for such scenarios and therefore in the same epoch, may generate different task parameter values for each PERMIT Stick. In this work, we focus on the independent temperature task which has a fixed execution time, and can be executed independently of any other task on the same PERMIT Stick. We also assume PERMIT determines the task parameter values.

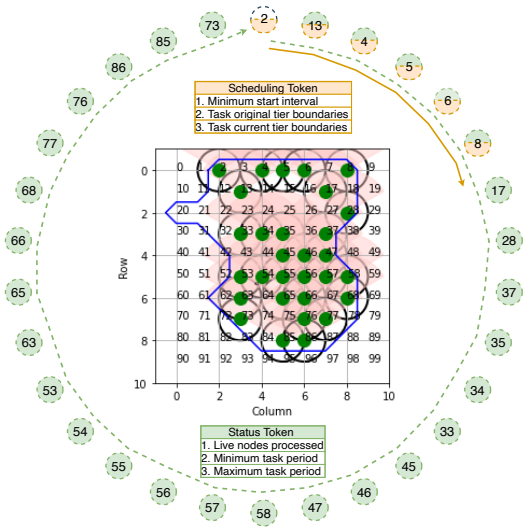

Figure 5.2: Example deployment (physical and logical network).

Design of a Multi-Sensor IoT Monitoring Environment

Leader Node We assume the network has a leader IoT device which may change between epochs and is determined using existing leader selection algorithms.

Inter-device Communication We assume that IoT devices communicate with each other using a circular Distributed Hash Table (DHT)-like network whereby all IoT

devices know the address of their nearest live neighbor [91]. Fig. 5.2 shows an example deployment with 30 randomly placed IoT devices in a deployment area covered by numbered grid points. The central inner figure represents the physical network and the sensing coverage areas for the temperature task (circular) and the video/image tasks (conical) are illustrated by the blue circles and red sectors respectively. The outer circle of nodes represents the logical network where the leader node (2) is white and we assume the IoT device ID is the grid point number where it is located. In this figure, the tokens travel in a clockwise direction and we show the contents of the Status and Scheduling tokens. The figure shows the protocol progress, with the Status token having been processed completely by all the nodes (shown by the dashed node borders). The Scheduling token has been processed by the first 6 nodes (shown by the orange color) and is sent to the next node (17).

PERMIT currently uses WiFi for communication. However, since we are parsimonious in using the communication for only uploading sensor results and strive to minimize the communication between devices, we also explore the use of LoRaWAN (Long Range WAN), a Low Power Wide Area Network (LPWAN) technology. LoRa consumes less power [92] compared to WiFi, albeit having lower bandwidth (250 Kbps vs. 54 Mbps for 802.11 b/g), but can be suitable for the DTTS system e.g. in [93] for data transmission at 10Kbps over 50 meters distance between transmitter and receiver, WiFi uses 100mW while LoRaWAN uses 20mW. Another advantage for LoRAWAN is its longer range (few Kms for LoRA compared to a few 100 meters for WiFi), which permits larger-scale deployments.

Tokens With DTTS, each IoT device does not need to know the sensing schedule of other IoT devices. DTTS strives to transmit a minimal amount of data between IoT

devices, allowing each IoT device to schedule its tasks in a completely distributed manner. However, generating the data to be shared requires gathering some information across the entire IoT network. An efficient way to first gather this information and then share minimal data between IoT devices is through tokens. Token passing is a well-understood technology (e.g., IEEE 802.5 [94], FDDI [95]). Challenges like handling lost/duplicate tokens are easily resolved using existing techniques where leader nodes handle token generation and initiate token recovery after failure by using timers [96, 97].

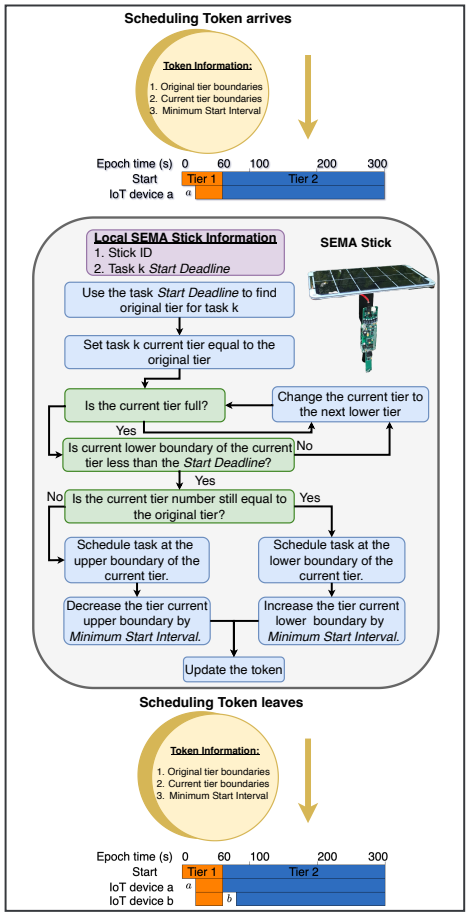

Figure 5.3: DTTS algorithm for scheduling round

5.3 Scheduler Design

Given the task parameter value for an individual sensor on a particular IoT device, the key design goal of DTTS's scheduler is to distribute the task execution across the epoch and to minimize the temporal overlap from execution of the same sensing task across multiple IoT devices in the vicinity of each other.

We ensure that on each IoT device, the first time the task is executed per epoch i.e. the task *start time*, is before the *start deadline*. This is accomplished by using tiers in DTTS. Our algorithm divides the epoch time into a given number of non-overlapping intervals called tiers. Tasks are then scheduled within tiers based on their *start deadline*. This ensures that tasks with earlier start deadlines are scheduled before tasks with later deadlines. The number of tiers required is provided in advance and is at least one. Each tier has a lower and upper boundary and the first tier (Tier 1) is always from 0-*minimum task period*. The remaining time interval from *minimum task period* to *maximum task period* is then divided equally into the remaining tiers. In this work, we use only two tiers so Tier 2 runs from *minimum task period* to *maximum task period*.

Our algorithm also minimizes inter-IoT device communication costs by piggy-backing minimal meta-data in tokens that are passed around between the IoT devices arranged in a logical ring.

In terms of operation, the DTTS algorithm relies on two rounds of token passing each epoch traversing through the network of IoT devices to schedule the start time for each sensing task on each IoT device. The leader IoT node first generates a *status token* that traverses the network and gathers key information like the *minimum and maximum*

task periods from all the IoT devices before returning to the leader node. The leader node first processes the information to determine key parameters such as the tier boundaries. It then generates a *scheduling token* which traverses the network and sets the task *start time* for all the tasks at each IoT device it passes through.

5.3.1 Status Round

The first status round token is generated by the leader node to carry three key pieces of critical information that are updated as the token travels between IoT devices. The first is the *number of live nodes* that have processed the token so far in the Status round and the next two are the *minimum task period* and *maximum task period* for each task seen so far. When the token returns to the leader node, its information is used to determine the tier boundaries and the *minimum start interval*.

The tier boundaries are determined as described earlier by using the *minimum task period*, *maximum task period*, and the specified number of tiers. Given that we have an IoT device in the network reporting a *minimum task period*, we need to set the *minimum start interval* to be small enough that all the devices start their tasks within this *minimum task period*. This guarantees that no (*start deadline*) will be violated. The *minimum start interval* is calculated by dividing the *minimum task period* by the total number of active IoT devices (see Fig. 5.1). Once the *minimum start interval* and the tier boundaries have been determined, the scheduling token can then be generated.

5.3.2 Scheduling Round

The key steps of the scheduling round algorithm for independent tasks are shown in Fig. 5.3. The leader node generates a scheduling token, that carries three key pieces of information per task: the *minimum start interval*, the *original tier boundaries*, and the *current tier boundaries*.

When the scheduling token arrives at an IoT device, DTTS checks the device's task *start deadline* and by looking at the task's *original tier boundaries* it determines which tier this *start deadline* is in.

If the tier is full i.e., no available slots for scheduling tasks, then DTTS finds the next lowest tier that is not full. When it finds a tier that is not full, DTTS checks whether the *current lower boundary* of the tier is lower than the device's *start deadline*. If so, then we set the task *start time* to the *current lower boundary* value because this means the task can be scheduled at this time without violating its *start deadline*. DTTS then updates the tier's *current lower boundary* value in the token, increasing it by the *minimum start interval*. If however, the tier's *current lower boundary* is higher than the IoT device's *start deadline* then the task cannot be scheduled in that tier without violating the *start deadline*. In this case, DTTS moves to the next lower tier and schedules the task at the *upper boundary* of this lower tier. DTTS then updates the tier's *current upper boundary* value in the token, decreasing it by *minimum start interval*.

DTTS repeats this scheduling process for all device tasks before forwarding the token to the next IoT device. By scheduling tasks at tier boundaries and updating the current boundaries in the token, DTTS schedules tasks in an entirely distributed manner

IoT ID	Task Parameter Values	Period (s)	Start Deadline (s)
a	5	60	59
b	2	150	149
c	1	300	299
d	4	75	74

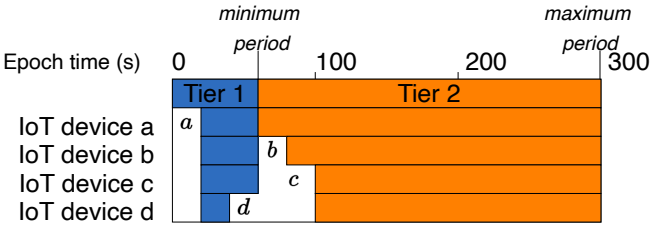

Figure 5.4: Temperature task scheduling example.

with minimal information being communicated between IoT devices. More information is in [4]. Given T tiers and K tasks, DTTS complexity is $O(TK)$.

5.4 Example

We now describe the operation of DTTS using an example with four IoT devices. Additionally, the evaluation in Section 5.5 is with 30 devices. Consider a network with four IoT devices a, b, c, and d running a temperature task with a 1 second task execution time. The table in Fig. 5.4 shows the task parameter values, *periods*, and *start deadlines* for a 5-minute epoch. For example, the task parameter value (number of measurements per epoch) for a is 5 and therefore has a 60s *period* with a *start deadline* of 59s. By comparing all the periods, the *minimum task period* is 60s, the *maximum task period* is 300s, and since there are four devices, the *minimum task period* is 60s. If we have 2 tiers, then Tier 1 runs from 0-60s, and Tier 2 runs from 60-300s.

Fig. 5.4 illustrates how the tasks are scheduled on the four IoT devices. The blue and orange areas indicated slots in Tier 1 and Tier 2, respectively, that are available for scheduling while the white slots already have tasks scheduled. As the token travels to each

Iot device, DTTS schedules the temperature task at each additional device and the tier boundaries are moved progressively inwards.

If a is the leader node it is the first to be scheduled. Since its *start deadline* is 59s, we see that from the original tier boundaries this is in Tier 1 which is not full. The *current lower boundary* of Tier 1 is 0 (less than 59s), so we schedule the task at the beginning of the epoch (i.e. at 0 seconds). We then move the Tier 1 *current lower boundary* value up by 15s (the *minimum start interval*) in the token before forwarding it to b.

At b, its *start deadline* is 149s which is in Tier 2 (from checking the task *original boundaries*). The *current lower boundary* of Tier 2 is 60s (less than 149s) so we can schedule the task at 60s without violating the *start time* deadline. The Tier 2 *current lower boundary* is increased by 15s to 75s in the token, before forwarding it to c.

At c, its *start deadline* is 299s which is in Tier 2 (from the *original boundaries*). Similar to the case with b, we can schedule this task at the *current lower boundary* of Tier 2. The *current lower boundary* of Tier 2 is updated from 75s to 90s in the token before it is forwarded to d.

At d, its *start deadline* is 74s which according to the *original tier boundaries* is in Tier 2. However, the *current lower boundary* of Tier 2 is 90s, which is beyond this task *start deadline*. Therefore, in this case, we fall back to the next lower tier (Tier 1) and schedule the task at the *current upper boundary* of Tier 1. The *current upper boundary* of Tier 1 is then reduced by 15s to 45s. Since d is the last IoT device the token returns to the leader.

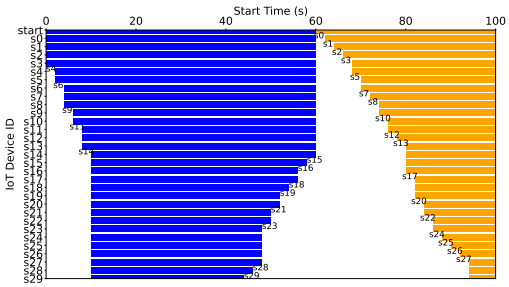

Figure 5.5: DTTS temperature task schedule for a sunny day

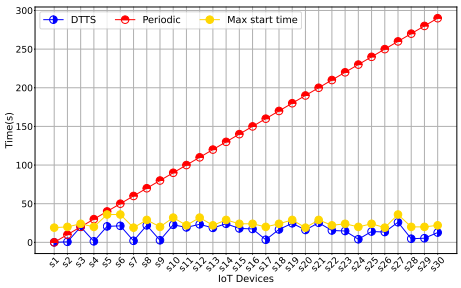

Figure 5.6: Comparing start deadline and start time with DTTS and the periodic schedulers

5.5 Results

The results from experiments done via simulations show that our DTTS algorithm schedules tasks with earlier *start deadlines* in earlier tiers and tasks with later *start deadlines* in later tiers. The results also show that DTTS always schedules the task *start time*, before the *start deadline* expires.

Using 5 and 15-minute epochs we evaluated the performance of our algorithm using simulations and 30 IoT devices. We assume the tokens travel between IoT devices based on increasing device IDs. Here we show the results for the temperature task only, while in [4] we considered tasks like image and video with different parameters e.g. duration, not just the number of executions per epoch. We used 3 different types of epochs to represent cloudy, sunny, and hybrid days. Based on the SEMA work, the maximum task parameter value for a temperature sensing task during a 5 min epoch is 5. Therefore on a sunny day, task parameter values range from 3-5 due to sufficient energy to charge the IoT device and execute more sensing activities. On a cloudy day, task parameter values range from 1-2 since the IoT device is conserving energy due to low solar to recharge the battery. On a

hybrid day, task parameter values range from 1-5 since the solar energy varies.

For each day type, we generated the task parameter values for all the IoT devices and then ran the DTTS scheduler to schedule all tasks. This was done multiple times and the results from the 5-minute epoch are presented in Fig. 5.5 for one sample schedule on a sunny day.

On a sunny day, higher task parameter values correspond to a shorter task *period* and smaller tiers for the temperature sensing task. From Fig. 5.5, we see Tier 1 is from 0-60s and Tier 2 is from 60-100s. Task *start times* are set to earlier in the epoch, i.e., all task *start times* are set within the first 100s even though the epoch duration is 300s.

On a cloudy day, however, lower task parameter values correspond to a longer task *period* and larger tiers so tasks are spread throughout the entire epoch.

We also compare our DTTS scheduler with a simple round-robin periodic scheduler which calculates its *minimum start interval* by dividing the total epoch time by the number of live IoT devices. All the IoT devices are then scheduled one after another with the *start times* separated by an interval of *minimum start interval* seconds.

Fig. 5.6 compares the scheduled *start times* and *start deadlines* between DTTS and the periodic scheduler at the IoT devices during a sunny epoch. The results show that DTTS always sets the *start time* before the *start deadline* expires while the periodic scheduler frequently exceeds that deadline.

If multiple devices in a deployment area simultaneously send their data to the sink (due to independent scheduling), there is a high risk of congestion loss at the sink, resulting in retransmissions by IoT devices, potentially wasting energy. In addition, if

multiple devices have overlapping coverage areas, simultaneous sensing results in duplicate data, wasting device energy and unnecessary processing upstream for deduplication. DTTS mitigates this by coordinating the scheduling across all devices, reducing redundant data, reducing simultaneous transmission, and minimizing the need for retransmissions. DTTS is thus more energy efficient compared with independent scheduling.

5.6 Conclusion

Cooperative monitoring among multiple IoT devices helps manage their energy consumption while monitoring large physical areas. Our Distributed Token and Tier-based task Scheduler scheduling protocol (DTTS) presented here is an energy-efficient distributed scheduler suitable for an IoT network. Using a simple protocol with minimal information sharing between IoT devices, DTTS works with multi-sensor IoT devices utilizing *start deadlines* to distribute task *start times* every epoch to minimize temporal overlap. Experiments show that DTTS always schedules the task start time of the IoT device before the start deadline expires.

Chapter 6

Task Adaptation and Intelligent Scheduling in IoT Networks using a Cooperative Sensing Approach

6.1 Introduction

Multi-sensor IoT devices can be used to monitor different environmental phenomena for example an agricultural IoT device can have different sensors for monitoring temperature, soil moisture, and humidity. However, these IoT devices depend on batteries and renewable energy sources such as solar energy for power and need to operate for long periods of time in remote areas. Therefore, efficient energy management solutions are needed to maximize the IoT device lifetime while ensuring appropriate and sufficient data is captured to be processed by the IoT application.

If several neighboring IoT devices have overlapping geographical coverage areas, there is potential for temporal sensing overlap and wasted energy due to data redundancy, when the neighboring IoT devices execute their sensing tasks simultaneously. One way to address this challenge is by using cooperative sensing. Cooperative sensing allows multiple IoT devices to collaborate and coordinate their sensing operations to attain the required data while minimizing energy use. Examples of solutions that use cooperative sensing include [87] which presents a distributed multi-sensor cooperative scheduling model for target tracking and [88] which presents a cooperative power minimization scheme for IoT networks.

Implementing cooperative monitoring is a scheduling problem and although a centralized scheduler can be used to determine the schedule, this may not be desirable in an IoT environment. A centralized scheduler may result in high communication costs for resource-constrained devices which may need to send regular state updates to the centralized scheduler. In addition, in an IoT network, there may be additional communication challenges due to poor network connectivity. A lightweight distributed scheduler can be used to address this challenge and two examples are in [90] and [5]. Jarvis [90], uses a hierarchy of control tasks operating in the Cloud/Fog while the DTTS scheduler [5] uses a token-based approach and schedules tasks before their start deadlines however it only considers the temporal overlap of the start time of the first task execution. The authors in [49] present a task scheduler that uses dynamic programming but can be executed on an IoT device with a small overhead. We leverage cooperative sensing to increase energy efficiency and minimize temporal overlap in IoT networks. We consider the temporal overlap of all task executions, unlike [5] and we do not require a hierarchy of task executions like [90].

We leverage cooperative sensing to schedule the sensing operations at each IoT device to minimize temporal overlap and also to adapt the IoT device sensing operations based on information from its neighbors.

We present our distributed scheduler to manage sensing across multiple IoT devices. Our distributed "Block Scheduler" minimizes overlap across all executions of a sensing task among neighboring devices by reframing the scheduling problem as a block placement problem. Our approach shares minimal information between neighboring IoT devices by use of tokens. As the token travels between IoT devices our Block Scheduler obtains the best schedule by minimizing total block overlap every epoch.

Our Distributed Task Adaptation (DTA) algorithm enables devices to adjust their sensing tasks by leveraging information from neighboring devices. This is achieved by dividing the monitored area into a grid and setting a coverage threshold, that determines the minimum number of measurements per grid point. Using two token rounds, each IoT device uses information from its neighbors to fine-tune its sensing task parameters. In the first round, the devices assess their current grid point coverage and adjust their task parameters to meet the threshold and in the second round, the algorithm then refines the selected task values to rectify any over- or under-covered grid points.

Through simulations, we have assessed the performance of our DTA algorithm, revealing that in small networks operating under both sunny and cloudy weather conditions, it achieved an average energy savings of 5% per IoT device. Notably, for some devices with overlapping coverage areas, we achieved remarkable energy savings exceeding 30%. In larger IoT networks, our DTA algorithm exhibited an average energy savings of 3.34%

on cloudy days and up to 38.53% on sunny days. These findings underscore its effectiveness across diverse environmental conditions. Furthermore, in various scenarios, our Block scheduler consistently demonstrated performance closely aligned with the optimal solution, showcasing its capability to efficiently reduce temporal overlap during scheduling.

6.2 Background and Related Work

6.2.1 Related Work

Cooperative sensing for energy management is used in [87] which presents a distributed multi-sensor cooperative scheduling model for target tracking based on the partially observable Markov decision process. Another example is [88], which presents an IoT network cooperative power minimization scheme. Nodes receive task requests with estimated task execution times and schedule the tasks by scaling the CPU core's operating frequency ensuring task completion within the estimated time.

