Reinforcement Learning for Data Scheduling in Internet of Things (IoT) Networks

by

Hootan Rashtian

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

August 2020

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Reinforcement Learning for Data Scheduling in Internet of Things (IoT) Networks

submitted by **Hootan Rashtian** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Electrical and Computer Engineering**.

Examining Committee:

Sathish Gopalakrishnan, Electrical and Computer Engineering
Supervisor

Vijay Bhargava, Electrical and Computer Engineering
Supervisory Committee Member

Lutz Hans-Joachim Lampe, Electrical and Computer Engineering
University Examiner

Bhushan Gopaluni, Chemical and Biological Engineering
University Examiner

Louis Almeida, Electrical and Computer Engineering, University of Porto
External Examiner

Abstract

I investigate data prioritization and scheduling problems on the Internet of Things (IOT) networks that encompass large volumes of data. The required criteria for prioritizing data depend on multiple aspects such as preservation of importance and timeliness of data messages in environments with different levels of complexity. I explore three representative problems within the landscape of data prioritization and scheduling.

First, I study the problem of scheduling for polling data from sensors where it is not possible to gather all data at a processing centre. I present a centralized mechanism for choosing sensors to gather data at each polling epoch. Our mechanism prioritizes sensors using information about the data generation rate, the expected value of the data, and its time sensitivity. Our work relates to the restless bandit model in a continuous state space, unlike many other such models. The contribution is to derive an index policy and show that it can be useful even when not optimal through a quantitative study where event arrivals follow a hyper-exponential distribution.

Second, I study the problem of balancing timeliness and criticality when gathering data from multiple sources using a hierarchical approach. A central decision-maker decides which local hubs to allocate bandwidth to, and the local hubs have to prioritize the sensors' messages. An optimal policy requires global knowledge of messages at each local hub, hence impractical. I propose a reinforcement-learning approach that accounts for both requirements. The proposed approach's evaluation results show that the proposed policy outperforms all the other policies in the experiments except for the impractical optimal policy.

Finally, I consider the problem of handling timeliness and criticality tradeoff when gathering data from multiple resources in complex environments. There exist dependencies among sensors in such environments that lead to patterns in data that are hard to capture. Motivated by the success of the Asynchronous Advantage Actor-Critic (A3C) approach, I modify the A3C by embedding Long Short Term Memory (LSTM) to improve performance when vanilla A3C could not capture patterns in data. I show the effectiveness of the proposed solution based on the results in multiple scenarios.

Lay Summary

When we have numerous sources of data, we cannot collect data from every source. Our ability to gather data is bounded by fundamental constraints such as bandwidth. Consequently, we need to prioritize data sources based on criticality and timeliness requirements. We explore data prioritization questions in the Internet of Things setting wherein we may have to manage data collection from many sensors; in the presence of requirements and restrictions in the network environment. The solutions we present here build upon recent work in the area of reinforcement learning.

Preface

All of the work presented henceforth (Chapters 2-4) was conducted in the Real-Time and Dependable Computing Laboratory (RADICAL) at the University of British Columbia, Point Grey campus. I developed the algorithms, worked on proofs of correctness, and carried out the numerical evaluations described in this dissertation. My supervisor, Sathish Gopalakrishnan, worked closely with me on sharpening the problem formulation and clarifying the research context. Chapter 2 includes to-be-published work co-authored by Bader Alahmad.

- Chapter 2 is under peer review, following revisions.

I developed, implemented and evaluated the method. Bader Alahmad helped with proofs verification and proof reading of the work. Profesor Gopalakrishnan provided feedback and suggestions in improving the formulation, the methodology and the evaluation.

- Chapter 3 has been published as **H. Rashtian** and S. Gopalakrishnan,

"Balancing Message Criticality and Timeliness in IoT Networks" in IEEE Access, vol. 7, pp.145738-145745.

I developed, implemented and evaluated the method. Profesor Gopalakrishnan provided feedback and suggestions in improving the formulation, the methodology and the evaluation.

- Chapter 4 has been published as **H. Rashtian** and S. Gopalakrishnan, "Using Deep Reinforcement Learning to Improve Sensor Selection in the Internet of Things" in IEEE Access, vol. 8, pp. 95208-95222.

I developed, implemented and evaluated the method. Profesor Gopalakrishnan provided feedback and suggestions in improving the methodology and the evaluation.

Contents

Abstract . . . . . **iii**

Lay Summary . . . . . **v**

Preface . . . . . **vi**

Contents . . . . . **viii**

List of Tables . . . . . **xii**

List of Figures . . . . . **xiv**

Glossary . . . . . **xxvii**

Acknowledgments . . . . . **xxviii**

1 Introduction **1**

1.1 Broad Research Agenda 2

1.2 Problem Statement 4

1.3 Contributions 5
1.3.1 Polling IoT sensors with time-sensitive data 5
1.3.2 Balancing criticality and deadline in IoT networks . . 6
1.3.3 Handling the trade-off of criticality vs. timeliness in complex environments 7

2 Polling Sensors with Time-Sensitive Data: Restless Bandits Revisited 9
2.1 Introduction 10
2.2 Model 15
2.3 Indexability, or the Existence of a Priority-Driven Policy . . . 19
2.4 Computing the Index/Priority 26
2.5 Adaptive Estimation of Accrued Utility 32
2.6 Explicit Analysis of Stochastic Arrivals 34
2.7 Numerical Evaluation and Analysis 44
2.7.1 Identical bandwidth costs 49
2.7.2 Varied bandwidth costs 49
2.7.3 IP_v vs. IP_f 51
2.7.4 Selecting arms using indices: top-k arms vs. knapsack packing 52
2.7.5 Insight from numerical evaluations 54
2.8 Related Work 55
2.9 Conclusions 62