Our Distributed Task Adaptation algorithm does not require additional hardware like [88] for frequency scaling and is less complex than [87]. It utilizes tokens to share minimal information between neighbors and can adapt predetermined task values based on neighbors information shared using tokens.

Implementing cooperative monitoring is a scheduling problem. A centralized scheduler may not be desirable in an IoT environment, from the point of view of resiliency and the potential need to frequently communicate every IoT device's state information to the central scheduler. However, setting up an energy-efficient distributed scheduler is challenging. First, IoT devices are resource-constrained and therefore can only store and

process a limited amount of their neighbors' scheduling/state information. Next, since IoT devices are usually energy-constrained they typically limit communication to conserve energy. Therefore, schedulers requiring significant inter-device communication to share state may not be energy efficient. Lastly, as IoT devices go to sleep or become inactive, the network topology changes must be communicated to an IoT device, thus consuming energy. These challenges make designing a distributed task scheduling algorithm more complex than a centralized scheme like Earliest Deadline First, where all necessary information (deadlines) of all nodes is known in advance.

Some examples of IoT task schedulers are in [87, 88, 49, 89, 90, 4, 5]. In [49], the authors use an energy neutrality constraint and dynamic programming to find a task schedule that maximizes the Quality of Service. Their approach uses tasks with different qualities of service and they select the task versions with the highest quality of service based on the available energy. The Lazy Scheduling Algorithm (LSA)[89] determines whether all task deadlines can be met before creating a schedule and uses task energy requirements, task deadlines, and the current battery capacity of rechargeable IoT devices to make a scheduling decision. Jarvsis [90] is a distributed task scheduler that uses a hierarchy of control tasks operating in the Cloud/Fog to control robots and IoT devices on the ground. In [98] the authors present an energy-aware task scheduler for batteryless devices that uses a Mixed Integer Linear Programming (MILP) framework. Their algorithm decomposes the application task into smaller subtasks and based on the available energy determines which subtasks to be executed. In case of a power failure due to low energy, the runtime keeps track of the active task and can re-execute it when the device turns on again.

Another example of a distributed IoT task scheduler is DTTS (Distributed Token and Tier-based task Scheduler) [4], [5]. This algorithm divides the monitoring time (epoch) into a set of non-overlapping intervals called tiers. Then in a distributed manner and using tokens to share minimal information between the IoT devices, their algorithm schedules tasks with earlier start deadlines in the earlier tiers and tasks with later start deadlines in later tiers. While this solution is lightweight and distributed, it only focuses on scheduling the start time of the first task execution ignoring potential overlap during later task executions in the epoch.

The Block Scheduler algorithm builds on [4] and [5] and can run on the IoT device itself unlike [98]. Our scheduler is simple to execute unlike the dynamic programming approach in [49], the MILP approach in [98], or Markov decision process in [87]. The Block Scheduler does not require all task deadlines in advance like [89]. The Block Scheduler can also operate with multiple tasks that have different service levels unlike [89] which assumes a single task with varying service levels. In the Block Scheduler, each IoT device independently schedules its task execution. It is different from [90], where nodes higher in the hierarchy control task execution of IoT nodes lower in the hierarchy. In [88], task execution is triggered by requests from another node. While both the Block Scheduler and [88] are distributed and consider deadlines when scheduling, *Local* [88] requires each IoT device to also keep track of neighbor's deadlines. The Block Scheduler instead uses tokens for inter-device communication of minimal information, with independent decision-making at each IoT device. The Block Scheduler also addresses the overlap of all task executions within an epoch unlike [5] which only focuses on the overlap of the first task execution.

6.3 System Design

6.3.1 System Assumptions

There are a few assumptions that we make for our design. For our scheduler operation, we assume that the task parameter values for each task are available. In this work we assume these task values (f_k) are provided by existing algorithms, such as Signpost [53] or PERMIT [3]. We focus on three tasks in this work (temperature, humidity, and image tasks) and assume that the task values provided are the frequency of execution per epoch for each task i.e. n_i, n_t, and n_h for the image, temperature and humidity tasks respectively. Note that although the initial PERMIT solution in [3] used image quality as the variable parameter for the image task, modifying PERMIT to use image execution frequency as the variable parameter with a fixed quality provides similar results. In this case, PERMIT adapts the frequency of image captures based on the available device energy, and the number of measurements per epoch ranges from 1 to 30.

We also assume that the tasks are independent i.e., multiple tasks can be executed at the same time at a device for example the temperature, humidity, and soil moisture tasks.

We assume that for tasks that have multiple executions per epoch, once the task start time of the first is determined, the remaining task executions are uniformly distributed throughout the epoch (similar to the assumption in [5]).

We also assume that the IoT network is time synchronized to enable our algorithm to detect the potential for temporal overlap if IoT devices are scheduled to operate at the same time. The last assumption is that using existing coverage algorithms, each IoT device knows its neighbors and the grid points that are shared with each neighbor.

6.3.2 Discretization and Task Abstraction

This section discusses abstractions made for the Block scheduler design. We use the concept of epochs from [3] where we divide the monitoring period e.g. 24 hours into 15-minute epochs e and we discuss the task abstractions in terms of one sensing task e.g. temperature task on the stick. However, the concept extends to multi-sensor IoT devices. In this document, we may refer to the IoT device as a PERMIT stick or stick interchangeably.

Given stick i with id s_i, we assume that the task is executed a total of N_i times per epoch. The number of executions per task and per IoT device depends on factors such as the IoT device's battery state, solar prediction, and energy goals and can be provided by an algorithm like PERMIT [3] or Signpost [53]. The time between consecutive task executions in an epoch is the period w_i and is fixed since task executions are uniformly spread throughout the epoch, i.e., the period $w_i = e/N_i$.

If a task has to complete multiple executions per epoch and the executions are uniformly distributed within the epoch, then the deadline by which the first task execution must be started will be $s_{i,dline} = w_i - t_e$ where t_e is the task execution time For example, if we have a 15-minute epoch and stick s_1 has $N_1 = 15$, this means it must take 15 temperature measurements within the epoch. The period of s_1 is $w_1 = (15 * 60)/15 = 60s$ and if the task execution time $t_e = 1s$ then the start deadline $s_{1,dline} = w_1 - t_e = 59s$ ensuring that the first task execution is completed in the worst case within the first 60s.

Since the task period is fixed at each stick, we can visualize the task executions at a given stick as a comb with N teeth (or spikes) separated by a distance w (see Fig.6.1). Since we are trying to schedule tasks at the different IoT devices in order to minimize overlap,

the goal of our scheduling algorithm is then to see how we can place multiple 'combs' along the epoch timeline such that the distance between the teeth of all the combs combined is maximized and combs do not overlap (see Fig.6.2). This is a comb placement problem that maximizes the interval between any two consecutive teeth and minimizes the number of teeth that overlap.

Figure 6.1: Task executions as a comb

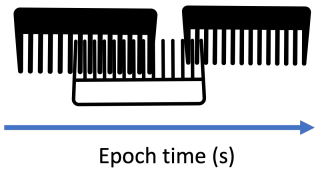

Figure 6.2: Comb placement problem

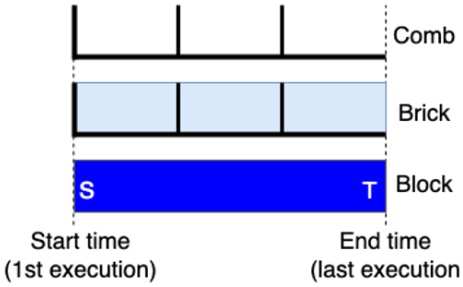

Figure 6.3: Comb to brick to block abstraction

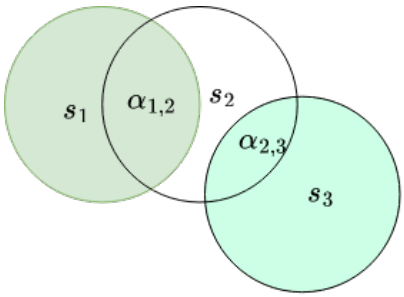

Figure 6.4: Overlapping coverage

This problem however is complex and not easily solved at an IoT device with limited computation because each comb (stick) has a different number of teeth (determined by PERMIT) and also the task periods differ based on the number of teeth.

To develop an energy-efficient solution we further relax the comb abstraction. Since the measured data value at the time of a spike is valid until the next task execution we can "spread out" the spike from one execution to the next forming a brick. This "spreading" out of the spike is a valid assumption because for example in periodic sensor measurements, data obtained from one measurement is considered valid/fresh until the next periodic measurement is due.

Therefore multiple spikes transition into multiple bricks side by side since the task executions are uniformly distributed with the epoch. The multiple bricks side by side can then be represented by a single block with a start time S when the first brick starts and an end time T when the last brick ends. This transition from comb to brick to block abstraction is illustrated in Fig.6.3. Since we are now dealing with a block representation, we can now frame the problem as a block placement problem. In addition, we only consider the temporal overlap with neighboring nodes. Therefore at a given PERMIT stick, the goal of our scheduler is to place this stick's block on top of its neighbor's blocks in a way that minimizes total overlap of all the blocks.

6.3.3 Overlap Abstraction

Applications that use IoT devices, e.g., used for agricultural monitoring may have uniform or non-uniform deployments. In either case, it is possible that there may be overlapping sensing areas where multiple sticks cover the same geographical areas. If IoT devices with overlapping sensing areas perform the same sensing task at the same time, then there is a risk of redundant data being sent as well as a waste of energy. To reduce the cost of redundant data our scheduler limits the number of neighboring sensors executing the same

sensing task at the same time by intelligent scheduling. However in order to do this we need to be able to determine a measure of overlap between neighbors. For this work, we consider two types of overlap: temporal overlap and geographical coverage overlap. We use temporal overlap to determine the number of neighboring sticks simultaneously executing the same task and this is discussed further in Section 6.5.

The second type of overlap is geographical coverage overlap and we use this to determine which devices are neighbors. We divide the monitored area into a grid where each stick covers a given number of grid points. A grid point is covered by a stick if the Euclidean distance between the stick and that grid point is less than or equal to the sensing radius of the stick. Depending on the task type, the number of grid points covered by the task may differ for example a temperature task with circular coverage would cover more grid points while the image task with conical coverage covers fewer grid points. Two sticks are neighbors as long as they share coverage of at least 1 grid point and as a result of the different task coverage patterns, each task type may have a different set of neighbors e.g. the image task neighbors may be a subset of the temperature task neighbors. To determine the neighbors of any IoT device within the network we use a data structure called an α array. This represents the geographical overlap relationship between any two IoT devices in a network of n IoT devices. The α array is an nxn matrix where each cell represents the overlap value between two intersecting sticks.

We consider two approaches for measuring the overlap: binary and area. In the binary case, we assume that if two nodes are neighbors, their overlap value is 1 otherwise it is 0 and therefore the alpha array only contains 1's or 0's. In the area-based approach, we

assume the amount of overlap between neighbors is proportional to the shared area between them. We assume each stick is able to determine the overlapping coverage area with each of its neighbors. For example assuming we have three nodes s_1, s_2 and s_3 and s_1 and s_2 are neighbors and s_2 and s_3 are neighbors (see Fig.6.4). Then using the binary approach, the α array is then $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

6.4 Protocol

This section discusses our distributed scheduling protocol which has three main phases once the token has been received at the current stick. These include extracting the neighbors information from the token, using this information to schedule the start time and updating the token before forwarding it to the next stick. We discuss each section in detail next however we first discuss the environment that the scheduler will operate in in terms of deployment patterns and the communication abstractions.

6.4.1 Deployment Patterns

There are generally two deployment patterns possible: deterministic and stochastic (or non-uniform). In the deterministic deployment pattern (or grid pattern) the user is able to predetermine the device locations and thus has some control over the amount of geographical coverage overlap between neighboring IoT devices. However placing these devices over large areas can be time-consuming and expensive, or the environment and terrain is hostile and thus a deterministic approach cannot be used. In the non-uniform deployment pattern, IoT devices are dispersed randomly over the deployment area. Our

algorithm can operate with both deployment patterns and only requires that the underlying communication network enables each stick to communicate with its neighbors.

6.4.2 Communication Abstraction

Inter-device communication

We assume the devices communicate with their neighbors using a circular Distributed Hash Tables (DHT) [91] like network. Each IoT device knows its nearest live neighbor's address, and is resilient to changes in the connectivity as is typical with DHTs [91]. This intercommunication among neighbors is maintained independent of our scheduler and the communication range is independent of the sensing coverage range.

Token Passing

We use a token-passing approach to share information between the IoT sticks and assume that existing techniques such as those for token ring networks and FDDI (IEEE 802.5) [94], [99], [95], [100]) can address challenges like handling lost or duplicate tokens. The token is generated by the leader Stick which can be elected using existing leader election mechanisms This leader node is then responsible for detecting lost and duplicate tokens and initiates token recovery after a failure [96], [97]. For scheduling using only the Block Scheduler, the token requires only one pass through the network i.e. it does a 'scheduling' round. As the token travels downstream, it carries scheduling information from the upstream nodes that have already scheduled their tasks to the downstream nodes. The information carried in the token includes the stick id s_i and information for each task k including S_i^k the scheduled task start time, N_i^k the total task executions in that epoch, and

m_i^k which is the id of the last neighbor that needs to use this stick's information. In this work, we assumed the token is processed by the sticks in ascending ID order.

Limiting Token Size One key challenge with using token-based communication is how to ensure that the token maintains a reasonable size which can be a concern, especially in large networks with multiple tasks operating at each IoT device. As the scheduling token is processed by more IoT nodes, it needs to carry additional information downstream increasing its size which can affect its propagation time and the cost of transmission at each IoT device. Therefore a mechanism to limit its size is required.

We utilize the fact that each stick only uses its neighbor's information when making its schedule. Therefore, once the token has been processed by all the downstream neighbors of a particular stick, that stick's info can be deleted from the token because that data will not be used again. To implement this solution, when a stick updates the token, it includes the ID of its last downstream neighbor (last neighbor ID m_i^k) before forwarding the token. When the token is processed at the next stick, if the current Stick ID is greater than any of the last neighbor IDs in the token, then the stick data with the affected last neighbor ID can be deleted from the token. The last neighbor ID can either be included on a per-task basis or the last overall neighbor out of all the tasks. In this work, we use the latter case and refer to the last neighbor id as m_i. Therefore as the token travels downstream, unnecessary data can be removed, thus addressing the challenge of the token size.

6.4.3 Distributed Block Scheduling Protocol

The Block Scheduling protocol has three phases: extracting the neighbors information from the token, using this information to schedule the task start time, and updating

the token before forwarding it downstream.

Extracting Neighbors Information

We assume for each of its i neighbors, the current stick knows the neighbor ID and location. The token carries information about the upstream nodes that have already been scheduled including the stick id, task start time, the total number of executions, and the maximum neighbor id (s_i, S_i^k, N_i^k, m_i). When a stick receives this information it extracts its upstream neighbor's data from the token and generates a block for each of the i neighbors tasks. The block information includes the neighbor stick id, task start time (block start time), block end time, and block height $(s_i, S_i^k, S_i^k + w_i^k N_i^k, 1/w_i^k)$. Note that since we know the epoch duration we can determine the period by $w_i^k = e/N_i^k$.

Scheduling the Tasks

The current stick generates a block b_0^k for each task k using the task start deadline, period, and the total number of executions i.e. $(S_{0,dline}^k, w_0^k, N_0^k)$. Then, given the neighbor's information in the token, it creates blocks for each neighbor and runs the Block Scheduler to determine the best start time S_0^k for its task. The Block Scheduler algorithm is discussed in detail in Section 6.5.

Updating the Token

Once the algorithm has determined the best start time S_0^k for each of the k tasks that will run on the current stick s_0 the token needs to be updated. Updating the token consists of two steps. First, we remove any information that will not be used by any other

downstream node by comparing all the last neighbor ids included in the token with the current stick id. If any id is greater than the current stick id i.e. $m_x >= s_0$ then we can delete this information from the token. The next step to update the token is to add the current stick's information to the token. The information added includes the current stick id and the task start time and task parameter values for each task and the stick's last neighbor id i.e. s_0, S_0^k, N_0^k, m_0). Once the token is updated it can then be sent downstream to the next stick.

6.5 Scheduling

Intuitively, the sensing times of the neighboring PERMIT sticks (or PERMIT sticks that have overlapping coverage areas) should be *staggered* as much as possible to minimize the possibility that those PERMIT sticks are all sensing the same area at the same time. As described in Section 6.4, for complexity reasons, we restrict the sensing times for each of the K sensors (tasks) associated with PERMIT stick to a uniform pattern, with a frequency determined by the MPC-based optimization algorithm. This implies that the only variable through which the sensing times can be optimized is to move the starting time of the sensing within the limits as described in Section 6.4. In this section, we describe a method to do it optimally (under the constraints imposed by the protocol) and in a computationally efficient manner.

6.5.1 Block Scheduling Model

Towards developing an algorithm to optimize the scheduling of the sensing tasks, we first describe a model that will allow us to meaningfully capture the quality of a given sensing schedule, and optimize it in polynomial time. We introduced the block abstraction in Section 6.3.2 and with this representation, we have 'spread out' the sensing instant temporally, for the calculation of sensing overlaps, as we see next.

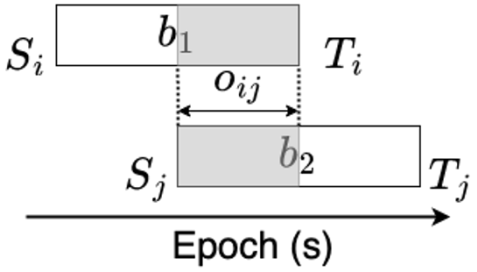

Figure 6.5: Calculating block overlap

For a given sensing pattern for each PERMIT stick, represented as a block as described in Section 6.3.2, our goal is to minimize the overlaps in the temporal coverage provided by multiple sensors that are covering the same area. Since we are scheduling the PERMIT sticks (sensor) sequentially according to their position in the DHT, when a sensor is scheduled, our objective is to place its corresponding sensing block so that it minimizes the coverage overlap with all prior sensing blocks whose positions have already been chosen.

Note that since the number of sensing points and their intervals are already determined, the width and height of the sensing block are already determined for each task k, and the only variable to optimize is the start time S_i^k (and therefore the end-time T_i^k). This

must be optimized given the start times of the previous P sensors in the DHT, and their spatial overlaps in coverage with the current stick i being scheduled. We assume that the PERMIT sticks have some sense of their relative locations (there are a variety of ways of establishing relative coordinate systems without requiring a GPS from which such overlap is easily calculated. For any other stick j, let α_{ij} denote its spatial coverage overlap with sensor i. The value of α_{ij} is obtained from the α array (discussed in Section 6.3.3) at the intersection of row i and column j for sticks i and j.

Let $O_{ij}(S_i^k)$ denote the overlap of the two sticks when the task k sensing block of stick i (which is currently being scheduled) is positioned to start at S_i^k (variable), while the position of the sensing block of j is fixed (pre-determined). The overlap $O_{ij}(S_i^k)$ is calculated as shown in Figure 6.5. More specifically, the overlap is calculated by considering the total shaded area formed by overlaying the sensing blocks of i and j, and counting only the times at which they overlap. Then, the optimization goal is to choose S_i^k so as to minimize the overlap with all J sticks scheduled prior to i, i.e.,

$$S_i^{*k} = \arg\min_{S_i^k} \sum_j \alpha_{ij} O_{ij}(S_i^k). \tag{6.1}$$

where

$$O_{ij} = \left(\frac{1}{w_i^k} + \frac{1}{w_j^k}\right)\Big(\min(T_i^k, T_j^k) - \max(S_i^k, S_j^k)\Big) \tag{6.2}$$

In general, the function $O_{ij}(S_i^k)$ is neither convex nor concave. This implies that standard convex optimization tools cannot be applied to solve this problem. Further, even though S_i^k is a scalar, it varies over a continuous space, making it a one-dimensional continuous optimization problem. Given that (6.1) needs to be solved each each PERMIT stick (which has limited computation power and memory resources), we seek to develop a low-complexity

solution to this optimization problem.

Towards that end, the effectiveness of sensing coverage in an epoch (computed across all PERMIT sticks) at any given point in space can be computed as a convex function of the area under the blocks of the sticks covering that point.