3 Balancing Message Criticality and Timeliness in IoT Networks: A Q-Learning Approach 66

3.1 Introduction 67

3.2 System Model 68

3.3 Optimal Offline Policy 70

3.4 Using Reinforcement Learning in a Decentralized Policy . . . 75

3.4.1 At Local Hubs 76

3.4.2 At the Central Hub 78

3.5 Alternative Policies at the Central Hub 79

3.6 Quantitive Evaluation 80

3.6.1 Experimental Parameters 81

3.6.2 Evaluation Scenarios 82

3.6.3 Results 85

3.7 Related Work 87

3.8 Conclusions and Future Work 90

4 Handling the Message Criticality vs. Timeliness Tradeoff in Complex IoT Environments: A Deep RL Approach 92

4.1 Introduction 93

4.1.1 Context 98

4.1.2 Contributions 100

4.2 System Model 101

4.2.1 Performance Metric 102

4.2.2 Problem Statement 103
4.3 Background 104
4.3.1 Overview of Deep Reinforcement Learning 104
4.3.2 A3C Network 109
4.3.3 Proposed Approach 114
4.3.4 Environment 119
4.4 Quantitative Evaluation 119
4.4.1 Evaluation Scenarios 120
4.4.2 Experimental setup 122
4.4.3 Results 123
4.5 Related Work 127
4.6 Conclusions and future work 128

5 Conclusions & Future Work 132
5.1 Limitations 134
5.2 Potential Future Directions 136

Bibliography 138

A Supplementary material for Chapter 2 147

B Supplementary results of Chapter 4 149
B.1 Experiments results for 4 sensors 149
B.2 Experiments results of 16 sensors 152
B.3 Table of results summary 155

List of Tables

Table 2.1 **Comparison of algorithms in the cases of fixed and varied bandwidth costs**. The table shows the percentage of simulation runs wherein the index policy outperforms other algorithms as well as the average performance advantage. 48

Table 3.1 The table shows the average Missed Criticality Ratio for the policies in each of the scenarios. As shown in green colored cells, the proposed policy (VWP) consistently performs as the second best policy after the optimal policy. . 87

Table 4.1 Example of sensor selection in a 4-step time window, when dependency **cannot** be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor i is shown as $S_i = (cr, d)$. As the table shows, the total accumulated criticality at the end of the time step $t = 4$ is 8. 96

Table 4.2 Example of sensor selection in a 4-step time window, when dependency **can** be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor i is shown as $S_i = (cr, d)$. As the table shows, the total accumulated criticality at the end of the time step $t = 4$ is 11. 96

Table 4.3 The table summarizes the results for the case of 8 sensors. It reports the average Criticality-weighted Deadline Miss Ratio for the policies in each of the scenarios. As shown in green-coloured cells, the proposed policy (m_A3C) consistently performs as the best policy. Even when it is not the best (i.e., Scenario II), it still performs reasonably well with only one percent of difference (concerning ρ) compared to the best result. Also, the red-coloured cells represent the worst performance (by the greedy policies) across all the scenarios. 131

Table B.1 The summary of all experiments concerning the Criticality-weighted Deadline Miss Ratio. 155

List of Figures

Figure 2.1 **The flowchart of sensor polling process.** 31

Figure 2.2 **Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 4, Bandwidth Limit = 2.** The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 50

Figure 2.3 **Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 8, Bandwidth Limit = 4.** The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 51

Figure 2.4 **Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 16, Bandwidth Limit = 8.** The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 52

Figure 2.5 **Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 32, Bandwidth Limit = 16** The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 53

Figure 2.6 **Performance of different policies with randomly selected bandwidth costs where #Bandits = 4, Bandwidth Limit = 2.** The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 54

Figure 2.7 **Performance of different policies with randomly selected bandwidth costs where #Bandits = 8, Bandwidth Limit = 4.** The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 55

Figure 2.8 **Performance of different policies with randomly selected bandwidth costs where #Bandits = 16, Bandwidth Limit = 8.** The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 56

Figure 2.9 **Performance of different policies with randomly selected bandwidth costs where #Bandits = 32, Bandwidth Limit = 16.** The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 57

Figure 2.10 **Comparison of $\mathbf{IP_v}$ and $\mathbf{IP_f}$ in terms of total accrued rewards with identical bandwidth.** $\mathrm{IP_v}$ shows a consistent advantage with respect to $\mathrm{IP_f}$ in all simulations setups. 58

Figure 2.11 **Comparison of $\mathbf{IP_v}$ and $\mathbf{IP_f}$ in terms of total accrued rewards with varied bandwidth.** $\mathrm{IP_v}$ shows a consistent advantage with respect to $\mathrm{IP_f}$ in all simulations setups. 59

Figure 2.12 **Comparing top-k selection with knapsack packing for index policy**: We notice no significant difference between the two approaches so the top-k approach suffices. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages. 60

Figure 3.1 The hierarchical model where unit-length messages with deadlines (d_i) and criticalities (κ_i) arrive at local hubs and then transmitted to the central hub (based on a policy). 69

Figure 3.2 The details of processing messages at local hubs. First, sensors detect events and send messages to local hubs. Second, messages are routed to appropriate queues based on the priority assigned to each messages. Messages in the highest priority queue, are emptied first before messages at queues with lower priority level. 75

Figure 3.3 We use four evaluation scenarios based on varying the distribution of messages in terms of criticality and deadline values. Scenario V, not in the figure, is where each the deadline and criticality of each message is chosen uniformly at random from a range of values. 81

Figure 3.4 Range of criticality-weighted miss ratio (ρ) values for Scenario I. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy. 82

Figure 3.5 Range of criticality-weighted miss ratio (ρ) values for Scenario II. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy. 83

Figure 3.6 Range of criticality-weighted miss ratio (ρ) values for Scenario III. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy. 84

Figure 3.7 Range of criticality-weighted miss ratio (ρ) values for Scenario IV. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy. 85

Figure 3.8 Range of criticality-weighted miss ratio (ρ) values for Scenario IV. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy. 86

Figure 4.1 The system model where the central unit selects a sensor at a time to transmit data. Sensors hold a message with deadline (d_i) and criticality (κ_i). The model makes no assumptions about the arrival rate of events sensed by the sensors. 102

Figure 4.2 An example of A3C's limitations, where its performance degrades in complex scenarios with 8 sensors. The Y-axis is the *Criticality-weighted Deadline Miss Ratio* and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} \frac{Cr_i \times \lambda_{arrival}}{d_i}$. According to the definition of the workload intensity, it increases with an increase in criticality (Cr_i), an increase in data arrival ($\lambda_{arrival}$) or a decrease in deadline (d_i) of messages. 112

Figure 4.3 The proposed A3C-based network with embedded memory (i.e., LSTM layer). 116

Figure 4.4 An example graph of the reward function for some parameter choices ($\iota = 1$, $\alpha = 2$, $\beta = 4$, and κ and d are randomly selected from interval of $(1,5)$. The peaks correspond to the cases where no penalty is incurred (i.e., $I = 0$), whereas the troughs correspond to the cases with a penalty (i.e., $I = 1$). 118

Figure 4.5 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C. . . 123

Figure 4.6 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C. . . 124

Figure 4.7 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) outperforms other policies. 126

Figure 4.8 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) outperforms other policies. 126

Figure B.1 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 150

Figure B.2 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 150

Figure B.3 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 151

Figure B.4 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 151

Figure B.5 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C. . . 152

Figure B.6 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 153

Figure B.7 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 153

Figure B.8 Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C. 154

Glossary

IOT Internet of Things

A3C Asynchronous Advantage Actor-Critic

LSTM Long Short Term Memory

VWP Value-Weighted Policy

DG Deadline-Greedy

CG Criticality-Greedy

CDG Criticality Density Greedy

RA Random

GRA Global Random

Acknowledgments

I want to express my most profound appreciation to my parents, brother, and my wife, for always standing by me. I love you!

I would also like to acknowledge the support of my supervisor Professor Sathish Gopalakrishnan, who believed in me and encouraged me to gain new industrial and academic experiences. The shortest of conversations and interactions with him were a learning experience for me.

Also, there are many friends of mine with whom I enjoyed exchanging ideas.

Finally, I would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of British Columbia for their financial support.

Chapter 1

Introduction

The amount of data being generated – by sensors and by humans – and shared online through the Internet has reached a volume that is exceedingly challenging to manage and work with. IBM reported that *more than 2.5 quintillion data bytes* are being generated each day [2]. An example of a rapidly growing segment of the cyber-physical world that will add to the data volume is the Internet of Things (IOT) [3]. Whether data are generated for direct human consumption or preliminary processing by other computational devices, the amount of data being generated will stress computational infrastructure (e.g., networks and processing nodes) and compete for human attention.

As another example of this enormous generation of data, Twitter reported

that its users add 350,000 tweets each minute [5]. Similarly, there are 680,000 new Facebook posts [1] and 100 hours worth of videos are uploaded to YouTube [6] *each minute*.

Computational approaches to prioritize data are essential to coping with the generation of such data. Such methods will allow system operators (such as IoT network admins) to allocate suitable resources to content that need it (e.g., replication levels to manage content that is frequently requested). For individuals that consume the data/content, these methods will help prioritize content and draw attention to relevant items. Algorithms for prioritizing data will prevent users from being overwhelmed by the data deluge. Such methods will also have to incorporate the time-varying nature of data: only some items may be effectively useful over the long-run, but capturing short-term usefulness is also essential. A notion of "value" also captures the *criticality* of some data. In some cases, criticality may be established ahead of time, but it is also the case that we may have to infer criticality in certain situations.

1.1 Broad Research Agenda

The goal of the work we performed was to prioritize data streams *effectively* and *quickly*. Quick detection is essential because some content may lose value after a short period and to be effective, we need to make decisions timely. An example in this context may be data from IoT sensors. For

example, traffic intensity information at an intersection is valuable when someone is planning their travel route, and any data obtained late is not of much value. On a longer timescale and different contexts, we see the same effect with data (i.e., videos) on playing a game like Pokemon Go. Tutorial videos on how to play the game may have high viewership when the game is released but may see diminished interest after gameplay has been assimilated into popular culture [4].

Discerning when a data source is important/critical is difficult because there are no apparent cues in a general setting. One will need to use certain aspects of the content – the metadata – as features that help in the identification process. In the IoT context, despite the large volume of data, less metadata may be available. However, we will still need to prioritize data flows when resources (e.g., storage, bandwidth, and power) are limited. Such property imposes an additional challenge compared to the systems (e.g., social media) that social engagement features (e.g., likes and comments) could be used as metadata.

Broadly, this thesis explores three problems related to prioritizing data streams in different IoT settings. These settings can be articulated, at least partially, along three dimensions: (i) resource limitations (ii) data properties (e.g., criticality and timeliness) and (iii) environment complexity (e.g., whether data resources (e.g., sensors) are correlated in time or space). The first problem is about collecting valuable data in an environment that

is constrained by resources (i.e., bandwidth limit). At the same time, it contains time-sensitive data from sensors in a less complex environment dynamics. The second problem is very much about balancing sensors data properties of criticality and timeliness while dealing with some limitations in less complex settings. In the third problem, we considered a case that encompasses resource limitations and data properties in a complex environment, where data resources (e.g., sensors) may be correlated in time or space.

The goal here was to study these cases as a representative set of cases alongside the above dimensions. Indeed, these cases can be further explored in scenarios with different configurations concerning resource limitations, properties associated with sensible data, and environmental complexity.

1.2 Problem Statement

Given the research agenda explained in Section 1.1, our enquiries can be summarized by the following question:

What techniques can we use to prioritize data given the existing resource constraints and properties in data-rich environments such as IoT networks?

While this research question shapes the theme of the research throughout the thesis, we answer three specific questions that help us answer the more general question state above.

1.3 Contributions

The contributions of this dissertation are three-fold:

1.3.1 Polling IoT sensors with time-sensitive data

When we have to periodically poll sensors (from a large set), we present a mechanism for determining which sensors to gather data from at each polling epoch. Our sensor polling mechanism prioritizes sensors using information about the data generation rate, the expected value of the data as well as its time sensitivity.

Our problem formulation and its solution relate to the restless bandit model for sequential decision making (**Section 2.2**). The problem we study bears similarity with the work by Kleinberg [31] but the fact that a sensor's state may grow with time (as events are observed) and decay (as the data becomes stale) does not permit for the same analysis as in the case of recharging bandits. Prior work has focused on the state space of the arms being discrete, but, as we shall see, in our formulation, the state space is continuous (albeit closed) and requires that we establish the existence (**Section 2.3**) and suitability of priority/index policies [50]. We proved that similar techniques as those in restless bandits with discrete space could be used because of particular characteristics of the underlying problem (**Section 2.7**). We then showed that our approach could be advantageous even when not optimal through an extensive quantitative study where event arrivals fol-

low a hyper-exponential distribution (**Section 2.7**).

1.3.2 Balancing criticality and deadline in IoT networks

When sensors are organized hierarchically, we present a reinforcement learning approach for message scheduling in a system with many nodes that will help us strike a balance between timeliness and criticality (**Section 3.4**). In a two-level hierarchical architecture, devices that generate data transmit them to a local hub. A central decision-maker then has to decide which local hubs to allocate bandwidth to, and the local hubs have to prioritize the messages they transmit when allowed to do so. We showed and proved that an optimal policy does exist for this problem (**Section 3.3**). Though such a policy would require global knowledge of messages at each local hub, rendering such a scheme impractical.