6.5.2 Block Scheduling Algorithm

This scheduler is suited for IoT devices and it utilizes current neighbor information available to make the best decision possible. The Block Scheduler Algorithm is shown in Fig.6.6 and the overlap calculation algorithm is shown in Fig.6.7. Both algorithms illustrate the case for scheduling a single task. If a device has multiple different tasks in operation e.g. temperature and humidity tasks, the algorithm will need to be run separately to schedule each task. To explain the operation of our Block Scheduler algorithm we use an example of a current stick s_0 which is scheduling a temperature task. Stick s_0 has four upstream neighbors (s_1-s_4) that have already scheduled their temperature task and have included this information in the token. In the following sections where the superscript k is omitted in some notation, it refers to variables for a particular task since we describe the operation of our algorithms with respect to a single task at a time. For example the start time at stick s_0 for task k may be referred to as S_0 or S_0^k.

Generating the Blocks

Each task k at a stick is represented by a block which is characterized by the stick id s_0, task start time S_0, end time T_0, and height $1/w_0$ where w is the task period.

Algorithm 1 Block Scheduler Algorithm

Require: $token(s_x, S_x, N_x, m_x)$
Require: $b_0(s_0, S_{0,dline}, T_{0,dline} = S_{0,dline} + w_0 * N_0, \frac{1}{w_0})$
  $output \leftarrow []$
  {Create current stick block b_0 and neighbor blocks}
  $nbors \leftarrow getNbors(s_0)$
  $m_0 \leftarrow getLastNbor(nbors)$
  $blocks \leftarrow createBlocks(token, nbors)$
  {Create wall with neighbor blocks and create transition point array y}
  $wall \leftarrow createWall(blocks)$
  $y \leftarrow getTransitionPoints(wall)$
  for t_r $in \leftarrow [0,\ T_{0,dline}]$ **do**
    if t_r not in y **then**
      $y.insert(t_r)$
    end if
  end for
  $y.sort()$
  {Check new block when start times and end times hit transition points}
  for $i = 0$ .. $len(y)$ **do**
    $S \leftarrow y[i]$
    if $S < S_{0,dline}$ **then**
      $new \leftarrow [S, S + w_0 * N_0, 1/w_0]$
      $overlap \leftarrow calculateOverlap(new, blocks)$
      $output.append() \leftarrow [S, overlap]$
    end if
    if $S >= T_{start}$ and $S <= T_{0,dline}$ **then**
      $S_T \leftarrow S - (w_0 * N_0)$
      $new \leftarrow [S_T, S_T + w_0 * N_0, 1/w_0]$
      $overlap \leftarrow calculateOverlap(new, blocks)$
      $output.append() \leftarrow [S_T, overlap]$
    end if
  end for
  {Get start time with min overlap and update the token}
  $stime \leftarrow getMinOverlap(output)$
  $new \leftarrow [s_0, stime, N_0, m_0]$
  $token \leftarrow updateToken(token, new)$

Figure 6.6: Block Scheduler algorithm

Blocks are created based on the task values determined by PERMIT. Each stick also receives information about its upstream neighbors in the scheduling token. For the tasks run at each neighbor's stick s_i, the token information includes the start time of the task S_i, its task frequency N_i, and the last neighbor id m_i. Using the neighbor task information the Block Scheduler can then create the corresponding task blocks for each neighbor's task. Using our example, stick s_0 creates block b_0 for its own temperature task (see Fig. 6.8), and blocks

Algorithm 2 Calculate overlap

Require: $blocks(s_x, S_x, T_x, \frac{1}{w_x})$
Require: new block b_0 $(s_0, S_0, T_0, \frac{1}{w_0})$
  $overlap \leftarrow 0$
  $blocks.append(b_0)$
  $alpha \leftarrow getAlpha(blocks)$
  for $i = 0$ .. $len(blocks)$ **do**
    for $j = 0$ .. i **do**
      $overlap \leftarrow overlap + alpha[i,j] * \max\left(0, \left[(\frac{1}{w_i} + \frac{1}{w_j}) * (\min(blocks[i,2], blocks[j,2]) - \max(blocks[i,1], blocks[j,1]))\right]\right)$
    end for
  end for

Figure 6.7: Overlap calculation algorithm

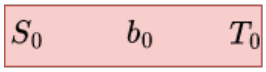

Figure 6.8: Block at current stick

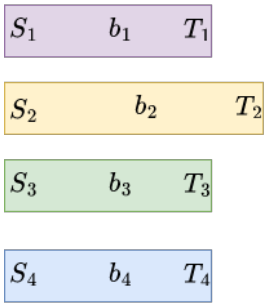

Figure 6.9: Neighbors blocks

b_1, b_2, b_3 and b_4 for the tasks scheduled at its neighbors (see Fig. 6.9).

Creating the Wall

After creating the blocks, the block scheduler arranges the neighbors blocks along an epoch timeline based on the start time of each block. This forms a "wall structure" where blocks with overlapping start and/or end times are placed one on top of another (see Fig 6.10). From the wall structure, we can then clearly identify "transition points" where the total height of the wall changes as blocks are added/removed from the wall. A transition point array, y is used to store the transition points. See Fig 6.11 shows the wall

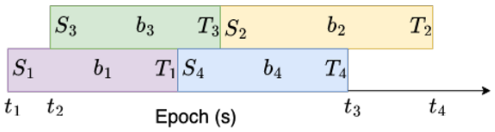

Figure 6.10: Wall with different blocks

Figure 6.11: Wall transition points

with transition points at t_1, t_2, t_3, and t_4 which are then stored in y.

Evaluating Overlap

Once we have the wall structure and the transition points, the goal of the scheduler is to find the start time that does not exceed its start deadline and minimizes the total block overlap with all other neighbor blocks that have already been placed on the wall. This requires positioning the block at different start times and evaluating the overlap at each point before selecting the best start time

A brute-force method is to place the new block at every interval from 0s to the deadline and calculate the overlap at each time interval. However, if you have a small interval e.g. 1s and a long epoch duration, this approach can be time-consuming especially if the deadline is at the end of the epoch. On the other hand, if you have a large interval then you may miss start times that give you potentially less overlap because they fall with a large interval.

We use a more efficient method by leveraging the transition points. We note that the height of the wall only changes at the transition points and it is only when the new block is placed at those transition points that the height and hence the overlap will change. This reduces the number of evaluations to a maximum of $2(t_r + 1)$ where t_r is the number of

transition points along the current wall. We multiply by 2 because we evaluate the overlap when either the start or end boundaries of the current block b_0 hits a transition point. We included two more transition points in the transition point array y to cater to three cases. The first case is where the existing wall starts at $t > 0$ (i.e. the first transition point $t_1 > 0$). Since it is possible to place the new block at $t = 0$ we need to add this as a transition point. The second case occurs where the wall ends before the task end deadline of the current block i.e. $t_4 < (t_{startdline} + N_0 * w_0)$. If the wall ends before the task end deadline of the current block then we add its end deadline as a transition point.

Our algorithm places the current block at either $t = 0$ or $t = t_1$ and then calculates the total overlap between the current block and all blocks in the wall. Our scheduler then slides the block across the wall until either the start or end boundary of the current block hits a transition point and then calculates the total overlap again. This process is repeated until the current block start time is at its start deadline. Figs.6.12-6.18 illustrate this process and how we evaluate the overlap at the transition points.

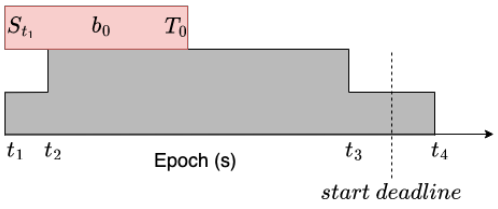

Figure 6.12: Placing block b_0 at t_1

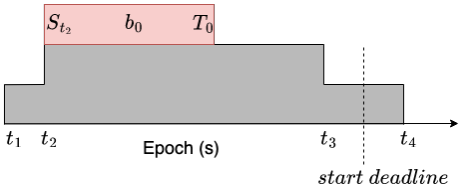

Figure 6.13: Placing block b_0 at t_2

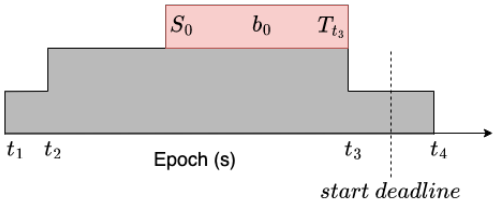

Figure 6.14: Placing block b_0 at t_3

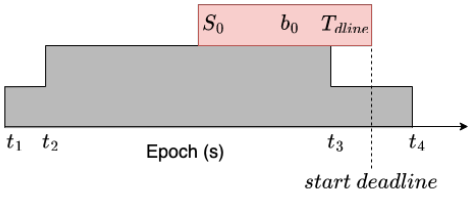

Figure 6.15: Placing block b_0 at t_{sdline}

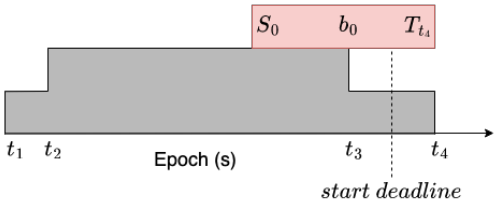

Figure 6.16: Placing block b_0 at t_4

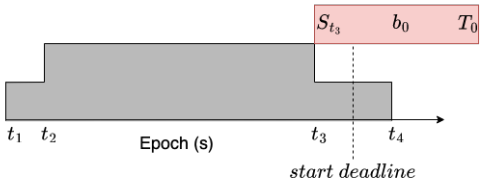

Figure 6.17: Placing block b_0 at t_3

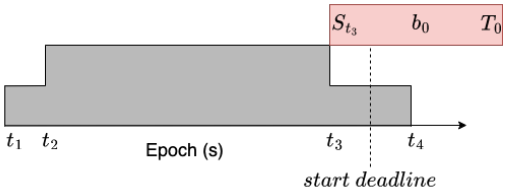

Figure 6.18: Placing block b_0 at t_{sdline}

Selecting the Start Time

After assessing the overlap at each transition point, the ultimate starting time is established by choosing the time that results in the least overlap. In instances where multiple starting times yield identical overlaps, our algorithm opts for the later time. This decision guarantees that sticks that receive the token later in the process and have more stringent start deadlines, still have the opportunity to start earlier without risking temporal overlap.

In our example, the start time with the minimum overlap is at the task start deadline of stick s_0 $S_{t_0} = t_{sdline}$.

6.5.3 Benchmark Algorithms

We compare the performance of our Block Scheduler protocol with several benchmark algorithms. We consider our algorithms: a random scheduler, an optimal scheduler using non-linear programming (NLP), an Iterative Distributed Block Scheduler, and an Alternate Scheduler. The Iterative Distributed and non-linear optimization approaches are centralized algorithms and the NLP solution provides an upper bound to the performance of our Block Scheduler which operates in a distributed manner.

Random Algorithm

With this algorithm, as the token travels to each IoT device, a random start time is selected for each task. However, in order to ensure that all task executions can be completed within the epoch we limit the start time to be in the interval from 0 to the start deadline. The random scheduler operates in a decentralized manner and does not consider neighbors' information when determining a start time.

Centralized Optimal Scheduler

This algorithm is a centralized scheduler that can operate at the sink and incorporates knowledge from all the sticks including the task values and overlap between the nodes. The optimization problem formulation for a single task k is shown in Equation 6.3 and 6.4 and we use Gurobi to find the optimal solution. The results were also verified using

an Integer Linear Programming approach.

$$\min \sum_{ij} \alpha_{ij} \max\left(0, o_{ij}\right), \tag{6.3}$$

where

$$o_{ij} = \left(\frac{1}{w_i} + \frac{1}{w_j}\right) \left[\min(S_i + N_i w_i, S_j + N_j w_j) - \max(S_i, S_j)\right] \tag{6.4}$$

Iterative Distributed Scheduler

The main difference between the Block Scheduler and the Iterative Distributed Block Scheduler algorithm is that the latter works with information from all the neighbors, not only the upstream ones. In addition, with the iterative approach, it keeps randomly selecting an IoT device to schedule until the solution converges i.e. the overlap value does not change even when we iterate through all the nodes again. The scheduler Algorithm is shown in Fig.6.19.

Alternate Scheduler

The Alternate Scheduler alternately schedules the start times of a task at 0 or the start deadline as the token traverses the network. This approach was selected because, from our analysis, both the Block Scheduler and the Iterative Distributed Scheduler tend to schedule the task start times at either 0 or the start deadlines. Therefore we evaluate the performance difference with the Alternate Scheduler which can also potentially operate as a simpler more lightweight distributed scheduler.

Algorithm 3 Iterative Scheduler Algorithm

Require: $all\ stick\ data(s_x, S_{x,sdline}, N_x, m_x)$
  $converged \leftarrow False$
  $token \leftarrow []$
  while $!converged$ **do**
    $s_0 \leftarrow getRandomSid()$
    $ntoken \leftarrow [s_0, S_{0,sdline}, N_0, m_0]$
    $b_0 \leftarrow createBlocks(ntoken)$
    $s_{time} \leftarrow runBlockScheduler(b_0, token)$
    $ntoken \leftarrow [s_0, s_{time}, N_0, m_0]$
    $token \leftarrow updateToken(ntoken, token)$
    $blocks \leftarrow createBlocks(token)$
    $converged \leftarrow checkConvergence(blocks)$
  end while

Figure 6.19: Iterative Distributed Scheduler algorithm

6.6 Task Adaptation

In this section, we discuss our Distributed Task Adaptation (DTA) algorithm. We introduce the concept of coverage threshold C_T which is the minimum number of measurements or sensor readings required per grid point, per epoch, per task. For example for a temperature task, if $C_T = 9$ then we require at least 9 temperature measurements over each covered grid point per epoch. The coverage threshold is determined by the application and only applies to grid points that are covered by IoT devices.

Achieving the coverage threshold requires coordination between the different neighboring sticks, especially when multiple IoT devices cover the same grid points. The coverage value over grid point g for task k in epoch n is denoted by $c^n_{g,k}$. It can be determined by adding the total number of task executions for task k from all sticks covering grid point g.

If the calculated coverage value is greater than the required coverage threshold $c^n_{g,k} > C_T$, the device is wasting energy because the extra sensing measurements are not required. If the coverage value is less than the required coverage value $c^n_{g,k} < C_T$ then the grid point is under-covered, and insufficient information is being provided regarding that grid point. Our target is to ensure that the coverage value is equal to the coverage threshold $c^n_{g,k} = C_T$, which satisfies the coverage requirements without spending excess energy.

Our DTA algorithm determines how to adapt the task values at an IoT device by using the neighbor's task information to ensure that the coverage value over all its grid points meets the coverage threshold. Our algorithm uses tokens for sharing information between the IoT devices and the tokens traverse the network in 2 rounds. We discuss the operation of the token in each round next.

6.6.1 Token Travelling in Forward Direction (North-South)

Consider the example deployment shown in Fig.6.20 where Stick 1 covers grid points 1, 2, 3 and 4 and each grid point is shared with a different neighbor. Figs.6.21-6.23 illustrate the operation of the algorithm for a single task when the token is travelling in the forward direction (North-South) and arrives at Stick 1 after being forwarded from its nearest upstream neighbor.

DTA first extracts the neighbors task information and determines the current coverage of all Stick 1's grid points ($c_{g_1}, c_{g_2}, c_{g_3}$ and c_{g_4}) for grid points 1-4 respectively (represented as the blue bars in Fig.6.21). For each grid point, the algorithm determines the difference between the current coverage value and the coverage threshold C_T, and these differences are represented by the red bars in Fig.6.22. DTA then finds the mean of these

values i.e. $d = mean(d_1, d_2, d_3, d_4)$. The updated task value d_f is the minimum of the upper bound task value f_k determined from PERMIT and d i.e. $d_f = min(f_k, d)$. Fig. 6.23 shows the updated coverage values when we recalculate the coverage and include the current sticks task values. Stick 1 then includes its updated task value d_f in the token and sends it downstream to its next neighbor. In this way, each IoT device adapts its task value based only on information provided by its upstream neighbors.

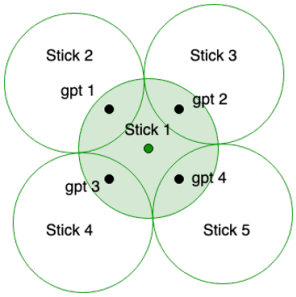

Figure 6.20: DTA deployment example

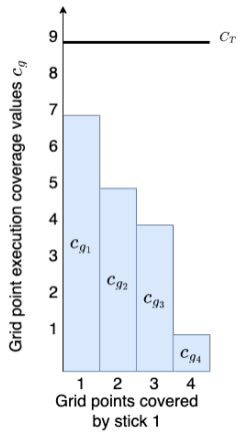

Figure 6.21: Token 1 - Determine coverage

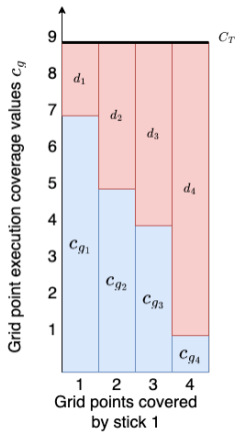

Figure 6.22: Token 1 - Calculate task value

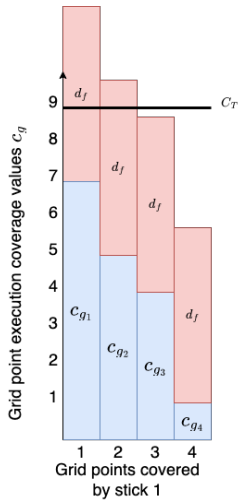

Figure 6.23: Token 1 - Update token

6.6.2 Token Travelling in the Reverse Direction (South-North)

When the token returns to the leader node after the first round, the leader node then generates the second token which travels in the reverse direction. The token now carries information about the downstream neighbors to the upstream neighbors. Once the token arrives at a stick and calculates the current coverage of its grid points there are two cases to consider.

Case 1: Coverage of all grid points is greater than the threshold. This case is illustrated in Figs.6.24-6.26. If the coverage value of all grid points is greater than the threshold (as shown by the blue bars in Fig.6.24) then we have an over-coverage scenario and DTA takes this opportunity to reduce the over-coverage. Using information provided in the token, DTA determines the excess coverage per grid point (d_1, d_2, d_3 and d_4 which are represented by the green bars in Fig.6.25. DTA then finds the maximum value that can be deleted while ensuring all coverage remains at least C_T which is d_3 in the example. Calculation of the current coverage values uses the task value set by Stick 1 in the first round i.e. d_f, therefore, the new task value $d_f^{'}$ is the initial value minus the maximum value that can be deleted ($d_f^{'} = d_f - d_3$). This reduces the over-coverage but ensures that the grid point coverage of all the covered grid points does not go below the threshold. Fig.6.26 shows the updated grid point coverage using the updated $d_f^{'}$ values. The token is then updated and forwarded upstream.

Case 2: Coverage of at least one grid point is less than the threshold. This is illustrated in Figs.6.27-6.29 If the coverage value of at least one grid point is less than the threshold as shown by the blue bars in Fig.6.27) then we have an under-coverage

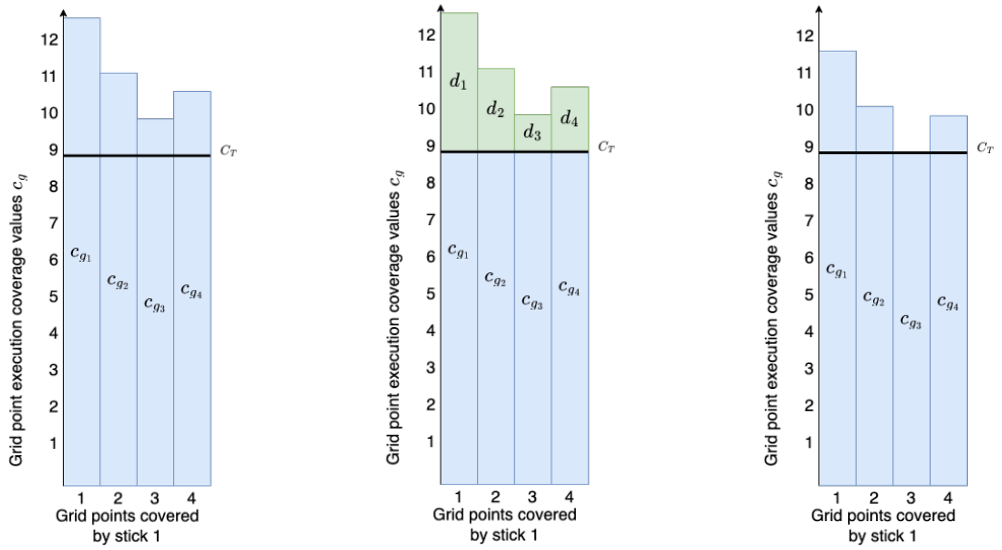

Figure 6.24: Token 2 (Case 1) - Determine coverage

Figure 6.25: Token 2 (Case 1) - Calculate task value

Figure 6.26: Token 2 (Case 1) - Update token

scenario and DTA takes this opportunity to adapt the task values to reach the coverage threshold. Using information provided in the token, DTA determines the under coverage per grid point (d_1, d_2, d_3 and d_4 which are represented by the red bars in Fig.6.28). DTA then finds the grid point with the minimum coverage value (grid point 4 in this case which needs d_4 more task executions to meet the threshold) and tries to pull up this coverage as much as possible. Calculation of the current coverage values uses the task value set by Stick 1 in the first token round (d_f) therefore the new task value $d_f^{'}$ is the minimum of the upper bound given by PERMIT f_k and the task value set during token round 1 plus the maximum value needed to push coverage of the most under-covered grid point up to the threshold i.e. in this case ($d_f^{'} = min(f_k, d_f + d_4$. This reduces the under-coverage and the token is updated and forwarded upstream.