Despite similarities of job scheduling in distributed environments [36, 37] on timing constraints, our work is distinguished by focusing on striking criticality and timeliness in a distributed IoT-like setup. Also, there has been a large amount of optimization research done on job scheduling in the distributed computing environment [12, 28, 56, 63, 72], which is not directly applicable to the massage scheduling problem that we addressed in hierarchical networks.

We evaluated our solution using a criticality-weighted deadline miss ratio as the performance metric (**Section 3.6**). The performance analysis was

done by simulating the behaviour of the proposed policy as well as that of several natural policies under a wide range of system conditions. The results show that the proposed policy outperforms all the other policies - except for the optimal but impractical policy - under the range of system conditions studied and that in many cases, it performs close (3% to 12% lower performance depending on the condition) to the optimal policy.

1.3.3 Handling the trade-off of criticality vs. timeliness in complex environments

When data sources may be correlated in space and time, we suggest a deep reinforcement learning solution to the problem of handling timeliness and criticality trade-offs (**Section 4.3.3**). Such correlation patterns are difficult to utilize without enhancing existing schemes (**4.4.1**).

While our work is similar to a body of research that addresses understanding patterns in complex networks [14, 33, 71], it is different from them both in terms of the system model (i.e., centralized) and the solution approach (i.e., memory-based DRL approach).

We chose Asynchronous Advantage Actor-Critic (A3C) as the underlying model for our proposed solution. We first mapped vanilla A3C into our problem to compare its performance in terms of *Criticality-weighted Deadline Miss Ratio* to the considered baselines in multiple scenarios. Then, we modified the A3C network by embedding "Long Short Term Memory

(LSTM)" to improve performance in cases that vanilla A3C could not capture patterns in data. Simulation results (**Section 4.4.3**) show that the modified A3C reduces the metric of "Criticality-weighted Deadline Miss Ratio" from 0.3 to 0.19. Moreover, results indicate that the proposed solution performs well in less complex cases, which makes it a generalized solution.

Chapter 2

Polling Sensors with Time-Sensitive Data: Restless Bandits Revisited

Summary. In a sensor-rich Internet of Things environment, we may be unable to gather all data at a processing centre at the rate at which the data is generated. The rate of data collection from a sensor may be limited by available bandwidth/cost (or energy considerations), especially if one were to use cellular networks for such systems. In this context, we present a mechanism for determining which sensors to gather data from at each polling epoch. Our sensor polling mechanism prioritizes sensors

using information about the data generation rate, the expected value of the data as well as its time sensitivity. Our problem formulation and its solution relate to the restless bandit model for sequential decision making. Whereas existing methods for the restless bandit model are not directly applicable because the state space is continuous and not discrete, we prove that similar techniques can be used because of special characteristics of the underlying problem. We then show that our approach can be very effective even when not optimal through an extensive quantitative study where event arrivals follow a hyper-exponential distribution.

2.1 Introduction

Sensor systems are a significant component of the Internet of Things (IoT). Dense sensor deployments are a rich source of data about the physical world and are used to inform decisions in a variety of applications. In essence, agriculture, transportation, and emergency responses are a few such applications. Another source of data is certain types of online social networks/services, such as Twitter, which also contain feeds that influence decision making in the types of applications that also rely on physical sensing. Also, *privacy* in data collection can be an important application that uses data from sensor deployments in environments.

We consider a scheduling problem that arises in the context of detecting events across multiple locations in a system. Imagine that we would like to

know if an event was observed at some location. We would like to know of this event relatively quickly (delay sensitivity). However, there are many locations to monitor, making it infeasible to poll each location at each decision epoch. One possible approach to this problem would be to set up triggers at each location for every object or event of interest and be informed when the event occurs. A downside to this approach – one that we consider – is that we may have to inform each location of the events of interest, potentially compromising the privacy of the events of interest (privacy sensitivity). We, therefore, study a model where a central entity polls an observation point (location) and obtains all observations since the previous polling request to that point. This central entity may then identify the events of interest from the obtained set of observations to take suitable action. We can also use the term sensors to refer to these observation points, and we will use these terms interchangeably for the rest of this chapter.

From the design perspective, and with the above-mentioned central entity, we can assume that we have a centralized data service that can provide subscription services to applications. The centralized entity may not have the most accurate knowledge of sensor data (and thus, the value of the data). However, one benefit of performing centralized scheduling is to enforce a global bandwidth constraint. ("Centralization" may still use distributed computing techniques but we can think of a single entity that plans the data collection and data warehousing.) The central entity has limited bandwidth

and can only poll some sensors at each decision epoch. The question that then arises is: Which sensors should be polled? We may want to prioritize sensors that have not been polled recently and at the same time account for the fact that some sensors may provide more valuable data consistently.

Another advantage of this model is that sensor deployment costs may be amortized over multiple applications, and simple application programming interfaces (APIs) allow many applications to be developed easily. This model has also been referred to as *sensing-as-a-service* [45]. In the sensing-as-a-service model, as more sensors join the network, it becomes increasingly challenging to select appropriate sensors such that the services mentioned above (e.g., APIs) are efficiently available.

Our focus is on the abstraction of polling sensors where we have some notions of delay sensitivity and privacy sensitivity (or other limitations) that prevent a push-driven architecture. If the value derived from different events may be different, then a simple round-robin polling policy may not be suitable. *We study how a single entity can perform as an efficient scheduler for sensors data collection operations when detecting incidents/events in IoT platforms.* The growth in sensor deployments and the volumes of data generated [45] necessitate careful scheduling of data collection from sensors. Such a process is essential due to potential constraints such as the available communication bandwidth (which can be overwhelmed by an approach that simply queries all sensors periodically) or the direct cost of

accessing the underlying network. One possible design strategy for such systems would be to utilize the cellular network infrastructure, which can be expensive, although simple to implement and manage.

We propose a centralized approach to periodically collecting sensor data. To manage the process of polling sensors for data, we consider two issues related to the sensors and the data they collect:

1. *Data Value*: Not all data is equal in a data-rich world. The data stream from one sensor may be deemed more important than the stream from a different sensor. Such judgment of value may be a result of factors such as sensor location and sensor type. The *value* can be quantified, for instance, by the number of API queries to use the data in the sensing-as-a-service model.

2. *Time Sensitivity*: Some data may be important for long-range statistics, but some data may be needed soon after the associated observation. Further, data may lose its value as the time from the associated observation increases.

Using the context that we have provided so far, we propose a policy to prioritize sensors at each polling (data collection) epoch to respect the bandwidth (or cost) constraints and to maximize the long-term average value obtained. In modelling the underlying problem (Section 2.2), we treat the sensors as offering time-varying rewards based on when we poll them. The

reward from a sensor is the "value" seen by the sensing service. Such a reward depends on the data that the sensor recorded and on when it recorded the data.

Our problem model is related to the restless bandit model for sequential decision making [69] due to Whittle. *The central difference, however, between our formulation and the classic restless bandit formulation is that the state-space for our problem is continuous, whereas it is discrete in Whittle's formulation. It is this difference that requires the rigorous treatment we present.*

Our main contributions are as follows:

- Establishing that the polling problem can be solved using a dynamic program that has a unique solution (Section 2.3);
- Deriving a simple dynamic priority, or index, policy that allows us to approximate the solution to the [intractable] dynamic program (Sections 2.4 and 2.6);
- Identifying an adaptive and improved index policy when event arrivals at sensors are modeled using hyper-exponential distributions to capture a wider range of operating conditions (Section 2.5);
- Demonstrating the effectiveness of the index policy using numerical evaluations. (Section 2.7).

Organization. We start by presenting the related work (Section 2.8) to support and position our contributions. We then discuss the model that captures the sensor scheduling problem (Section 2.2) before presenting the main technical results highlighted in the list of contributions (Sections 2.4, 2.5, 2.6 and 2.7). The last section summarizes our findings and discusses extensions to this work (Section 2.9).

2.2 Model

Consider a sensor deployment consisting of N sensors $S_1, \ldots, S_N$. Let $[N] = \{1, \ldots, N\}$. The data sensed by sensor $i \in [N]$ has initial value v_i, which is a non-negative random variable with known finite mean $\overline{v}_i < \infty$, and this value decreases exponentially with rate $\beta_i > 0$. This means that data collected t units of time after it was initially sensed has (random) value $v_i e^{-\beta_i t}$. We assume that $v_1, \ldots, v_N$ are i.i.d. Also, we assume that data is collected periodically with period $P > 0$.

Moon et al. have presented the approach of assigning value to gathered data based on its use in their work on a learning framework for improving search results [41].

We assume that the successive events that a sensor may detect are such that the inter-arrival times are governed by a hyper-exponential distribution, which is a mixture of exponential distributions. This assumption about inter-arrival times captures a wide range of operating conditions because

hyper-exponential distributions can capture exponential distributions and approximate heavy-tailed distributions. Sensor S_i, $i \in [N]$ senses new events from the environment at rate $\mu_{i,j} > 0$ with a probability q_j; we assume that events are sensed, or equivalently, events arrive at the sensors, according to a hyper-exponential process that is a mixture of M exponential distributions. At each collection period, we select the sensors to poll and thus collect the sensed event data.

Our next step is to describe the expected utility/value that we can obtain when we poll a sensor. We define the expected utility/value from polling sensor S_i as the average value over the average discounted time spanning one period; that is,

$$\upsilon_i = \sum_{j=1}^{M} q_j \mu_{i,j} \int_0^P \overline{v}_i e^{-\beta_i t}\, dt = \frac{\sum_{j=1}^{M} q_j \mu_{i,j}}{\beta_i} \overline{v}_i \left(1 - e^{\beta_i P}\right). \tag{2.1}$$

We will use the estimate of utility accrued in a period, as described above, in our initial discussion. Later (Section 2.5), we will show that we can obtain improved estimates using a property of the hyper-exponential distribution.

Each sensor may have sensed multiple events between two sampling instants. Let $\Upsilon_i(t)$ be the utility accumulated at sensor S_i at time t. This is also the state of S_i at time t. Let $a_i(t)$ be the action taken for sensor S_i at

time t, which we define as

$$a_i(t) = \begin{cases} 1, & \text{poll} \\ 0, & \text{idle (do not poll)}. \end{cases}$$

The evolution of the state of each sensor depends on the polling action. There are two cases. For convenience, let $\alpha_i = e^{-\beta_i P}$. First, if $a_i(t) = 0$, then no reward is obtained; that is, $r_i(t) = r_i\big(\Upsilon_i(t), a_i(t)\big) = 0$, and the state will be changed at the next polling period according to

$$\Upsilon_i(t+P) = \alpha_i \Upsilon_i(t) + \upsilon_i. \tag{2.2}$$

Second, if $a_i(t) = 1$, then the reward obtained is the accumulated utility up to time t; that is, $r_i(t) = r_i\big(\Upsilon_i(t), a_i(t)\big) = \Upsilon_i(t)$. In this case, the state is "reset" to its initial value: $\Upsilon_i(t+P) = \upsilon_i$.

Note that the controlled state process $\Upsilon = \{\Upsilon(t) : t \geq 0\}$ depends on the sequence of actions. Moreover, Υ is a deterministic process; this is because 1) we are working with the expectations of the random elements involved in the definition of the state, and 2) we are considering deterministic decisions.

In this model, our objective is to maximize the average reward over the

infinite horizon:

$$\liminf_{j\to\infty} \sum_{i=1}^{N} \frac{1}{j} \sum_{m=0}^{j} r_i\big(\Upsilon_i(mP), a_i(mP)\big).$$

There is also a "bandwidth" constraint defined as

$$\limsup_{j\to\infty} \sum_{i=1}^{N} \frac{1}{j} \sum_{m=0}^{j} \gamma_i a_i(mP) = \mathsf{B},$$

where $\gamma_i > 0$ is the "bandwidth" required by sensor S_i, and B is the given average bandwidth that is available for the overall expected sensor polling activity.

We assume that sensor S_i requires bandwidth γ_i irrespective of the number of observations it reports when polled. We could treat γ_i as a random variable, but the long-term behaviour can be approximated by using the mean $\overline{\gamma_i}$. We will assume a deterministic γ_i for the rest of the chapter.

Our formulation is related to the restless multi-armed bandit framework [69], *but* the difference in our formulation is that the state space is a closed domain in $\mathbb{R}^N$. The general analysis in this setting requires establishing some important results.