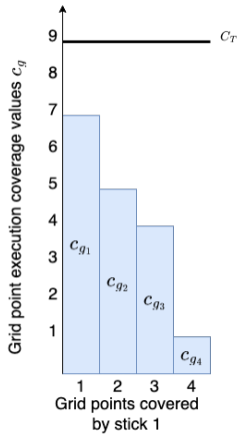

Figure 6.27: Token 2 (Case 2) - Determine coverage

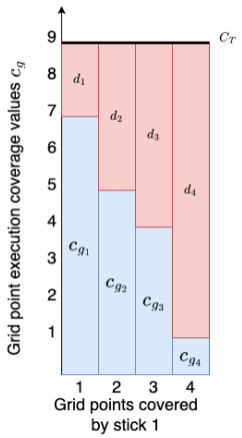

Figure 6.28: Token 2 (Case 2) - Calculate task value

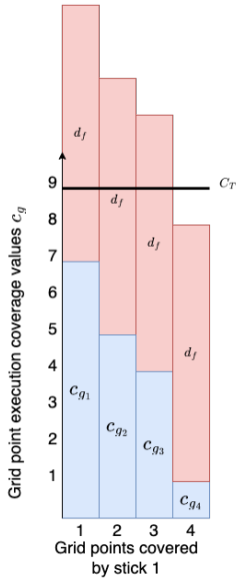

Figure 6.29: Token 2 (Case 2) - Update token

6.7 Evaluation and Results

In this section, we evaluate the performance of our DTA and Block scheduling algorithms using simulations and discuss our results in two sections. First, we demonstrate the operation and benefits of our algorithms using a small network and varying some parameters under our Base Case results. Then we present our results for a larger network. The main metrics we use for evaluation include total energy used over the experiment and the average total overlap.

6.7.1 Base Case Experiments

In the base case experiments, we use 8 IoT devices deployed in a 10x10 grid. Each IoT device can run three tasks (image, temperature, and humidity tasks) and the variable parameter in all cases is the frequency of execution of the tasks. The temperature and

humidity tasks have a circular coverage area with a radius of 2m while the image task has a conical coverage area with a radius of 2m. The experiments were run over 48 hours using a cloudy solar profile for both days and we used a low initial starting energy of approximately 6,000J at each device. The ID of each IoT device used corresponds to the grid point where it is located. For these experiments, the coverage threshold C_T was set to 9 and we slightly varied both the starting battery voltage and solar patterns for each IoT device. For the small network, we show the results when running the experiment using a non-uniform deployment pattern and a uniform deployment pattern (see Figs. 6.30 and 6.31).

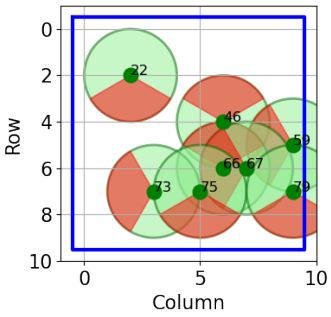

Figure 6.30: Non-uniform deployment

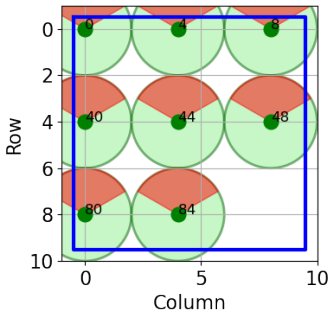

Figure 6.31: Uniform deployment

Non-uniform Deployment

In the non-uniform deployment experiment, as illustrated in Fig. 6.30, some sticks may have overlapping coverage (e.g. Stick 66) while others do not have overlapping coverage with any other nodes e.g., Stick 22. We first run the experiment with the DTA algorithm disabled and compare the performance between the five different schedulers (Block, Ran-

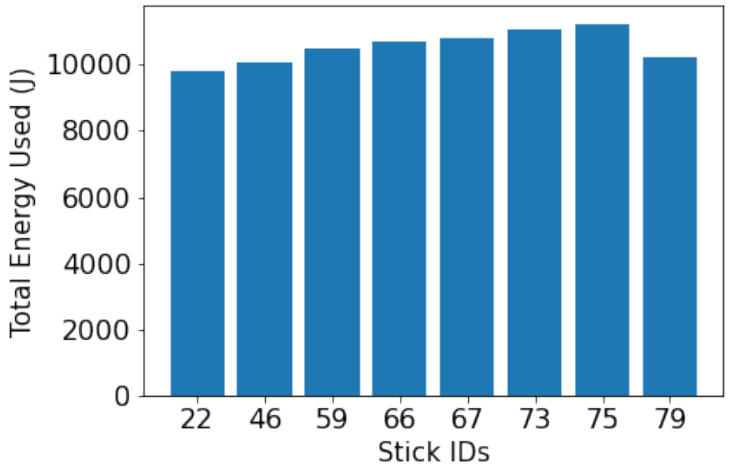

Figure 6.32: Non-uniform deployment base case total energy (no adaptation)

dom, Optimal (NLP), Iterative Distributed, and Alternate Schedulers).

Schedulers with No Adaptation With no adaptation, the original task values provided by PERMIT are used in all cases and there is only a slight difference in the total energy used at each stick due to variations in the initial starting battery and task values selected as illustrated in Fig.6.32. The battery discharge pattern shown in Fig. 6.33 shows the battery slowly discharging throughout the day for all sticks till all the devices shut down temporarily early on day 2. As the day progresses, more solar energy is available so the IoT devices are able to charge their batteries more and perform sensing tasks. However, since it is a cloudy day, the batteries are not charged fully and at around epoch 80 the IoT devices shut down again and do not recover.

The task values provided by PERMIT for the image, temperature, and humidity

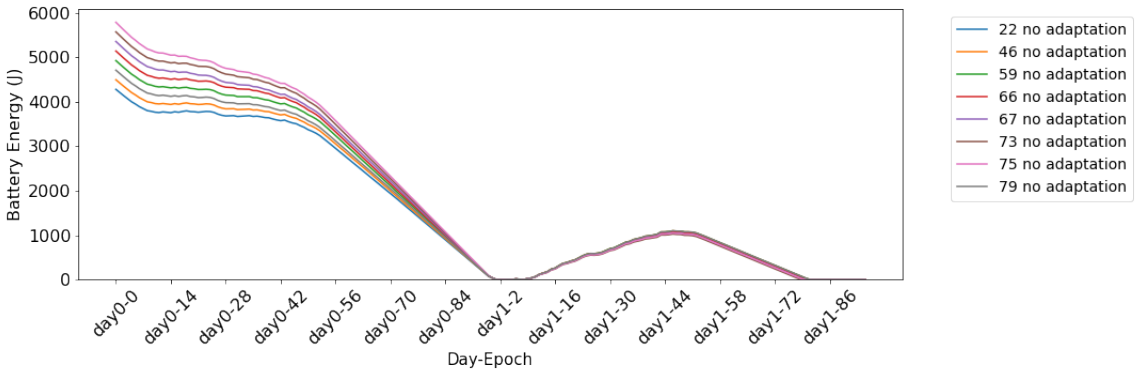

Figure 6.33: Non-uniform deployment base case battery energy (no adaptation)

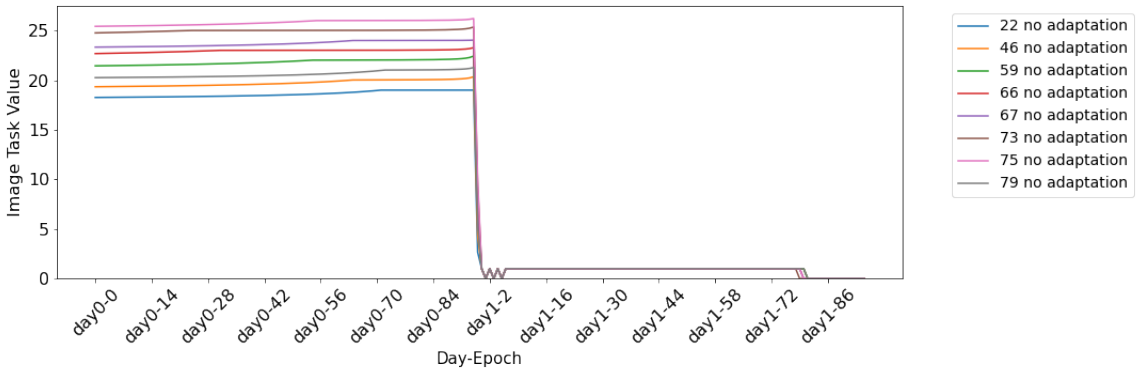

Figure 6.34: Non-uniform deployment base case image task values (no adaptation)

tasks are shown in Figs. 6.34-6.36). Note that all the schedulers use the same task values. From the figure, we note that on day 1 PERMIT provides high task values due to higher battery levels, and on day 2 it provides only minimum task values for the entire day.

In Fig.6.37, we illustrate the energy consumption per epoch, highlighting distinct patterns. On day 1, when task values are high, the energy consumption per epoch is notably elevated. In contrast, on day 2, the energy per epoch diminishes considerably due to low task values which are selected because of low battery energy. Furthermore, the figure

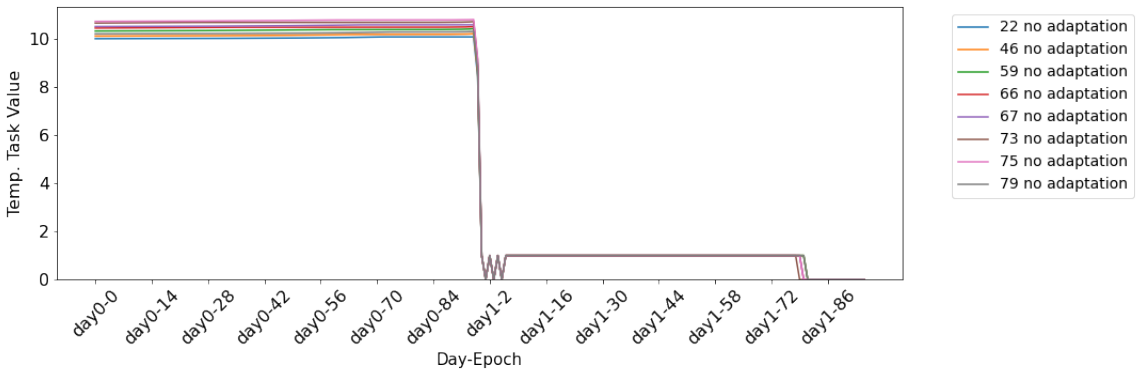

Figure 6.35: Non-uniform deployment base case temperature task values (no adaptation)

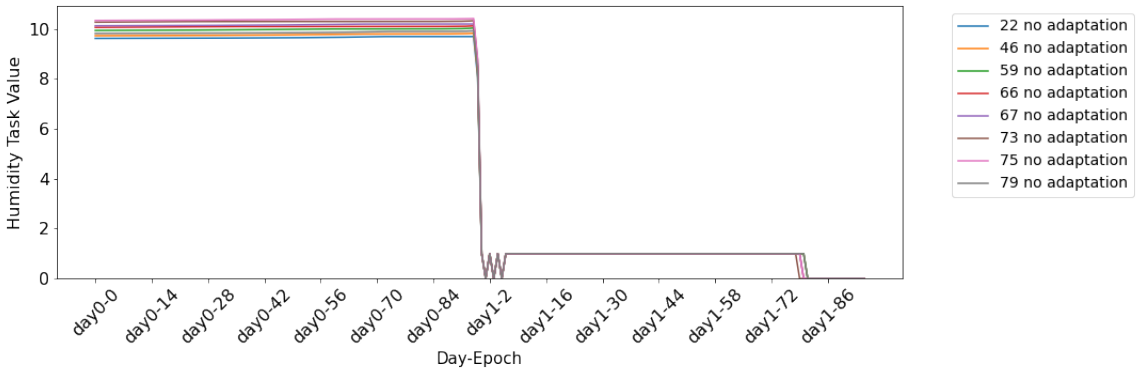

Figure 6.36: Non-uniform deployment base case humidity task values (no adaptation)

reveals a recurring sawtooth pattern during day 2, specifically in epochs 1, 3, 5, and 8. In these epochs, energy consumption alternates between approx. 40J and zero because the devices intermittently power down. This behavior arises because, within a single epoch, the battery accumulates enough charge to power up the device momentarily. However, any stored energy is immediately allocated to task execution, depleting the battery and leading to device shutdown. As the day progresses and more solar energy becomes accessible, the battery can be charged more effectively, mitigating this sawtooth pattern. Nevertheless, due

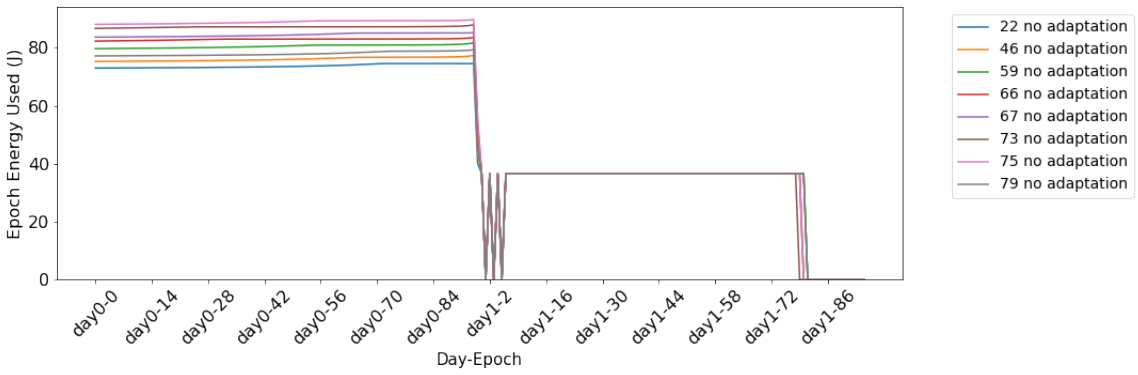

Figure 6.37: Non-uniform deployment base case epoch energy (no adaptation)

to overcast conditions, the battery cannot accumulate sufficient energy to sustain device operation throughout the entire day, ultimately resulting in device shutdown at epoch 80.

We also compare the temporal overlap obtained by each scheduler after running the experiments multiple times and the average overlap for each scheduler and task is shown in Fig.6.38. Since the goal is to minimize overlap, the results show that the Block Scheduler performs best with an overlap value close to the optimal (NLP) solution. The Iterative Distributed Scheduler and Alternate Schedulers perform next best and the Random scheduler has the worst performance.

The Iterative Distributed Scheduler, which leverages information from all neighbors and continues running until convergence, exhibits a slightly lower performance compared to the Block Scheduler. This discrepancy arises from the fact that the Iterative Distributed Scheduler's final converged overlap value is influenced by its initial "seed" during simulation. Consequently, in each experiment, we execute the Iterative Distributed Scheduler multiple times and then compute the average of the results to obtain the final converged value. This averaged result serves as the basis for comparison with the Block

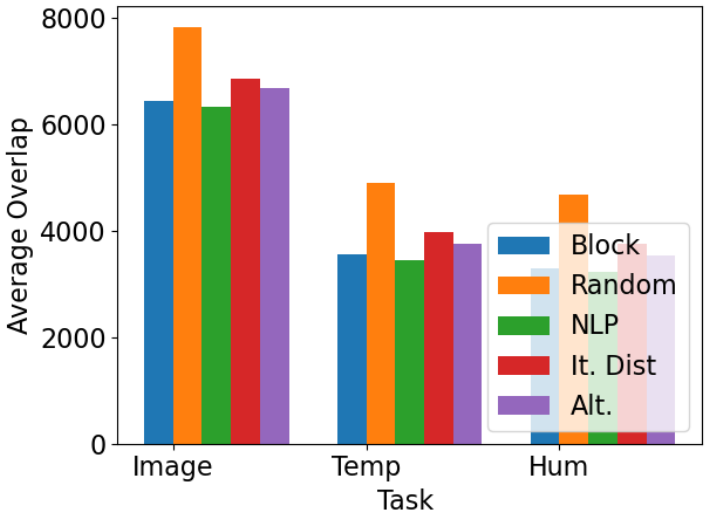

Figure 6.38: Non-uniform deployment base case average overlap (no adaptation)

Scheduler. Occasionally, due to this averaging process, the Iterative Distributed solution may yield a higher value than the Block scheduler.

The average overlap results show that the Block Scheduler can significantly reduce temporal overlap compared to the Random and Alternate schedulers. However, since there is no task adaptation, the energy used is still the same for all schedulers. Therefore we next evaluate the impact of adding task adaptation.

Schedulers with Adaptation Fig.6.39 compares the total energy used at each IoT device over the experiment when we add task adaptation using our DTA algorithm to the Block Scheduler. The DTA algorithm reduces the task values to be used at each IoT device (depending on the coverage threshold) and thus provides energy savings. In the figure, if we compare the total energy used with and without adaptation, we see energy savings at all sticks. However, with Stick 66 we see significantly higher energy savings with

task adaptation. This is because all the grid points covered by Stick 66 are also covered by its neighboring sticks (see Fig.6.30). During the second token run when the DTA algorithm is executed at Stick 66, it can reduce the task value at the device to the minimum value to reduce over-coverage without reducing the total coverage of all its grid points below the threshold C_T.

Fig.6.40 illustrates how the battery energy changes throughout the experiment at each stick. Similar to the previous experiment, without adaptation all the devices shut down at the beginning of the second day due to low battery (see the solid lines in the figure). Although the devices recover after a few epochs of charging, since the solar is low on a cloudy day the battery is still low and the devices shut down again before the end of the day. If we compare this with the battery energy using the adapted task values (dotted lines) we see that the devices remain on for the full 48 hours and Stick 66 saves much more energy than the others as explained above.

Since the PERMIT solution does not rely on weather forecasts, using our DTA algorithm can be beneficial for saving additional energy that can allow IoT devices to operate over a longer period.

Figs.6.41 - 6.43 show the task values selected every epoch and compare the initial values determined by PERMIT (solid lines) and the updated task values (dotted lines) used after running DTA. If we consider the image task in Fig.6.41 on day 1 without adaptation (solid lines) we use high task values and on day 2 due to low energy the devices can only use

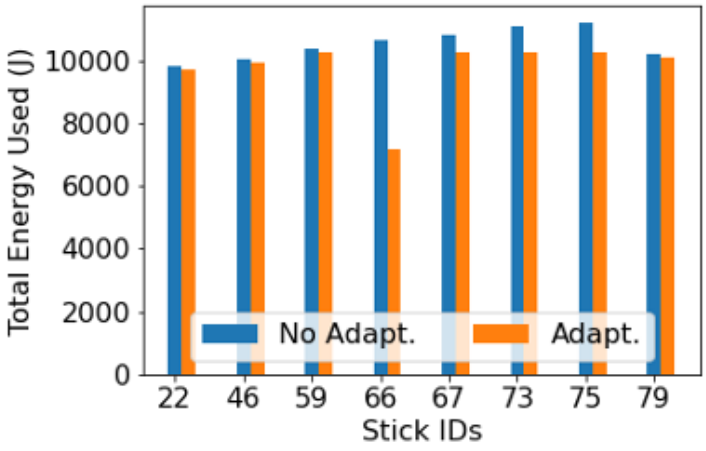

Figure 6.39: Non-uniform deployment base case total energy

low task values. With adaptation (dotted lines)using our DTA algorithm, the IoT devices are able to use lower task values on day 1 but ensure that all the tasks can be run over the two days. We also note that on day 1 when the neighbors of Stick 66 have sufficient energy to meet the coverage threshold using cooperative sensing Stick 66 uses only minimum task values. However on day 2 when some of the neighbors of Stick 66 (e.g. sticks 46 and 67 between epochs 1 and 71) can only use lower task values due to lower energy availability, Stick 66 increases its task value to ensure that the coverage threshold is met for all the grid points it covers. This illustrates how cooperative sensing enables IoT devices with higher energy to compensate for devices with lower energy to ensure the coverage threshold is met. Figs.6.42 and 6.43 show a similar pattern for the temperature and humidity cases respectively. The epoch energy results here shown in Fig.6.37 show a similar pattern to the results of the no adaptation case shown in Fig.6.44 i.e. with no adaptation (solid lines) there is high epoch energy on day 1 and low epoch energy on day 2 because the battery

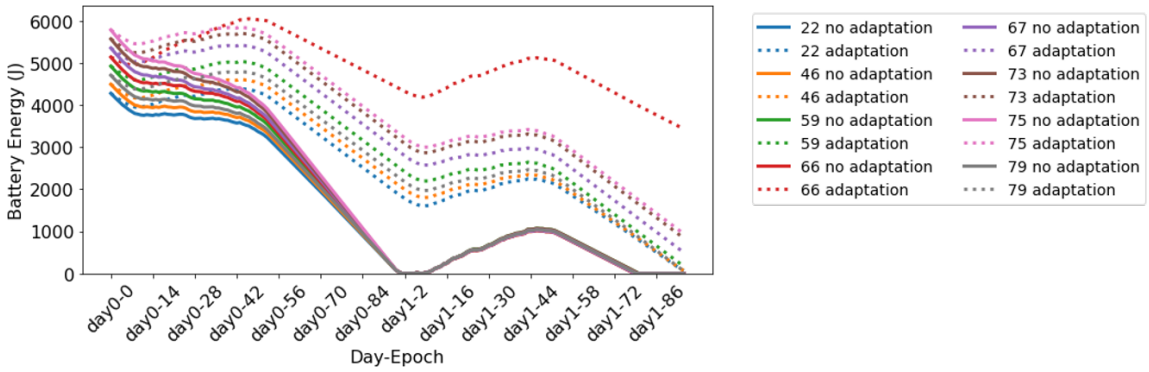

Figure 6.40: Non-uniform deployment base case battery energy

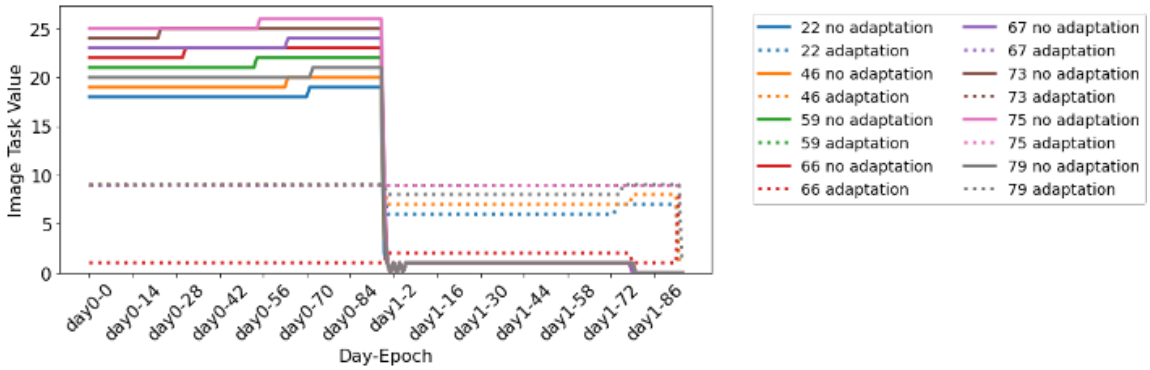

Figure 6.41: Non-uniform deployment base case image task values

is discharged. However in the adaptation case (dotted lines), the epoch energy remains relatively fixed and stable since the task values have been adapted and cooperative sensing between neighbors allows nodes with high energy to compensate for those with low energy.

For the non-uniform deployment case, we also compare the grid point coverage per grid point per epoch to evaluate whether we can achieve coverage equal to the required coverage threshold. The heat maps in Figs.6.45-6.50 illustrate the grid point coverage per grid point per epoch over the entire experiment and we compare the case using the Block Scheduler with and without adaptation. Red indicates grid points that are under-covered

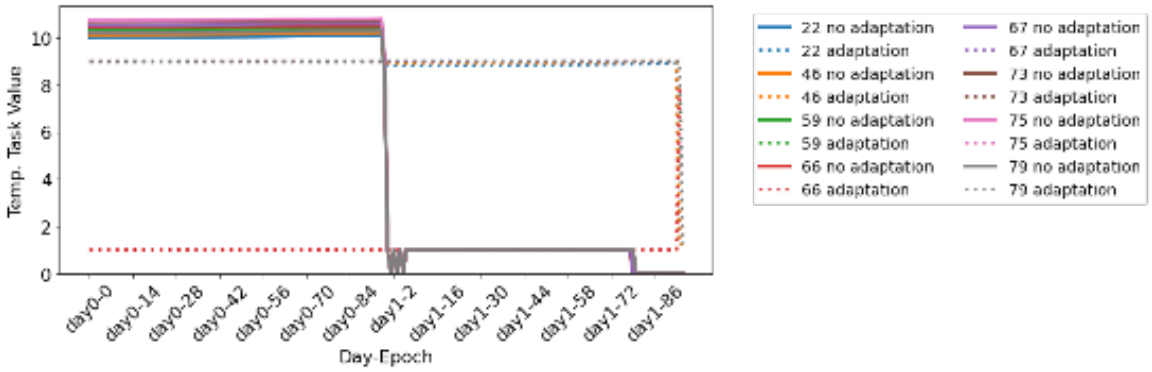

Figure 6.42: Non-uniform deployment base case temperature task values

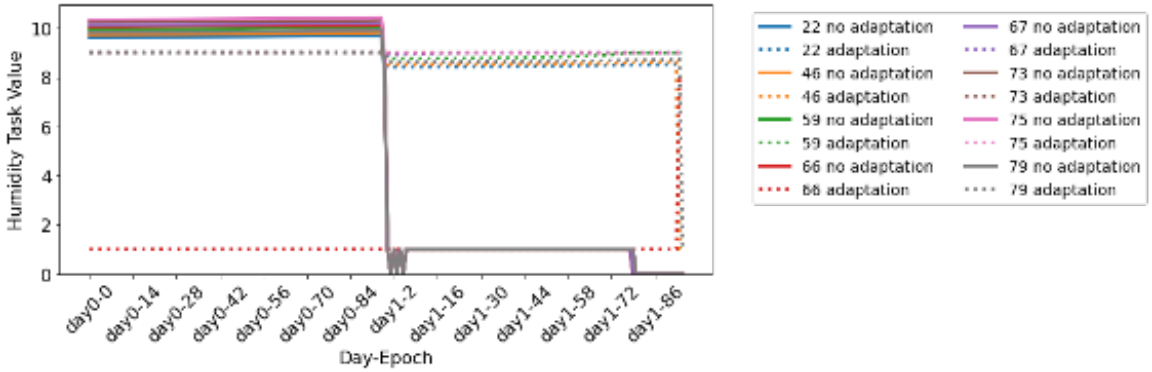

Figure 6.43: Non-uniform deployment base case humidity task values

i.e. the coverage threshold has not been met and thus we have low information utility. Green indicates grid points that are over-covered i.e. the coverage threshold was exceeded and thus we are wasting energy. Blue indicates grid points that are at the required coverage threshold and is our target.

Fig.6.45 compares the grid point coverage in the image task case without adaptation. We have a significant amount of over-coverage (green) on the first day and under-coverage (red) on the second day indicating that energy is being wasted on day 1 and insufficient information is being provided on day 2. However with adaptation (Fig.6.46)

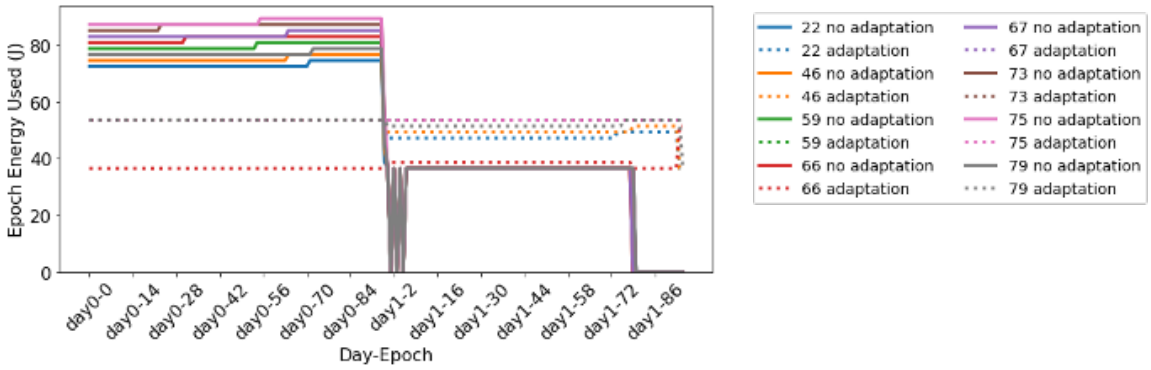

Figure 6.44: Non-uniform deployment base case epoch energy

we significantly reduce both the over-coverage and under-coverage and ensure the coverage threshold is met. We see similar performance improvements with the temperature task (compare Figs.6.47 and 6.48) and the humidity task compare Figs.6.49 and 6.50).

We also compared the total grid point coverage over the entire 2-day experiment and these results are shown in Figs.6.51-6.53. The figures show the total coverage counts and percentage of grid points that are either over-covered, under-covered, or at the coverage threshold during that experiment. In Fig.6.51, with no adaptation, 53.5% and 46.6% of the grid points are either under-covered or over-covered. When we add adaptation, the number of under-covered grid points reduces to 20.9%, the number of over-covered grid points reduces to 10% and the number of grid points at the coverage threshold increases to 69%.

Figs.6.52 and Fig.6.53 show similar results. However, in this case, the decrease in over-coverage with adaptation is less for image (44.5% to 38.8%) or slightly higher for humidity (27.4% to 38.6%). The discrepancy between the image task and these tasks can be attributed to their coverage patterns. Specifically, the image task employs a conical coverage pattern that encompasses fewer grid points, while the temperature and humidity tasks

employ circular coverage patterns, extending to a greater number of grid points. Reducing the under-covered grid points at a particular stick means increasing task executions and any change in task executions affects all the grid points covered by that stick. This then results in over-coverage of some of its grid points, especially those shared with other neighboring sticks. Since the image coverage pattern covers fewer grid points, this effect is less pronounced.

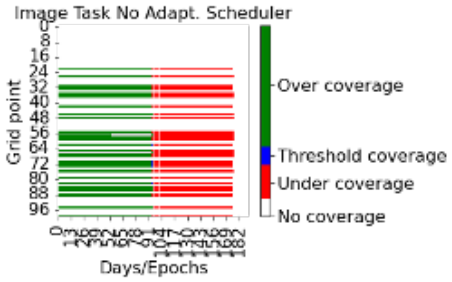

Figure 6.45: Grid point coverage: image task (no adaptation)

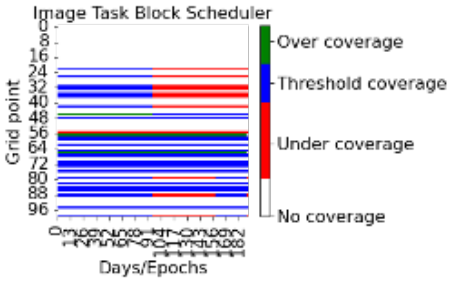

Figure 6.46: Grid point coverage: image task (with adaptation)

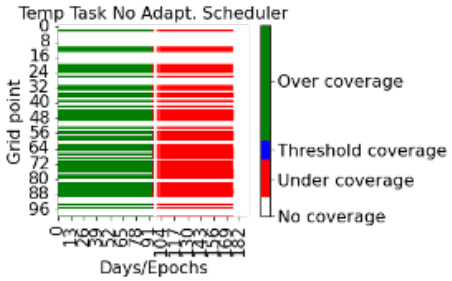

Figure 6.47: Grid point coverage: temp task (no adaptation)

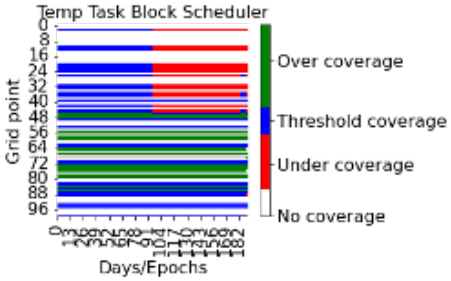

Figure 6.48: Grid point coverage: temp task (with adaptation)

Lastly, we compare the average overlap in the non-uniform deployment case to evaluate the performance of the different schedulers. Fig.6.54 shows that the Block scheduler performs

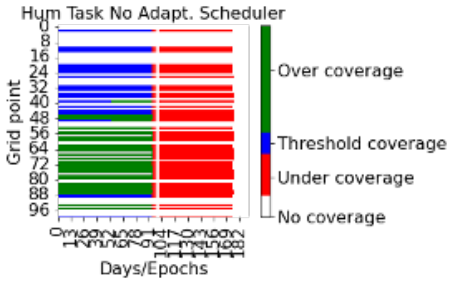

Figure 6.49: Grid point coverage: hum task (no adaptation)

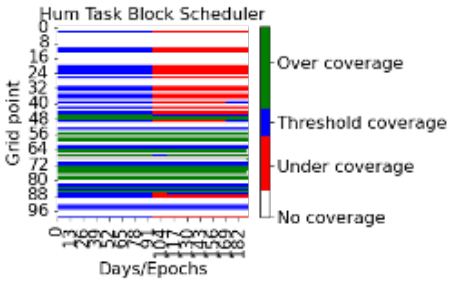

Figure 6.50: Grid point coverage: hum task (with adaptation)

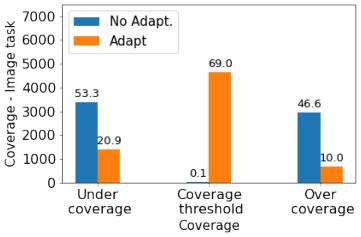

Figure 6.51: Base case image grid point coverage

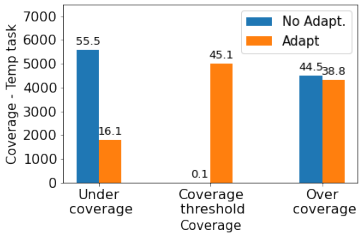

Figure 6.52: Base case temperature grid point coverage

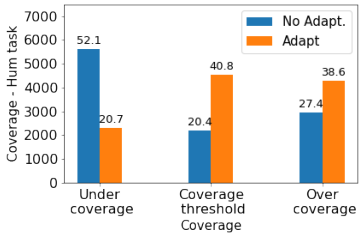

Figure 6.53: Base case humidity grid point coverage

best with values close to the optimal and the Random Scheduler performs worst and this result holds for all the tasks. Since the DTA algorithm adapts the task values close to the coverage threshold for all the tasks, the average overlap values for all the tasks are in a similar range i.e. less than 200. Fig.6.55 shows the percentage energy savings per IoT device as a result of using task adaptation. The results show energy savings ranging from 0.16% to 36% per IoT device and averaged at 5% per IoT device.

Uniform Deployment

The uniform deployment experiment setup is illustrated in Fig. 6.31. In this deployment, sticks have very limited overlapping coverage (e.g. at the coverage area boundaries). We run the experiment with the DTA algorithm and compare the performance be-

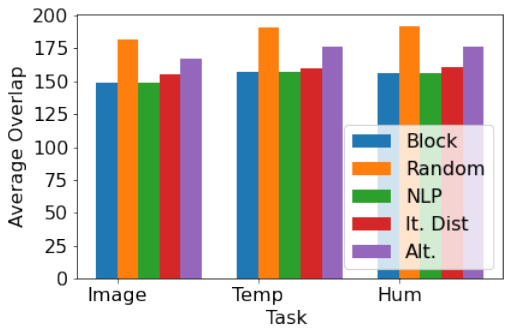

Figure 6.54: Non-uniform deployment base case average overlap

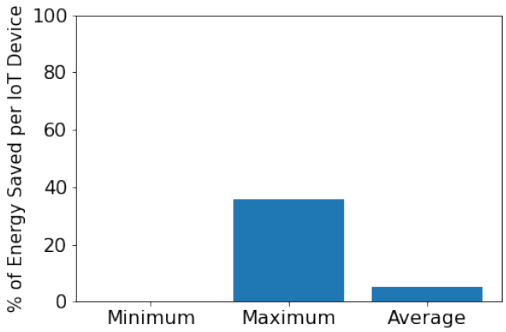

Figure 6.55: Non-uniform deployment base case average energy saved per device

tween the five different schedulers (Block, Random, Optimal (NLP), Iterative Distributed, and Alternate Schedulers).

Fig.6.56 compares the total energy used at each IoT device over the experiment when we use task adaptation. The DTA algorithm reduces the task values to be used (depending on the coverage threshold) and thus provides some energy savings. However, compared to the non-uniform deployment case, the energy savings here are modest. This is because the amount of overlap is low so each device can only adapt to meet the coverage threshold value C_T.

Fig.6.57 illustrates how the battery energy changes throughout the experiment at each stick. Devices with higher starting battery levels are able to adapt and operate over the full 2 days e.g., stick 80 and 48. However, some devices like stick 0, have a low starting battery and thus low task values that cannot be adapted. In addition, it has no neighbors who can compensate and cover its grid points. Therefore for this IoT device, there is no

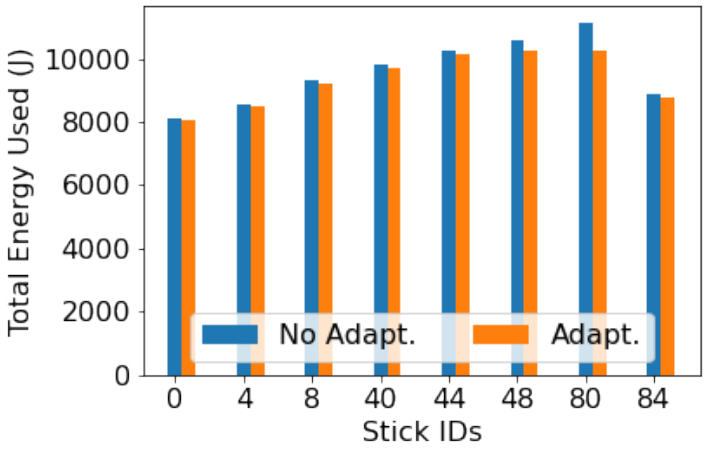

Figure 6.56: Uniform deployment base case total energy

difference between the adaptation and no adaptation case.