In the specific case that $\gamma_i = 1$ for all $i \in [N]$, our problem reduces to the original restless multi-armed bandit setting and B becomes the expected *number* of sensors that need to be polled at every decision epoch.

We shall next examine index policies, wherein the global problem with

N sensors (arms) is decomposed into N single-sensor problems. In what follows, we shall use the terms *arm* and *sensor* interchangeably. When referring to an arm, we will use the term *pulling* for the sensor polling activity.

In the first part of our discussion (Sections 2.3 and 2.4), we will assume a fluid-flow approximation of the stochastic event arrival process, which triggers observations at sensors. Event arrivals over a time window are averaged, and each observation at a sensor has the same value, although observations at different sensors may have a different value. Later (Section 2.6) we will relax this assumption and study the truly stochastic behaviour of the system where events arrive at discrete time points, and the value of observation may differ at a sensor, but the mean value of a sensor observation is known.

2.3 Indexability, or the Existence of a Priority-Driven Policy

We consider the average reward problem for S_i using the theory of Lagrange multipliers. To this end, denote as Υ_i the state space of sensor S_i, and let $v_i(s_i, a)$ be the immediate reward that S_i receives when the state is $s_i \in \Upsilon_i$ and the action taken is $a_i \in \{0, 1\}$. That is,

$$v_i(s_i, a_i) := s_i a_i + \lambda \gamma_i (1 - a_i), \tag{2.3}$$

where λ is a Lagrange multiplier. One may interpret λ as a subsidy allocated to sensor i so as to make idling (non-polling) more attractive.

Using ρ as discount factor ρ, we can define the discounted reward over an infinite horizon as:

$$\sum_{t=0}^{\infty} \rho^t v_i\big(\Upsilon_i(t), a_i(t)\big). \tag{2.4}$$

The value function associated with 2.4 is

$$V_\rho(s_i) := \sup_{\{a_i(t):t\}} \left\{ \sum_{t=0}^{\infty} \rho^t v_i\big((\Upsilon_i(t), a_i(t)\big) : \Upsilon_i(0) = s_0 \right\}.$$

Now the dynamic programming equation may be written as

$$V_\rho(s_i) = \max\big\{\lambda\gamma_i + \rho V_\rho(\alpha_i s_i + \upsilon_i), s_i + \rho V_\rho(\upsilon_i)\big\}. \tag{2.5}$$

Lemma 1. *The solution to dynamic program defined by 2.5, V_ρ is (i) unique and bounded, (ii) continuous over $\rho \in (0,1)$ (in the Lipschitz sense), and (iii) monotonically increasing and convex.*

Proof of Lemma 1

Proof. To show 1), we refer to the theory of Discrete Time Markov Chains (DTMC) where having a unique bounded continuous solution for V_ρ is standard [57]. For 2), consider $s_i, s_i' \in \Upsilon_i$ with $s_i' \neq s_i$, and consider processes $\{\Upsilon_i(t)\}$ and $\{\Upsilon_i'(t)\}$ with initial conditions (states) s_i and s_i', respec-

tively. Both processes are controlled by the same actions $\{a(t)\}$. Denote as T the first time instant at which the action is to poll the sensor; that is, $T = \inf\{t \geq 0 : a(t) = 1\}$. Since the state is reset when polling the sensor, it follows that $\Upsilon_i(t) = \Upsilon'_i(t)$ for all $t > T$ (since the action sequence is the same for both processes), and thus $\left[v_i\big(\Upsilon'_i(t), a(t)\big) - v_i\big(\Upsilon_i(t), a(t)\big)\right] = 0$ for all $t > T$. On the other hand, the action at each $t < T$ is to keep the sensor idle, and by 2.3, $v_i\big(\Upsilon'_i(t), a(t)\big) = v_i\big(\Upsilon_i(t), a(t)\big) = \lambda\gamma_i$ (independently of the state), and therefore $\left[v_i\big(\Upsilon'_i(t), a(t)\big) - v_i\big(\Upsilon_i(t), a(t)\big)\right] = 0$ for all $t < T$. Thus, we have

$$\begin{aligned} V_\rho(s'_i) - V_\rho(s_i) &= \sum_{t=0}^{\infty} \rho^t \left[v_i\big(\Upsilon'_i(t), a(t)\big) - v_i\big(\Upsilon_i(t), a(t)\big)\right] \\ &= \rho^T \left(\Upsilon'_i(T) - \Upsilon_i(T)\right). \end{aligned}$$

At $t = T$, the states of processes $\{\Upsilon_i(t)\}$ and $\{\Upsilon'_i(t)\}$ will have evolved according to (2.2), which for any integer $k \geq 0$ and initial state s_i gives

$$\Upsilon_i(kP) = \left(1 + \alpha_i + \cdots + \alpha_i^k\right) s_i = \left[\frac{1 - \alpha_i^{k+1}}{1 - \alpha_i}\right] s_i.$$

Observing that T is an integer multiple of P, we have

$$\Upsilon_i(T) = \Upsilon_i\big((T/P)P\big) = \left[\frac{1 - \alpha_i^{(T/P)+1}}{1 - \alpha_i}\right] s_i.$$

Thus, we have

$$V_\rho(s_i') - V_\rho(s_i) = \rho^T \left[\frac{1-\alpha_i^{(T/P)+1}}{1-\alpha_i} \right] (s_i' - s_i).$$

Interchanging the roles of s_i' and s_i, we obtain a symmetric inequality. Thus

$$|V_\rho(s_i) - V_\rho(s_i')| \leq \rho^T \left[\frac{1-\alpha_i^{(T/P)+1}}{1-\alpha_i} \right] |s_i - s_i'|.$$

In order to prove 3), take $s_i, s_i' \in \Upsilon_i$ with $s_i' > s_i$. Consider processes $\{\Upsilon_i(t)\}$ and $\{\Upsilon_i'(t)\}$ generated by a common action sequence $\{a(t)\}$, where the two processes differ only by the initial state. One can merely verify that $\Upsilon_i'(t) \geq \Upsilon_i(t)$ for all t. Thus

$$\sum_{t=0}^{\infty} \rho^t v_i\big(\Upsilon_i'(t), a(t)\big) \geq \sum_{t=0}^{\infty} \rho^t v_i\big(\Upsilon_i(t), a(t)\big).$$

The claimed monotonicity then follows by taking the supremum of both sides of the latter inequality overall valid action sets.