Figs.6.58 - 6.60 show the task values selected every epoch and compare the initial values determined by PERMIT (solid line) and the updated task values (dotted lines) used after running DTA. If we consider the image task in Fig.6.58 on the first day without adaptation (solid lines) IoT devices use high task values and on the second day due to low energy the devices can only use low task values. With adaptation (dotted lines) using our DTA algorithm, the IoT devices are able to use lower task values on day 1 and save energy to be used on day 2 though on day 2 the task values are lower due to the lower battery energy levels. If we consider the temperature and humidity tasks shown in Figs.6.59 and 6.60 since the PERMIT provided task values are not too high, the DTA algorithm can only do minimal task adaptation on day 1 however the energy saved from adapting the Image task can still ensure that all the tasks can be run on day 2. Although in this case there are

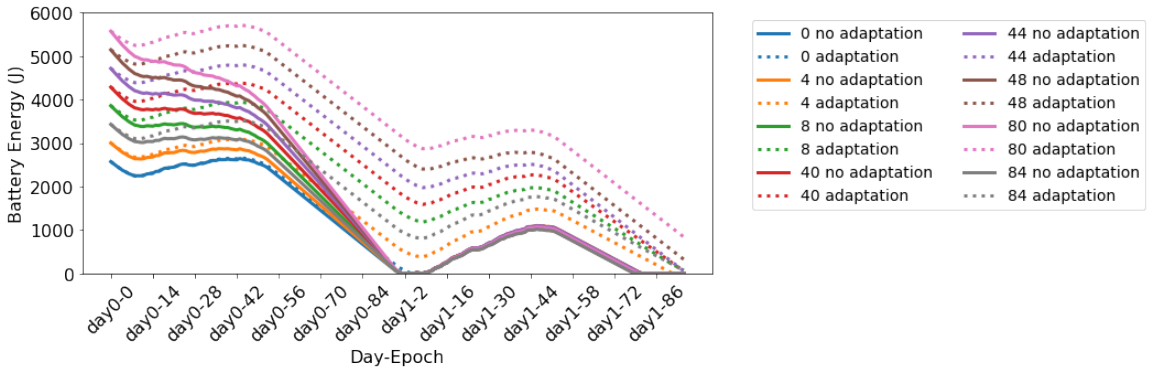

Figure 6.57: Uniform deployment base case battery energy

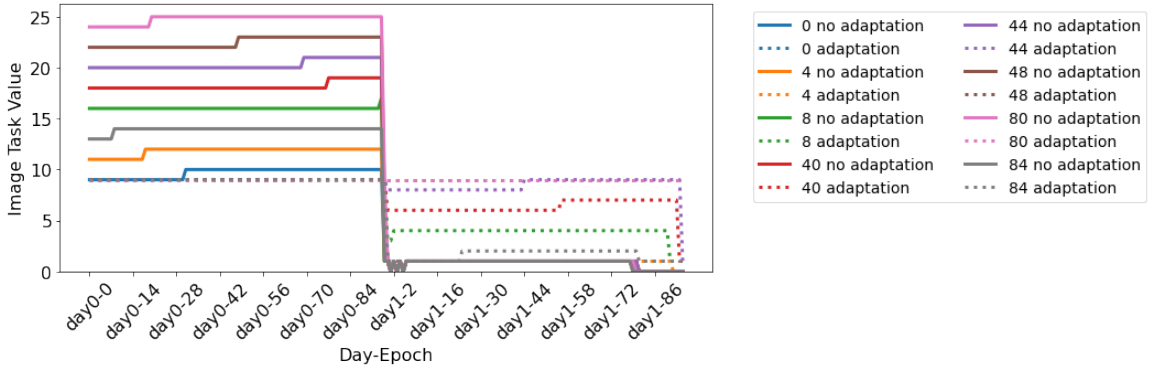

Figure 6.58: Uniform deployment base case image task values

limited benefits from cooperative sensing because there is limited overlap between nodes, the DTA algorithm is still able to adapt each IoT device's task value to meet the coverage threshold.

For the uniform deployment case, we compare the grid point coverage per grid point per epoch to evaluate whether we can achieve coverage equal to the required coverage threshold. The heat maps in Figs.6.62-6.67 illustrate the grid point coverage per grid point per epoch over the entire experiment and we compare the case using the Block Scheduler with and without adaptation. Comparing the grid point coverage in the image task

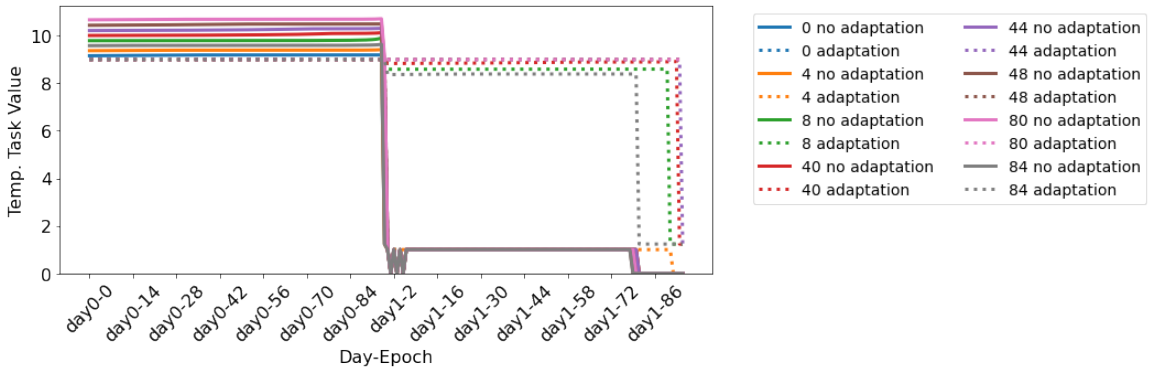

Figure 6.59: Uniform deployment base case temperature task values

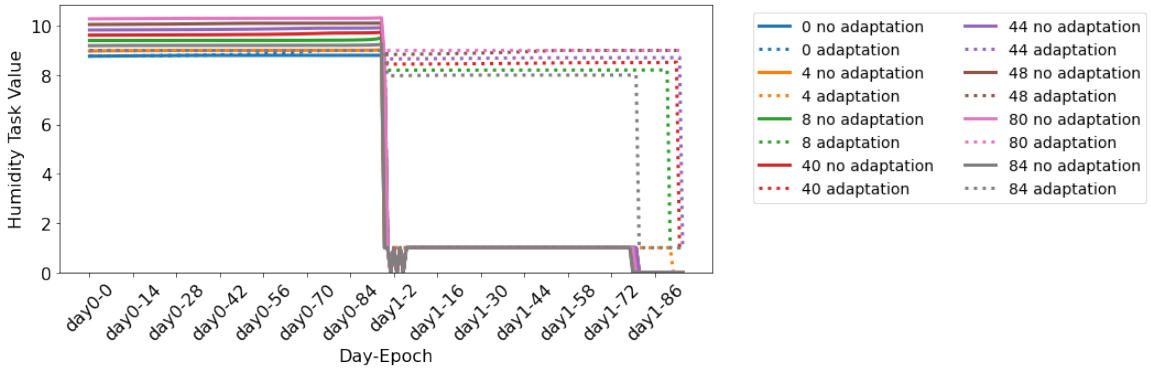

Figure 6.60: Uniform deployment base case humidity task values

case illustrates that without adaptation (Fig.6.62) we have a significant amount of over-coverage (green) on the first day and under-coverage (red) on the second day indicating we are wasting energy on the first day and providing insufficient information on the second day. However with adaptation (Fig.6.63) we significantly reduce both the over-coverage and under-coverage and ensure the coverage threshold is met. We see similar performance improvements with the temperature task (compare no adaptation in Fig.6.64 and adaptation in Fig.6.65)) and the humidity task (compare no adaptation in Fig.6.66 and adaptation in Fig.6.67). In both cases, our DTA algorithm reduces the over-coverage and under-coverage

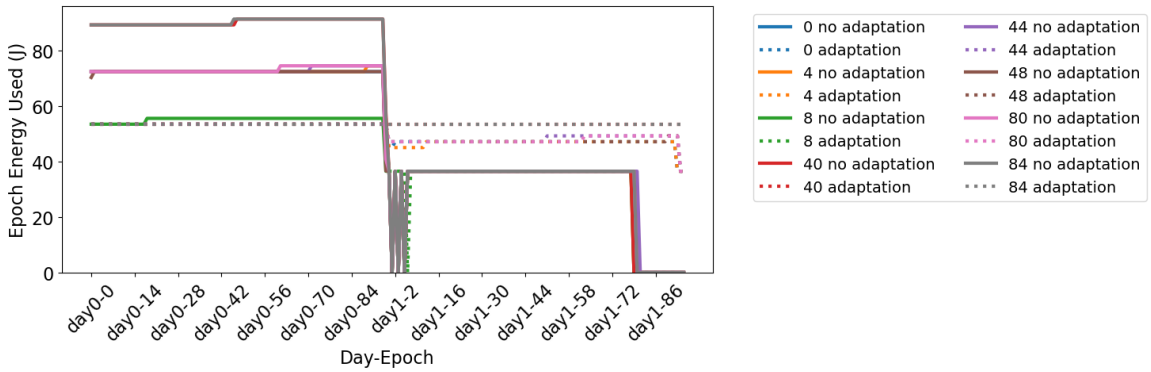

Figure 6.61: Uniform deployment base case epoch energy

significantly. Lastly, we compare the average overlap in the uniform deployment case to

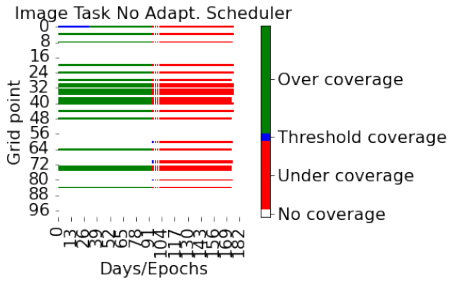

Figure 6.62: Grid point coverage: image task (no adaptation)

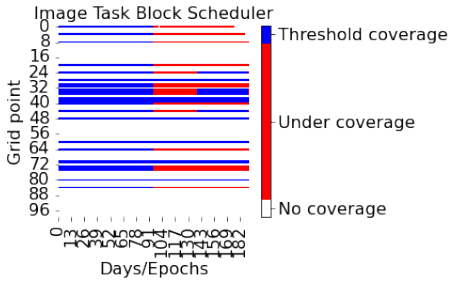

Figure 6.63: Grid point coverage: image task (with adaptation)

evaluate the performance of the different schedulers. Fig.6.68 shows that the Block Scheduler performs best with values close to the optimal, and the Random Scheduler performs worst. Fig.6.69 shows the benefits of task adaptation resulting in energy savings ranging from 0.16% to 10% per IoT device and averaged at 5% per IoT device.

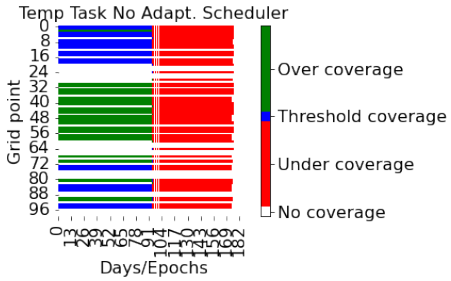

Figure 6.64: Grid point coverage: temp task (no adaptation)

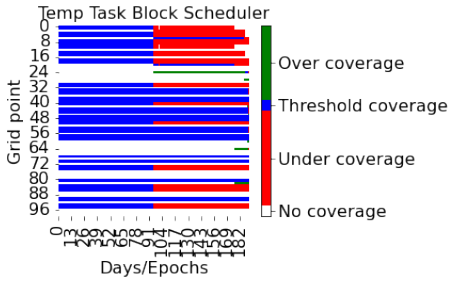

Figure 6.65: Grid point coverage: temp task (with adaptation)

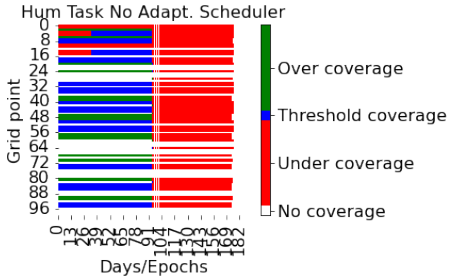

Figure 6.66: Grid point coverage: hum task (no adaptation)

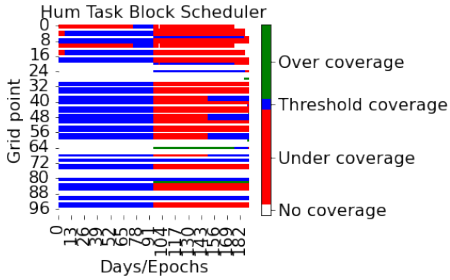

Figure 6.67: Grid point coverage: hum task (with adaptation)

6.7.2 Large Network Experiments

In our second set of experiments, we evaluate the performance of our algorithms over a larger network and under varying conditions. We vary parameters including the number of IoT devices, the weather, and the coverage threshold. Due to the size of the networks and runtime required for the Iterative Distributed Scheduler and Optimal NLP scheduler, we only compare the Block Scheduler with the Random and Alternate schedulers. The DTA algorithm is enabled in all cases and we use a non-uniform deployment pattern over a 100x100 grid with 100 sticks and $C_T = 9$.

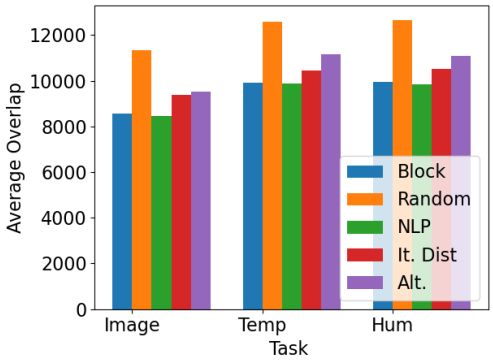

Figure 6.68: Uniform deployment base case average overlap.

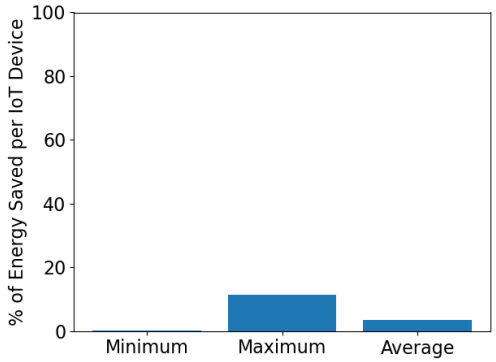

Figure 6.69: Uniform deployment base case average energy saved per device

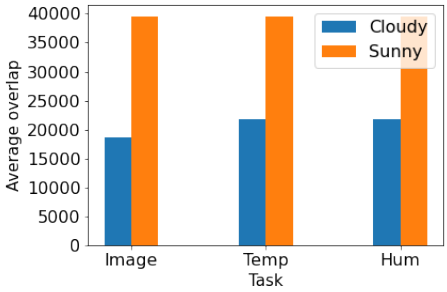

Figure 6.70: Large network average overlap when varying weather

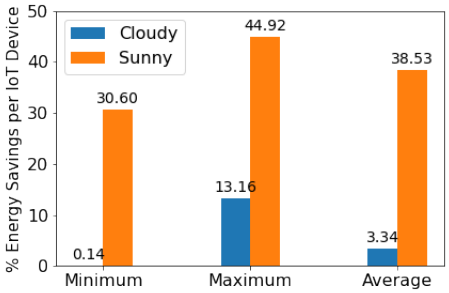

Figure 6.71: Large network average energy saved per device when varying weather

Varying Weather

We first compare the performance of the Block Scheduler and DTA during cloudy and sunny weather. From Fig.6.70 the average overlap on Sunny days is much larger than on cloudy days. This is because although task adaptation is being done, a larger number of IoT devices have higher task values even on the second day. DTA can adapt the task values downwards but because of the non-uniform deployment pattern where some sticks may share only some of their grid points with neighboring sticks, the potential for over-coverage is high especially when DTA tries to fix gaps in coverage during the second token

run. In contrast on a cloudy day, many IoT devices (especially on the second day) have lower task values and hence cannot benefit as much from adaptation.

Fig.6.71 shows the percentage of energy savings per device and we see that the energy saved on Sunny days is higher than on cloudy days because of the higher task values that are possible on Sunny days. On average, the per-device energy savings due to adaptation on a cloudy day adaptation was 3.34% vs. 38.53% on a sunny day. However, on a sunny day, we saw energy savings of up to 44.92% at an IoT device.

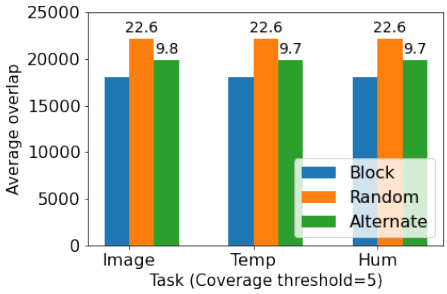

Figure 6.72: Large network C_T=5

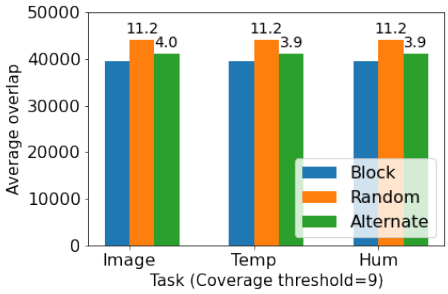

Figure 6.73: Large network C_T=9

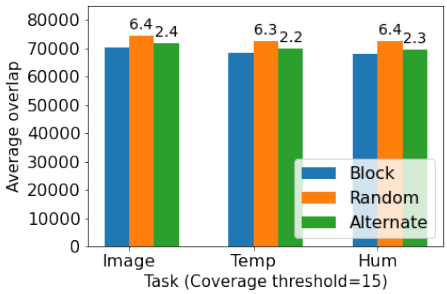

Figure 6.74: Large network C_T=15

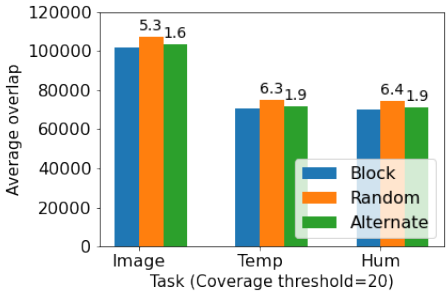

Figure 6.75: Large network C_T=20

Varying Coverage Threshold

We varied the coverage threshold value to compare the performance of the Block, Random, and Alternate Schedulers. The results are shown in Figs.6.72 to 6.75 for a Sunny day. The results show that even when we vary the coverage threshold, the Block Scheduler always performs best followed by the Alternate Scheduler then Random Scheduler. We also note that as the coverage threshold C_T increases, the average overlap also increases from less than 25,000 at $C_T = 5$ to 120,000 at $C_T = 20$. The results also show that as C_T increases the percentage difference in average overlap between the Block, Random, and Alternate Schedulers decreases. For example at $C_T = 5$ for the image task, the difference between the Random and Block Scheduler overlap is 22.6%, and this decreases to 11.2%, 6.4%, and 5.3% for $C_T = 9$, $C_T = 15$ and $C_T = 20$.

This is because as the coverage threshold C_T increases, more task executions (sensing operations) are needed to satisfy the coverage threshold. Scheduling with high task values then limits how much we can reduce the temporal overlap and the schedulers tend towards the same average overlap. The average image overlap at $C_T = 20$ is much higher than the temperature and image tasks because the image execution frequency is higher i.e. it has a maximum of 30 executions per epoch compared with only 15 for the temperature and humidity tasks.

The results from our experiments demonstrate that the Block Scheduler and the DTA algorithm perform well in both large and small networks.

6.8 Conclusion

Multi-sensor IoT devices are used to monitor different environmental phenomena however these devices depend on batteries and renewable energy sources for power. Therefore, efficient energy management solutions are needed to maximize device lifetime while ensuring appropriate and sufficient data is captured to be processed by the IoT application.

We leverage cooperative sensing to increase energy efficiency and minimize temporal overlap. Cooperative sensing allows multiple IoT devices to collaborate and coordinate their sensing operations to obtain the required data while minimizing energy use. Our Distributed Task Adaptation (DTA) algorithm empowers devices to modify their sensing task operations using information from neighboring devices. DTA works with our distributed Block Scheduler that minimizes overlap across all executions of a sensing task among neighboring devices by reframing the scheduling problem as a block placement problem and minimizing total block overlap every epoch.

Through simulations, we have assessed the performance of our DTA algorithm, revealing that in small networks operating under both sunny and cloudy weather conditions, it achieved an average energy savings of 5% per IoT device. Notably, for some devices with overlapping coverage areas, we achieved remarkable energy savings exceeding 30%. In larger IoT networks, our DTA algorithm exhibited an average energy savings of 3.34% on cloudy days and up to 38.53% on sunny days. These findings underscore its effectiveness across diverse environmental conditions. Furthermore, in various scenarios, our Block scheduler consistently demonstrated performance closely aligned with the optimal solution, showcasing its capability to efficiently reduce temporal overlap during scheduling. In summary, our

experimental results highlight the efficacy of both the DTA algorithm and the Block scheduler in conserving energy and mitigating temporal overlap among IoT devices, all while utilizing minimal information sharing through tokens.

Chapter 7

Conclusions

Multi-sensor IoT devices have revolutionized the monitoring of various phenomena by consolidating multiple functions into a single device. However, these devices heavily rely on batteries and renewable energy sources for power. Therefore efficient energy management solutions that maximize device lifetime and information utility and which can adapt to changes in the environment are important.