To establish convexity, let $V_{\rho,T}(s_i)$ be the finite horizon discounted value:

$$V_{\rho,T}(s_i) = \sup_{\{a(t)\}} \left\{ \sum_{t=0}^{T} \rho^t v_i\big(\Upsilon_i(t), a(t)\big) : \Upsilon_i(0) = s_0 \right\}.$$

This satisfies the following dynamic programming equation:

$$V_{\rho,T}(s_i) = \max\{\lambda\gamma_i + \rho V_{\rho,T-1}(\alpha_i s_i + \upsilon_i), s_i + \rho V_{\rho,T-1}(\upsilon_i)\}$$

for all $T \geq 1$, with $V_{\rho,0}(s_i) = s_i$. We can establish convexity of $V_{\rho,T}$ by induction on T. Since pointwise limits preserve convexity, the fact that $V_\rho(s_i) = \lim_{T\to\infty} V_{\rho,T}(s_i)$ implies that V_ρ is also convex. □

As discussed earlier, the reward is discounted over time. Now, let $\widetilde{V}_\rho(s) = V_\rho(s) - V_\rho(\upsilon)$, $x \in \Upsilon_i$. We can deduce that $\widetilde{V}_\rho(s)$ also satisfies Lemma 1. Furthermore, $(1-\rho)V_\rho(u)$ is bounded. Using the Bolzano-Weierstrass Theorem [51] and Arzela-Ascoli Theorem [10], we may pick a subsequence along which $\big(\widetilde{V}_\rho(\cdot), (1-p)V_\rho(u)\big)$ converges to (V, ξ). From 2.5 we have

$$\widetilde{V}_\rho(s) + (1-\rho)V_\rho(\upsilon) = \max\{\gamma_i\lambda + \rho\widetilde{V}_\rho(\alpha_i s + \upsilon), s\}. \tag{2.6}$$

As $\rho \to 1$ along an appropriate subsequence, (2.6) becomes

$$V(s) + \xi = \max\{\gamma_i\lambda + V(\alpha_i s + \upsilon), s\}, \tag{2.7}$$

which can be written as

$$V(s) + \xi = \max_{a\in\{0,1\}} \{as + (1-a)(\gamma_i\lambda + V(\alpha_i s + \upsilon_i)\}. \tag{2.8}$$

Now that we have derived the dynamic programming equation, we want to show that the value function increases monotonically, is convex, and that $V(s) = 0$. These conditions are easily verified since point-wise limits preserve convexity and monotonicity.

Also, we want to show that maximizing $V(s)$ in (2.8) is achieved by the optimal action and, correspondingly, that ξ is the optimal reward. To establish this, consider the following argument. Let $a^*(s)$ be the action that maximizes

$$\Big[as + (1-a)\big(\gamma_i\lambda + V(\alpha_i s + \upsilon_i)\big)\Big].$$

If multiple actions maximize the foregoing function then we can pick one of those actions arbitrarily. Under the condition $\big\{a(t) = a^*\big(s(t)\big) : t \geq 0\big\}$,

$$V\big(\Upsilon(t)\big) + \xi = v\big(\Upsilon(t), a(t)\big) + V\big(\Upsilon(t+1)\big).$$

Now, if we consider the average value of both sides over time, we get

$$\underbrace{\xi + \frac{1}{T}\sum_{t=0}^{T} V\big(\Upsilon(t)\big)}_{L} = \underbrace{\frac{1}{T}\sum_{t=0}^{T}\big[v\big(\Upsilon(t), a(t)\big) + V\big(\Upsilon(t+1)\big)\big]}_{R}. \tag{2.9}$$

As $T \to \infty$, ξ is the average reward per chosen control policy. For any other action set, we will have $L \geq R$ in (2.9) and hence ξ is greater than equal the average reward under a different set of actions. This implies the optimality of ξ.

Let us now define the set of states when we do not poll a sensor as well as the states when we do poll a sensor:

$$D^c = \{s \in S : \gamma\lambda + V(\alpha s + \upsilon) > s\}$$
$$D = \{s \in S : \gamma\lambda + V(\alpha s + \upsilon) \leq s\}.$$

If t_0 is when this sensor is first polled and if $t_0 < \infty$ ($t_0 = \infty$ is the "never poll" case), by using the optimal policy and iterating t_0 times with the optimal value function in (2.7), we may write the dynamic programming equation as

$$V(s) = (\gamma\lambda - \xi)t_0 + \left[\alpha^{t_0}s + \left(\frac{1-\alpha^{t_0}}{1-\alpha}\right)\upsilon - \xi\right].$$

If we use a different policy that is not optimal, then we will have

$$V(s) \geq (\gamma\lambda - \xi)t_0 + \left[\alpha^{t_0}s + \left(\frac{1-\alpha^{t_0}}{1-\alpha}\right)\upsilon - \xi\right].$$

Consequentially, we can write $V(s)$ as:

$$V(s) = \max\left\{(\gamma\lambda - \xi)t_0 + \left[\alpha^{t_0}s + \left(\frac{1-\alpha^{t_0}}{1-\alpha}\right)\upsilon - \xi\right]\right\}$$

where we maximize the reward over all action sequences. This implies that equation (2.7) has a unique solution.

2.4 Computing the Index/Priority

Now, we show that an index policy (or a dynamic priority policy) exists for the problem. (This is akin to Whittle's approach for the classic restless bandit problem.) To establish the existence of an index policy, we utilize the following facts:

- The value function is monotone;
- The value function is convex;
- The mapping from s to $s - V(\alpha s + \upsilon)$ is concave.

Consequently, as λ is varied from $-\infty$ to $+\infty$, the set D grows in a monotone fashion from the empty set to the entire state space S.

First, we will show that some corner cases can be ignored.