We performed a comprehensive analysis of the performance of four application layer protocols, particularly in scenarios with network impairments and timers play a key role in performance. These timers determine how long the node should wait before giving up on expecting a response and/or determine when the protocol can proceed to the next request. In the case of CoAP non-confirmable, we recommend an adaptive timer or a well-defined 'cookbook' of how applications should set the timer. We also delved into the experimental exploration of the relationship between the inclusion of additional protocol features and the corresponding energy consumption in IoT devices. The results of our

experiments underscore the substantial energy cost incurred by feature augmentation and interestingly, at shorter distances, the energy consumption associated with TCP and UDP transports displays remarkable similarity. This observation prompts the consideration of increasing packet sizes rather than multiplying the number of messages in an IoT environment. However, caution must be exercised when introducing new features within a single layer, as this can potentially lead to unforeseen cross-layer energy effects. Therefore, an intricate balance must be struck, weighing the energy expended against the performance optimization achieved through a more intricate protocol stack.

Drawing from these findings, we propose two strategies aimed at extending the operational lifespan of the IoT device. First, we recommend adapting the workload of the IoT device according to the available energy resources. This approach ensures an extended device lifespan while simultaneously fulfilling the application's information demands. Second, we advocate leveraging the power of cooperative sensing, where multiple sensors collaborate to further increase energy efficiency and limit temporal overlap.

This work presented PERMIT, our Predictive Energy Management solution for IoT, an adaptive and efficient energy management solution for IoT that addresses our first strategy. PERMIT uses the current battery state, predicted available solar energy, task energy, and utility models with an MPC-based approach to dynamically determine appropriate task values that maximize information utility while ensuring device energy goals are met. We also propose distributed PERMIT, an efficient MPC approximation approach that is simple enough to be run on the IoT device itself. The distributed PERMIT decomposes the original MPC optimization problem into an energy allocation problem (splitting available

energy into all epochs) and a sensor scheduling problem (dividing the epoch energy between all available tasks). Experimental results show that both the centralized and distributed PERMIT adapt the task values based on the available energy, and that the distributed PERMIT solution has very similar performance to the centralized PERMIT.

Our second strategy leverages cooperative sensing through the use of a distributed scheduler to manage sensing across multiple devices. We first propose DTTS protocol (Distributed Token and Tier-based task scheduling protocol) an energy-efficient distributed scheduler for an IoT network. DTTS finds a task's start time deadline and schedules tasks with shorter deadlines in earlier tiers in a distributed manner and with minimal information shared between IoT devices. Experiments showed DTTS always schedules an IoT device's start time before its deadline expires although it only considers the start time of the first task execution. To address this limitation, we extend this work and propose our distributed "Block scheduler". This uses a token-based approach to share minimal information across IoT devices while minimizing overlap across all executions of a sensing task among neighboring devices. Lastly, we further increase energy efficiency by leveraging cooperative sensing to dynamically adapt the task sensing operations at each IoT device by using our Distributed Task Adaptation (DTA) algorithm. This algorithm empowers devices to modify their sensing task operations using information from neighboring devices. Through simulations, we have assessed the performance of our DTA algorithm and Block Scheduler. In small networks operating under both sunny and cloudy weather conditions, our DTA algorithm achieved an average energy savings of 5% per IoT device. Notably, for some devices with overlapping coverage areas, we achieved remarkable energy savings

exceeding 30%. In larger IoT networks, our DTA algorithm exhibited an average energy savings of 3.34% on cloudy days and up to 38.53% on sunny days. These findings underscore the effectiveness of DTA across diverse environmental conditions. Furthermore, in various scenarios, our Block scheduler consistently demonstrated performance closely aligned with the optimal solution, showcasing its capability to efficiently reduce temporal overlap during scheduling. In summary, our experimental results highlight the efficacy of both the DTA algorithm and the Block scheduler in conserving energy and mitigating temporal overlap among IoT devices, all while utilizing minimal information sharing through tokens.

Bibliography

[1] E. Liri, P. K. Singh, A. B. Rabiah, K. Kar, K. Makhijani, and K. K. Ramakrishnan, "Robustness of iot application protocols to network impairments," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2018, pp. 97–103.

[2] E. Liri, N. Orie, H. Siezar, K. Ramakrishnan, P. K. Singh, W. Cai, L. Gerday, K. Kar, G. Lyon, and P. Sharma, "Communication energy and protocol considerations in iot deployments," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–7.

[3] E. Liri, K. K. Ramakrishnan, K. Kar, G. Lyon, and P. Sharma, "Invited paper: An efficient energy management solution for renewable energy based iot devices," in *Proceedings of the 24th International Conference on Distributed Computing and Networking*, ser. ICDCN '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 20–27. [Online]. Available: https://doi.org/10.1145/3571306.3571387

[4] E. Liri, K. Ramakrishnan, and K. Koushik, "A renewable energy-aware distributed task scheduler for multi-sensor iot networks," in *Proceedings of the ACM SIGCOMM Workshop on Networked Sensing Systems for a Sustainable Society*, 2022, pp. 26–32.

[5] E. Liri, K. Ramakrishnan, and K. Kar, "Energy-efficient distributed task scheduling for multi-sensor iot networks," *IEEE Network*, vol. 37, no. 2, pp. 318–324, 2023.

[6] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer *et al.*, "Internet of things strategic research roadmap," *Internet of Things-Global Technological and Societal Trends*, vol. 1, no. 2011, pp. 9–52, 2011.

[7] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *IEEE communications surveys & tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[8] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[9] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[10] P. López, D. Fernández, A. J. Jara, and A. F. Skarmeta, "Survey of internet of things technologies for clinical environments," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on.* IEEE, 2013, pp. 1349–1354.

[11] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on.* IEEE, 2012, pp. 257–260.

[12] D.-L. Y. F. L. Yi and D. Liang, "A survey of the internet of things," *Proc. of ICEBI*, 2010.

[13] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[14] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on.* IEEE, 2014, pp. 1–6.

[15] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *Computing, Networking and Communications (ICNC), 2013 International Conference on.* IEEE, 2013, pp. 334–340.

[16] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on.* IEEE, 2013, pp. 1–6.

[17] C. Bormann, K. Hartke, and Z. Shelby, "The constrained application protocol (coap)," *RFC 7252*, 2015.

[18] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—a publish/subscribe protocol for wireless sensor networks," in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on.* IEEE, 2008, pp. 791–798.

[19] A. Stanford-Clark and H. L. Truong, "Mqtt for sensor networks (mqtt-sn) protocol specification," *International business machines (IBM) Corporation version*, vol. 1, 2013.

[20] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-transport-11, 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-11

[21] M. Richardson and S. Wallace, *Getting started with raspberry PI. "* O'Reilly Media, Inc.", 2012.

[22] Debian Wiki, "TrafficControl." [Online]. Available: URL:https://wiki.debian.org/TrafficControl

[23] O. Bergmann, "libcoap: C-implementation of coap," *URL: http://libcoap. sourceforge. net, Date of access 13.09*, 2012.

[24] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, 2017.

[25] T. E. Foundation, "Eclipse Paho - MQTT and MQTT-SN software." [Online]. Available: https://www.eclipse.org/paho/

[26] T. C. Projects, "QUIC," accessed 2023-09-06. [Online]. Available: https://www.chromium.org/quic/playing-with-quic

[27] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[28] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer Networks*, vol. 67, pp. 104 – 122, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614001418

[29] H. Liu, A. Chandra, and J. Srivastava, "eSENSE: Energy Efficient Stochastic Sensing Framework Scheme for Wireless Sensor Platforms," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, ser. IPSN '06. New York, NY, USA: ACM, 2006, pp. 235–242. [Online]. Available: http://doi.acm.org/10.1145/1127777.1127815

[30] S. Ganeriwal, I. Tsigkogiannis, H. Shim, V. Tsiatsis, M. B. Srivastava, and D. Ganesan, "Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 843–856, June 2009.

[31] L. Mendes and J. Rodrigues, "A survey on cross-layer solutions for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 34, pp. 523–534, 03 2011.

[32] C. Bormann, s. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "Coap (constrained application protocol) over tcp, tls, and websockets," *RFC 8323*, 02 2018. [Online]. Available: https://tools.ietf.org/html/rfc8323

[33] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. [Online]. Available: https://tools.ietf.org/html/rfc5246

[34] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. [Online]. Available: https://tools.ietf.org/html/rfc6347

[35] T. A. Alghamdi, A. Lasebae, and M. Aiash, "Security Analysis of the Constrained Application Protocol in the Internet of Things," in *Second International Conference on Future Generation Communication Technologies (FGCT 2013)*. IEEE, 2013, pp. 163–168.

[36] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient Communication Protocol for Wireless Microsensor Networks," 2000, pp. 3005–3014.

[37] A. Liu, X. Jin, G. Cui, and Z. Chen, "Deployment Guidelines for Achieving Maximum Lifetime and Avoiding Energy Holes in Sensor Network," *Information Sciences*, vol. 230, pp. 197 – 226, 2013, Mobile and Internet Services in Ubiquitous and Pervasive Computing Environments. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025513000297

[38] B. Martinez, M. MontÃ³n, I. Vilajosana, and J. D. Prades, "The Power of Models: Modeling Power Consumption for IoT Devices," *IEEE Sensors Journal*, vol. 15, no. 10, pp. 5777–5789, Oct 2015.

[39] F. Kaup, P. Gottschling, and D. Hausheer, "PowerPi: Measuring and Modeling the Power Consumption of the Raspberry Pi," in *39th Annual IEEE Conference on Local Computer Networks*, Sep. 2014, pp. 236–243.

[40] A. Garcia-Saavedra, P. Serrano, A. Banchs, and G. Bianchi, "Energy consumption anatomy of 802.11 devices and its implication on modeling and design," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 169–180. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413197

[41] P. Serrano, A. Garcia-Saavedra, G. Bianchi, A. Banchs, and A. Azcorra, "Per-frame energy consumption in 802.11 devices and its implication on modeling and design," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1243–1256, Aug. 2015. [Online]. Available: https://doi.org/10.1109/TNET.2014.2322262

[42] H. T. Friis, "A note on a simple transmission formula," *Proc. IRE*, vol. 34, no. 5, pp. 254–256, 1946.

[43] M. Hata, "Empirical formula for propagation loss in land mobile radio services," *IEEE Trans. Veh. Tech.*, vol. 29, no. 3, pp. 317–325, 1980.

[44] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. B. Zikria, "Role of iot technology in agriculture: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 319, Feb 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/2/319

[45] F. K. Shaikh and S. Zeadally, "Energy harvesting in wireless sensor networks: A comprehensive review," *Renewable and Sustainable Energy Reviews*, vol. 55, pp. 1041–1054, 2016.

[46] M.-L. Ku, W. Li, Y. Chen, and K. J. Ray Liu, "Advances in energy harvesting communications: Past, present, and future challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1384–1412, 2016.

[47] C. Bergonzini, D. Brunelli, and L. Benini, "Algorithms for harvested energy prediction in batteryless wireless sensor networks," in *2009 3rd International workshop on advances in sensors and interfaces.* Trani,Italy: IEEE, 2009, pp. 144–149.

[48] Q. Ju and Y. Zhang, "Predictive power management for internet of battery-less things," *IEEE Transactions on Power Electronics*, vol. 33, no. 1, pp. 299–312, 2018.

[49] A. Caruso, S. Chessa, S. Escolar, X. del Toro, and J. C. López, "A dynamic programming algorithm for high-level task scheduling in energy harvesting iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2234–2248, 2018.

[50] A. Kansal, J. Hsu, M. Srivastava, and V. Raghunathan, "Harvesting aware power management for sensor networks," in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 651–656. [Online]. Available: https://doi.org/10.1145/1146909.1147075

[51] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: An iot platform for data-driven agriculture," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).* Boston, MA: USENIX Association, Mar. 2017, pp. 515–529.

[52] A. G. Mohapatra and S. K. Lenka, "Neural network pattern classification and weather dependent fuzzy logic model for irrigation control in wsn based precision agriculture," *Procedia Computer Science*, vol. 78, pp. 499–506, 2016.

[53] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, IEEE. Porto,Portugal: IEEE, 2018, pp. 188–199.

[54] P. Krupa, D. Limon, and T. Alamo, "Implementation of model predictive control in programmable logic controllers," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1117–1130, 2021.

[55] R. Carli, G. Cavone, S. Ben Othman, and M. Dotoli, "Iot based architecture for model predictive control of hvac systems in smart buildings," *Sensors*, vol. 20, no. 3, p. 781, 2020.

[56] H. Ren and K.-W. Chin, "Novel tasks assignment methods for wireless powered iot networks," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10 563–10 575, 2021.

[57] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice - a survey," *Automatica*, vol. 25, no. 3, pp. 335 – 348, 1989.

[58] T. Instruments. (2015, Feb) Characteristics of rechargeable batteries. Texas Instruments. [Online]. Available: https://www.ti.com/lit/an/snva533/snva533.pdf

[59] Battery University. (2021, Oct) Bu-501a: Discharge characteristics of li-ion. Battery University. [Online]. Available: https://batteryuniversity.com/article/bu-501a-discharge-characteristics-of-li-ion

[60] Digikey. (2021, Sept 2016) A designer's guide to lithium (li-ion) battery charging. Digikey. [Online]. Available: https://www.digikey.com/en/articles/a-designer-guide-fast-lithium-ion-battery-charging

[61] H. Luss, "On equitable resource allocation problems: A lexicographic minimax approach," *Operations Research*, vol. 47, no. 3, pp. 361–378, 1999.

[62] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.)*. USA: Prentice-Hall, Inc., 1992.

[63] T. Vidal, D. Gribel, and P. Jaillet, "Separable convex optimization with nested lower and upper constraints," *INFORMS Journal on Optimization*, vol. 1, no. 1, pp. 71–90, 2019.

[64] P. T. Akhil and R. Sundaresan, "Algorithms for separable convex optimization with linear ascending constraints," *Sādhanā*, vol. 43, no. 9, p. 146, 2018.

[65] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 4, p. 32–es, Sep. 2007. [Online]. Available: https://doi.org/10.1145/1274858.1274870

[66] J. Recas Piorno, C. Bergonzini, D. Atienza, and T. Simunic Rosing, "Prediction and management in energy harvested wireless sensor nodes," in *2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology*, 2009, pp. 6–10.

[67] A. Cammarano, C. Petrioli, and D. Spenza, "Pro-energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks," in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*. IEEE, 2012, pp. 75–83.

[68] S. Kosunalp, "A new energy prediction algorithm for energy-harvesting wireless sensor networks with q-learning," *IEEE Access*, vol. 4, pp. 5755–5763, 2016.

[69] R. Pi. (2021, Oct) Raspberry pi zero w. RaspberryPi. [Online]. Available: https://https://www.raspberrypi.com/products/raspberry-pi-zero-w/

[70] Adafruit. (2021, Oct) Huge 6v 6w solar panel - 6.0 watt. Adafruit. [Online]. Available: https://www.adafruit.com/product/1525

[71] ——. (2021, Oct) Lithium ion battery. Adafruit. [Online]. Available: https://www.adafruit.com/product/354

[72] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan, "ASAP: Adaptive stall-aware pacing for improved dash video experience in cellular networks," *ACM Transactions on Multimedia Computing, Communications and Applications (TOMM)*, vol. 14, no. 3s, p. 61, 2018.

[73] T. L. Saaty, "A scaling method for priorities in hierarchical structures," *Journal of Mathematical Psychology*, vol. 15, no. 3, pp. 234 – 281, 1977.

[74] D. Andrea. (2009, Oct) White paper - estimating the state of charge of li-ion batteries. Elithion. [Online]. Available: http://liionbms.com/php/wp_soc_estimate.php

[75] N. Sinenian and D. Shai, "3 - advances in power converters," in *The Power Grid*, B. W. D'Andrade, Ed. San Diego, California: Academic Press, 2017, pp. 57–92. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128053218000033

[76] S. Wilcox, "National solar radiation database 1991-2005 update: User's manual," National Renewable Energy Lab.(NREL), Golden, CO (United States), Tech. Rep., 2007.

[77] Lab11. (2018, Oct) Signpost. University of California Berkeley, Lab11. [Online]. Available: https://github.com/lab11/signpost/blob/master/test/solar_irradiance_model/scripts/irradiance_to_signpost.py

[78] M. M. Sandhu, S. Khalifa, R. Jurdak, and M. Portmann, "Task scheduling for energy-harvesting-based iot: A survey and critical analysis," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 825–13 848, 2021.

[79] Y. Tan and X. Yin, "A dynamic scheduling algorithm for energy harvesting embedded systems," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, pp. 1–8, 2016.

[80] T. Zhu, A. Mohaisen, Y. Ping, and D. Towsley, "Deos: Dynamic energy-oriented scheduling for sustainable wireless sensor networks," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2363–2371.

[81] D. Gao, Q. Sun, B. Hu, and S. Zhang, "A framework for agricultural pest and disease monitoring based on internet-of-things and unmanned aerial vehicles," *Sensors*, vol. 20, no. 5, p. 1487, 2020.

[82] N. Ahmed, D. De, and I. Hussain, "Internet of things (iot) for smart precision agriculture and farming in rural areas," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4890–4899, 2018.

[83] M. R. M. Kassim, I. Mat, and I. M. Yusoff, "Applications of internet of things in mushroom farm management," in *2019 13th International Conference on Sensing Technology (ICST)*, 2019, pp. 1–6.

[84] R. Hartung, U. Kulau, B. Gernert, S. Rottmann, and L. Wolf, "On the experiences with testbeds and applications in precision farming," in *Proceedings of the first ACM international workshop on the engineering of reliable, robust, and secure embedded wireless sensing systems*, 2017, pp. 54–61.

[85] M. A. Zamora-Izquierdo, J. Santa, J. A. Martínez, V. Martínez, and A. F. Skarmeta, "Smart farming iot platform based on edge and cloud computing," *Biosystems engineering*, vol. 177, pp. 4–17, 2019.

[86] L. Hang and D.-H. Kim, "Enhanced model-based predictive control system based on fuzzy logic for maintaining thermal comfort in iot smart space," *Applied Sciences*, vol. 8, no. 7, p. 1031, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/7/1031

[87] Z. Zhang, J. Wu, Y. Zhao, and R. Luo, "Research on distributed multi-sensor cooperative scheduling model based on partially observable markov decision process," *Sensors*, vol. 22, no. 8, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/3001

[88] S. R. Sarangi, S. Goel, and B. Singh, "Energy efficient scheduling in iot networks," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 733–740. [Online]. Available: https://doi.org/10.1145/3167132.3167213

[89] C. Moser, J.-J. Chen, and L. Thiele, "Dynamic power management in environmentally powered systems," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 81–88.

[90] M. De Benedetti, F. Messina, G. Pappalardo, and C. Santoro, "Jarvsis: a distributed scheduler for iot applications," *Cluster Computing*, vol. 20, no. 2, pp. 1775–1790, 2017.

[91] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[92] M. Afaneh. (2020) Wireless connectivity options for iot applications - technology comparison. [Online]. Available: https://www.bluetooth.com/blog/wireless-connectivity-options-for-iot-applications-technology-comparison/

[93] Which wireless standard makes sense for your application? 2020, last accessed: Feb 01 2022, https://www.volersystems.com/guide-wireless-technology.

[94] A. N. S. Institute, *Local Area Networks: Token Ring Access Method and Physical Layer Specifications–802.5*. USA: John Wiley & Sons, Inc., 1985.

[95] M. S. Kingley. (1996, Feb.) Ansi fiber distributed data interface (fddi) standards. [Online]. Available: http://www.bitsavers.org/pdf/datapro/communications_standards/2715_FDDI.pdf

[96] H. Yang and K. Ramakrishnan, "A ring purger for the fddi token ring," in *[1991] Proceedings 16th Conference on Local Computer Networks*. IEEE Computer Society, 1991, pp. 503–504.

[97] M. J. Johnson, "Reliability mechanisms of the fddi high bandwidth token ring protocol," *Computer Networks and ISDN Systems*, vol. 11, no. 2, pp. 121–131, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0169755286900127

[98] C. Delgado and J. Famaey, "Optimal energy-aware task scheduling for batteryless iot devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1374–1387, 2021.

[99] W. Bux, "Token-ring local-area networks and their performance," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 238–256, 1989.

[100] F. Ross, "An overview of fddi: the fiber distributed data interface," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1043–1051, 1989.

[101] N. Kushalnagar, G. Montenegro, and C. Schumacher, "Ipv6 over low-power wireless personal area networks (6lowpans): overview, assumptions, problem statement, and goals," IETF, Tech. Rep., 2007.

[102] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7252

[103] A. Banks and R. Gupta, "Mqtt version 3.1. 1," *OASIS standard*, vol. 29, 2014.

[104] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[105] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[106] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[107] B. Networks. Servers/brokers. [Online]. Available: https://github.com/mqtt/mqtt.github.io/wiki/servers

[108] C. Bormann. Quic: A udp-based multiplexed and secure transport. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-transport-10page-17

[109] J. Iyengar and M. Thomson. Implementations. [Online]. Available: http://coap.technology/impls.html

[110] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)," RFC 7641, Sep. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7641.txt

[111] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[112] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 609–614.

[113] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 183–196.

[114] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport," *draft-ietf-quic-transport-01 (work in progress)*, 2017.

[115] Y. Okumura, E. Ohmori, T. Kawano, and K. Fukuda, "Field strength and its variability in vhf and uhf land-mobile service," *Rev. Elec. Comm. Lab.*, vol. 16, no. 9-10, pp. 825–873, 1968.

[116] R. Foundation, "RaspberryPi Foundation," 2018. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837b0/README.md

[117] Espressif, "Espressif Systems ESP32-DevKitC." [Online]. Available: https://www.espressif.com/en/products/hardware/esp32-devkitc/overview

[118] AVZHY. (2019, Mar.) AVHzY CT-2 USB Power Meter Load Tester Voltage Detector. [Online]. Available: https://store.avhzy.com

[119] M. Younis and K. Akkaya, "Strategies and techniques for node placement in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 6, no. 4, pp. 621 – 655, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870507000984

[120] Z. Abbas and W. Yoon, "A Survey on Energy Conserving Mechanisms for the Internet of Things: Wireless Networking Aspects," *Sensors*, vol. 15, no. 10, pp. 24 818–24 847, 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/10/24818

[121] Q. Wang, M. Hempstead, and W. Yang, "A Realistic Power Consumption Model for Wireless Sensor Network Devices," in *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 1, Sep. 2006, pp. 286–295.

[122] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "Mqtt version 5.0," *OASIS Standard*, 03 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[123] O. Bergmann, "TinyDTLS Software Library Implementation," 01 2017. [Online]. Available: https://projects.eclipse.org/projects/iot.tinydtls

[124] C. Bisdikian, L. M. Kaplan, and M. B. Srivastava, "On the quality and value of information in sensor networks," *ACM Trans. Sen. Netw.*, vol. 9, no. 4, Jul. 2013. [Online]. Available: https://doi.org/10.1145/2489253.2489265

[125] T. L. Saaty, "Hard mathematics applied to soft decisions," in *Indonesian Symposium Analytic Hierarchy Process II Teknik Industri Universitas Kristen Petra Surabaya, Tidak Dipublikasikan, Surabaya: Universitas Kristen Petra*, 2002.

[126] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang, "A scheduling framework for adaptive video delivery over cellular networks," in *Proceedings of the 19th Annual International Conference on Mobile Computing &; Networking*, ser. MobiCom '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 389–400. [Online]. Available: https://doi.org/10.1145/2500423.2500433

[127] M. S. Mendez G.R., "A wi-fi based smart wireless sensor network for an agricultural environment," in *Wireless Sensor Networks and Ecological Monitoring*, M. Smart Sensors and Instrumentation, Eds. Berlin, Heidelberg: Oxford University Press, 2013, ch. 10, pp. 247–268.

[128] R. P. Foundation. (2022) Raspbery pi camera module. [Online]. Available: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md

[129] D. Jones. (2016) Camera hardware. [Online]. Available: https://picamera.readthedocs.io/en/release-1.13/fov.html

[130] T. Saaty, "Relative measurement and its generalization in decision making why pairwise comparisons are central in mathematics for the measurement of intangible factors the analytic hierarchy/network process." *Rev. R. Acad. Cien. Serie A. Mat*, vol. 102, pp. 251–318, 2008.

[131] I. Marsic, *Computer Networks Performance and Quality of Service*. Rutgers University, 2013.

[132] Y.-L. Huang and W.-L. Sun, "An ahp-based risk assessment for an industrial iot cloud," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. Lisbon, Portugal: IEEE, 2018, pp. 637–638.

[133] J. Wang, W.-C. Yeh, N. N. Xiong, J. Wang, X. He, and C.-L. Huang, "Building an improved internet of things smart sensor network based on a three-phase methodology," *IEEE Access*, vol. 7, pp. 141 728–141 737, 2019.

[134] P. T. M. Ly, W.-H. Lai, C.-W. Hsu, and F.-Y. Shih, "Fuzzy ahp analysis of internet of things (iot) in enterprises," *Technological Forecasting and Social Change*, vol. 136, pp. 1–13, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162518307030

[135] D. Davcev, K. Mitreski, S. Trajkovic, V. Nikolovski, and N. Koteli, "Iot agriculture system based on lorawan," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. Imperia, Italy: IEEE, 2018, pp. 1–4.

[136] N. Gondchawar1 and D. R. S. Kawitkar, "Iot based smart agriculture," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, pp. 838–842, 2016.

[137] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. B. Zikria, "Role of iot technology in agriculture: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 319, Feb 2020. [Online]. Available: http://dx.doi.org/10.3390/electronics9020319

[138] P. Pawar, M. TarunKumar, and P. Vittal K., "An iot based intelligent smart energy management system with accurate forecasting and load strategy for renewable generation," *Measurement*, vol. 152, p. 107187, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026322411931053X

[139] F. A. Kraemer, D. Ammar, A. E. Braten, N. Tamkittikhun, and D. Palma, "Solar energy prediction for constrained iot nodes based on public weather forecasts," *Proceedings of the Seventh International Conference on the Internet of Things IoT '17*, vol. 2, pp. 1–8, 2017. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3131542.3131544

[140] V. Pop, H. J. Bergveld, P. H. L. Notten, and P. P. L. Regtien, "State-of-the-art of battery state-of-charge determination," *Measurement Science and Technology*, vol. 16, no. 12, pp. R93–R110, oct 2005. [Online]. Available: https://doi.org/10.1088/0957-0233/16/12/r01

[141] J. Adkins, B. Ghena, N. Jackson, P. Pannuto, S. Rohrer, B. Campbell, and P. Dutta, "The signpost platform for city-scale sensing," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2018, pp. 188–199.

[142] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[143] B. Y. Liu and R. C. Jordan, "The long-term average performance of flat-plate solar-energy collectors: with design data for the us, its outlying possessions and canada," *Solar energy*, vol. 7, no. 2, pp. 53–74, 1963.

[144] Lab11. (2018, Oct) Signpost software. Berkeley. [Online]. Available: https://github.com/lab11/signpost-software

[145] U. S. Diego. (20122) High performance wireless research and education network (hpwren. UC San Diego. [Online]. Available: http://hpwren.ucsd.edu/cameras

[146] A. Wildfire. (2022) Alert wildfire. University of Nevada. [Online]. Available: http://www.alertwildfire.org

[147] N. R. E. L. (NREL). (2021, Oct) Nsrdb data viewer. US Department of Energy. [Online]. Available: https://tinyurl.com/msz5uxt6

[148] E. Liri, K. K. Ramakrishnan, K. Kar, G. Lyon, and P. Sharma. (2022, Oct) Appendix: An efficient energy management solution for renewable energy based iot devices. [Online]. Available: https://tinyurl.com/2p942x6m

[149] A. Valente, S. Silva, D. Duarte, F. Cabral Pinto, and S. Soares, "Low-cost lorawan node for agro-intelligence iot," *Electronics*, vol. 9, no. 6, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/6/987

[150] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.)*. USA: Prentice-Hall, Inc., 1992.

[151] ——, *Data Networks (2nd Ed.)*. USA: Prentice-Hall, Inc., 1992.

[152] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 4, pp. 397–413, 1993.

[153] ——, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 4, pp. 397–413, 1993.

[154] X. Liu, Y. Gao, and F. Hu, "Optimal time scheduling scheme for wireless powered ambient backscatter communications in iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2264–2272, 2019.

[155] N.-T. Nguyen, N. Van Huynh, D. T. Hoang, D. N. Nguyen, N.-H. Nguyen, Q.-T. Nguyen, and E. Dutkiewicz, "Energy management and time scheduling for heterogeneous iot wireless-powered backscatter networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[156] H. Yang and K. K. Ramakrishnan, "Frame content independent stripping," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, p. 276–286, aug 1990. [Online]. Available: https://doi.org/10.1145/99517.99564

[157] K. K. Ramakrishnan, M. Yuksel, H. Seferoglu, J. Chen, and R. A. Blalock, "Resilient communication for dynamic first responder teams in disaster management," *IEEE Communications Magazine*, pp. 1–7, 2022.

[158] F. Bu and X. Wang, "A smart agriculture iot system based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 99, pp. 500–507, 2019.

[159] H. M. Jawad, R. Nordin, S. K. Gharghan, A. M. Jawad, and M. Ismail, "Energy-efficient wireless sensor networks for precision agriculture: A review," *Sensors*, vol. 17, no. 8, p. 1781, 2017.

Appendix A

Task Characterization and Utility

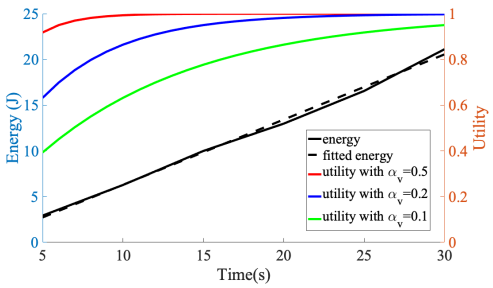

Figure A.1: Modelling video energy and utility with duration.

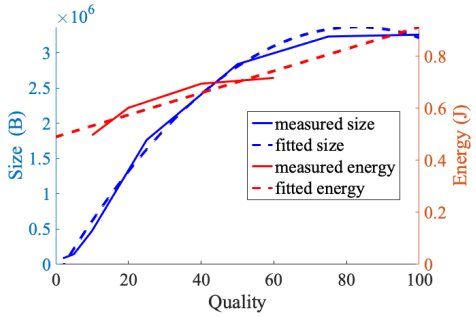

Figure A.2: Modelling image size and task energy with quality

Since we use a utility function $U_k = 1 - e^{-\alpha_k f_k}$, where f_k is the variable task parameter, α_k is a task-dependent utility constant determined through tuning such that even the lowest value of f_k can give a reasonable utility. Therefore the utility value at the lowest task value for all tasks is ∼60% of the maximum information utility. Fig. A.1 shows how information utility varies with different values of α_v for the video task. If α_v is too large (e.g., $\alpha_v = 0.5$), the information utility reaches the maximum very quickly, even for

small video durations. If α_v is too small, (e.g., $\alpha_v = 0.1$), the utility increases too slowly and is still low even at max video duration (30s). For video, we used $\alpha_v = 0.2$ which gives a reasonable increase in utility with increasing video duration.

In Fig. A.2 the solid red curve illustrates the linear relationship between the computation energy required to generate the image and the image quality and the dashed line is the fitted curve $E_{i,comp}$. Fig. A.2 also shows a quadratic relationship between image size s_i and quality q (see solid blue line).

Appendix B

Task Prioritization using AHP

Table B.1: Criteria for PERMIT AHP Ranking

Relevance	Freshness a (image/video) (mins)	Freshness a (monitoring) (mins)	Accuracy	AHP Score
Very	a$\leq$ 15	<5	3280x2646	9
-	$15 < a \leq 30$	$5 \leq a < 10$	1920x1080	8
-	$30 < a \leq 45$	$10 \leq a < 15$	1640x1232	7
-	$45 < a \leq 60$	$15 \leq a < 20$	1640x922	6
Relevant	$60 < a \leq 75$	$20 \leq a < 25$	1280x720	5
-	$75 < a \leq 90$	$25 \leq a < 30$	640x480	4
-	$90 < a \leq 105$	$30 \leq a < 35$	-	3
Equal	$105 < a \leq 120$	$35 \leq a < 40$	-	2
Not relevant	> 120	≥ 40	-	1

When a device has a limited amount of energy, it is important to use it judiciously. With PERMIT providing multiple types of sensor data, we seek to prioritize the most valuable data for a given context. We differentiate between the quality of data and the value of the data which depends on the context. For example, the environment's accurate temperature reading (high quality data) may have less value in an intrusion detection

application than even a low-resolution image (low quality data). We use the Analytic Hierarchy Process (AHP) to rate the different types of data the PERMIT Stick provides so that high-value data is appropriately prioritized. The Analytic Hierarchy Process [73] is a multi-criteria/attribute decision-making approach. It ranks different alternatives based on specified decision criteria and can be used for more complex decision-making because of its hierarchical nature. We use AHP because it requires limited computation, includes a mechanism to check for consistency, is well established and the relative scoring process can utilize both hard (objective) criteria as well as soft(subjective) criteria. A good example explaining how it works is in [125]. For the AHP process, the first step is understanding the pairwise comparison process and then the criteria used for the pairwise comparison.

B.1 Calculating the Task Priority

Pairwise Comparison AHP performs a pairwise comparison between two tasks and thus gives a relative score instead of an absolute score. When comparing task X vs Y, the relative score indicates how much more important or dominant task X is vs Y. AHP scores range from 1 (equally important) to 9 (very important) [130]. Determining these relative scores using pairwise comparisons depends on the user and is application dependent. Subjective pairwise comparisons for subjective criteria like Relevance depend purely on the domain expert. For objective comparisons using criteria like Freshness and Accuracy, instead of relying purely on the domain expert we use measurable data. We assumed a linear increase in the applicable range of objective values e.g. each age range for data can be mapped to an absolute AHP score in the range 1-9 and we generated Table B.1 by relying on domain

knowledge. More important objective values have a higher absolute AHP score, for example, fresher data has a higher absolute AHP score. For objective criteria, the relative AHP score is the ratio of the absolute AHP scores for the compared tasks.

For all pairwise comparisons, if the relative score for X vs Y is r then reversing the comparison i.e. Y vs X automatically has a relative score $1/r$. In this work, we consider a surveillance scenario that prioritizes video (V) and image (I) tasks. We assume the video task takes 1 video every 15mins at 1944x1080 resolution, the image task takes 1 image every 15 mins at 2592x1944 resolution and the temperature and humidity tasks take at least 1 reading every 15 mins. In this work, we use three criteria and these are Relevance, Freshness, and Accuracy.

Relevance Criteria The Relevance criteria indicates how relevant the results are to the current scenario. Domain knowledge is used to compare the relevance of different types of information in different scenarios. In the surveillance scenario, we decide that video data is very important compared to temperature data and give it a relative score of 9. Reversing the comparison automatically sets the relative AHP score of temperature to video data to 1/9. These 2 values are indicated in Table B.3 (video row, temperature column, first value and temperature row, video column, first value). This subjective pairwise comparison is also done for the criteria in order to be able to rank the criteria B.2.

Table B.2: Ranking Criteria for PERMIT AHP

	R	F	A
Relevance (R)	1	8	5
Freshness (F)	$\frac{1}{8}$	1	3
Accuracy (A)	$\frac{1}{5}$	$\frac{1}{3}$	1
Sum	1.325	9.333	9

Table B.3: Task relative AHP score per criteria (Relevance,Freshness,Accuracy))

	Video	Image	Temp.	Humidity
Video	1,1,1	$\frac{1}{9},\frac{9}{7},\frac{8}{9}$	$\frac{1}{3},\frac{9}{6},\frac{8}{9}$	$\frac{1}{3},\frac{9}{6},\frac{8}{9}$
Image	$9,\frac{7}{9},\frac{9}{8}$	1,1,1	$3,\frac{7}{6},1$	$3,\frac{7}{6},1$
Temp.	$6,\frac{6}{9},\frac{9}{8}$	$\frac{1}{3},\frac{6}{7},1$	1,1,1	1,1,1
Humidity	$6,\frac{6}{9},\frac{9}{8}$	$\frac{1}{3},\frac{6}{7},1$	1,1,1	1,1,1

Freshness Criteria The Freshness criteria considers how recent the results are and is objectively measured by the age of the data. For example, to determine the relative freshness AHP score when comparing a video that is 30 mins old (AHP score=8) and temperature data that is 29 mins old (AHP score=4) is $8/4 = 2$. A 2 in the AHP score range implies that according to the freshness criteria, 30 min old video data is only moderately more important than 29 min old temperature data. Reversing the comparison i.e. comparing the freshness of the 29 min temperature data and the 30 min video data gives a relative AHP score of 1/2.

Accuracy/Integrity Criteria The Accuracy (or Integrity) criteria refers to how satisfied we are with the level of accuracy of the results. Generally, this criteria can be objectively measured. In the case of temperature and humidity monitoring, the PERMIT Stick has only 1 accuracy level, so all monitoring measurements have an AHP of value 9. How ever

for the image and video tasks, we measure image and video accuracy using the resolution (see Table B.1).

Table B.4: Calculating Final Weights for Criteria

	Relevance	Freshness	Accuracy
Relevance	$\frac{1}{1.325} = 0.7547$	$\frac{8}{9.3333} = 0.8571$	$\frac{5}{9} = 0.5556$
Freshness	$\frac{0.125}{1.325} = 0.0943$	$\frac{1}{9.3333} = 0.1071$	$\frac{3}{9} = 0.3333$
Accuracy	$\frac{0.2}{1.325} = 0.1509$	$\frac{0.3333}{9.3333} = 0.0357$	$\frac{1}{9} = 0.1111$

Table B.5: Calculating Final Weights for Criteria

	Criteria Weights
Relevance	$average(0.7547, 0.8571, 0.5556) = 0.7225$
Freshness	$average(0.0943, 0.1071, 0.3333) = 0.1783$
Accuracy	$average(0.1509, 0.0357, 0.1111) = 0.0993$

Comparison Matrices and Final Weights Once we have completed the pairwise comparison and have the relative AHP scores for each criterion and task as shown in Tables B.2 and B.3, the AHP process can calculate the final weights for the tasks. The values in Table B.3 are generated based on the surveillance example data and Table B.1. Tables B.4 and B.5 show how to calculate the weights allocated to each of the criteria.

The final task weight is the sum of the products of the criteria weight and the task weight for that criteria i.e. for video $(0.0582 * 0.7225) + (0.3214 * 0.1783) + (0.2286 * 0.0993) = 0.1221$. The weights allocated to each of the criteria (Relevance, Freshness, and Accuracy) are 0.7225, 0.1783, and 0.0993. The resulting final task weights for Video, Image,

Temperature and Humidity are 0.4672, 0.3039, 0.1145 and 0.1145, respectively (Table B.6). The normalized task weights, $(w_k) = (0.569, 2.091, 1, 1)$, are then passed to PERMIT to be incorporated into the energy allocation and adaptation process.

Table B.6: Overall weight

	Relevance (0.7225)	Freshness (0.1783)	Accuracy (0.0993)	Final Weights	Normalized Weights
Video	0.0582	0.3214	0.2286	0.1221	0.569
Image	0.5241	0.25	0.2571	0.4488	2.091
Temp.	0.2088	0.2143	0.2571	0.2146	1
Humidity	0.2088	0.2143	0.2571	0.2146	1

B.2 Priority Mechanism Evaluation

To evaluate the AHP-based priority mechanism we use the actual measured Cloudy Day profile data and compare the task values selected when using weights $w1 = [1, 1, 1, 1]$ and $w2 = [0.57, 2.09, 1, 1]$ for the video, image, temperature, and humidity tasks respectively with PERMIT, and the results are shown in Fig. 4.12. Using weight $w1$ gives all tasks equal priority. With $w2$ the priority for the image task has been increased while the video task priority has been decreased.

With $w2$, we see that the task values for the image has increased since more energy has been allocated to it. The extra energy was mostly taken from the video task which now has the lowest priority. A small amount of energy was also taken from the temperature and humidity tasks because with the new weights, the proportion of energy allocated to them decreased slightly. The video task values do not have a substantial change, since changing a video task value by 1s needs about 0.713J. Comparatively the temperature and humidity

tasks need 0.0058J and 0.0086J respectively for each additional measurement. Therefore, a small reduction in video energy (less than what is needed to change the video duration by 1s) is sufficient to see a significant change in the image task values when it is given a higher priority. For tasks that have a significant difference in energy cost, causing a change in the task value for the higher energy task requires a large difference in priorities. In cases where all the tasks have a comparable energy cost, increasing/decreasing the task value per unit using priority weights like $w2$ is sufficient to result in a significant change in task values. This is because for every decrease in the task value of a lower-priority task, the energy saved from that task is sufficient to increase the task values for the higher-priority task(s).