1. If $\upsilon^* \in D$, that is the optimal action at υ^* is not to poll the sensor, and the related cost is $\gamma\lambda$. Then, $\xi = \gamma\lambda$ and the optimal strategy is to not poll the sensor at any state. Thus, $D = [\upsilon, \upsilon^*]$ and $D^c = \emptyset$. The index would then be calculated as

$$\lambda \geq \lambda_\upsilon = \max_{s \in [\upsilon, \upsilon^*]} \big(s - V(\alpha s + \upsilon)\big)/\gamma$$

2. If $\upsilon \in D^c$, we have $0 + \xi = u + 0 \Rightarrow \xi = \upsilon$. This means that it is optimal

to poll the sensor when the reward is υ Also, $D^c = [\upsilon, \upsilon^*]$ and $D = \emptyset$. In this case λ should obey the following inequality:

$$\lambda \leq \lambda_l := \min_{s \in [\upsilon, \upsilon^*]} \big(s - V(\alpha s + \upsilon)\big)/\gamma.$$

What we have now is that the deterministic control policies $a(t) = 0$ and $a(t) = 1$ have cost $\gamma\lambda$ and υ respectively, and ξ must then satisfy two conditions:

- $\xi \geq \min(\gamma\lambda, \upsilon)$, and
- $\xi \geq \min(\gamma\lambda, \upsilon)$ when $\lambda \in (\lambda_l, \lambda_\upsilon)$ and λ_l and λ_υ are lower bound and upper bound, respectively. Also, D and D^c are non-empty.

There is also some $\upsilon^+ \in (\upsilon, \upsilon^*)$ where polling or not polling a sensor are equally good. In this case, υ^* increases with λ. We can obtain $g(x)$ as the inverse of this function. $g(x)$ increases with $x \in (\upsilon, \upsilon^+)$. $g(x)$ is, in essence, λ when polling or not polling a sensor are both suitable decisions.

Lemma 2. *The sets* $[\upsilon, \upsilon^+)$ *and* $(\upsilon^+, \upsilon^*]$ *correspond to* D *and* D^c *for some* $\upsilon^+ \in [\upsilon, \upsilon^*]$.

Proof of Lemma 2

Proof. Since V is convex, either

1. For some $\upsilon_2 > \upsilon_1$, $D = [\upsilon, \upsilon_1) \cap (\upsilon_2, \upsilon^*]$, or

2. For some $\upsilon^+, D = [\upsilon, \upsilon^+)$ and $D^c = (\upsilon^+, \upsilon^*]$.

But at $s = \upsilon^*$, the optimal action is to poll the sensor. Thus, $s^* \in D^c$ and hence only the second condition can hold. □

Corollary 1. *The function from s to $V(\alpha s + \upsilon)$ is monotonically non-decreasing over $[\upsilon, \upsilon^*]$.*

Let $\lambda = g(s)$ for some $s \in (\upsilon, \upsilon^*)$. Every time we poll the sensor, the corresponding state resets to υ. The optimal policy is then periodic: do not poll a sensor until the state enters D^c and then poll it. Finite perturbations in initial conditions do not impact long-term behaviour, so we assume without loss of generality that $s(0) = \upsilon$ (i.e., initial state). Define $\tau(s) = \min\{t : \Upsilon(t) \in D^c\}$, where s is the initial state. Then

$$\Upsilon\big(\tau(s)\big) = \big(1 - \alpha^{\tau(s)}\big)\upsilon^* \Rightarrow \tau(s) = \left\lceil \log_\alpha^+\left(1 - \frac{s}{\upsilon^*}\right)\right\rceil,$$

where

$$\log_\alpha^+(s) = \begin{cases} \log_\alpha(s), & s > 0 \\ 0, & \text{otherwise.} \end{cases}$$

In the long-run, the overall average cost will converge to the average cost

over one polling period. Thus,

$$\xi = \frac{\lambda\gamma(\tau(s)-1)+\Upsilon\big(\tau(s)\big)}{\tau(s)}. \tag{2.10}$$

Theorem 1. *The index of sensor S_i is*

$$g_i(s) = \frac{1}{\gamma_i}\left[\tau_i(s)\big((1-\alpha_i)s-\upsilon_i\big)+\left(\frac{1-\alpha_i^{\tau_i(s)}}{1-\alpha_i}\right)\upsilon_i\right]$$

where

$$\tau_i(s) = \left\lceil \log_\alpha^+\left(\frac{\upsilon_i-(1-\alpha_i)s}{\upsilon_i}\right)\right\rceil.$$

Note. We introduced the subscript i so that we can have independent indices for different sensors. Also, if a sensor is polled even once, the states $\{\Upsilon_i(t)\}$ are discrete after that with jumps every time step. The states depend on υ_i and α_i solely. For a sensor that is never polled, the states can be restricted to discrete values depending on $\Upsilon_i(0)$. If we restrict attention to such discretized states, the index can be reduced to

$$g_i(s) = \frac{1}{\gamma_i}\left[\tau_i(s)\big((1-\alpha_i)s-\upsilon_i\big)+s\right].$$

Proof. For state $s \in D^c$, we obtain $V(s) = s-\xi$ using (2.7). Also, using Lemma 2, for $s' = \alpha s+\nu \in D^c$, we can obtain $V(s') = s'-\xi$. Combining

these results with (2.7) and the definition of an index for restless bandits, we have

$$g_i(s) = \frac{(1-\alpha_i)s - v_i + \tilde{\xi}_i(s)}{\gamma_i}, \tag{2.11}$$

where $\tilde{\xi}_i$, due to (2.10), is the optimal policy cost when $\lambda_i = g_i(s)$.

Using (2.10) yields

$$\tilde{\xi}_i = \frac{1}{\tau_i(s)} \left\{ \gamma_i g_i(s)(\tau_i(s) - 1) + (1 - \alpha_i^{\tau_i(s)} \upsilon_i) \right\}, \tag{2.12}$$

where $\tau_i(s) = \left\lceil \log_\alpha^+ \left(\frac{\upsilon_i - (1-\alpha_i)s}{\upsilon_i} \right) \right\rceil$.

Using (2.12) in (2.11), we can solve a linear equation for $g_i(s)$. Thus:

$$g_i(s) = \frac{1}{\gamma_i} \left[\tau_i(s)\big((1-\alpha_i)s - \upsilon_i\big) + \left(\frac{1-\alpha_i^{\tau_i(s)}}{1-\alpha_i} \right) \upsilon_i \right].$$

□

Index Policy. At each integer multiple of the period, poll the sensors with the highest indices until the bandwidth constraint is violated. That is, at a decision epoch, calculate the indexes $g_1(s_1), \ldots, g_N(s_N)$, where s_i is the state of sensor $i \in [N]$ at the decision epoch, sort the sensors in non-increasing order of their indices, and activate the sensors with the highest indexes until the sum of sensor bandwidths γ_i exceeds the total available bandwidth

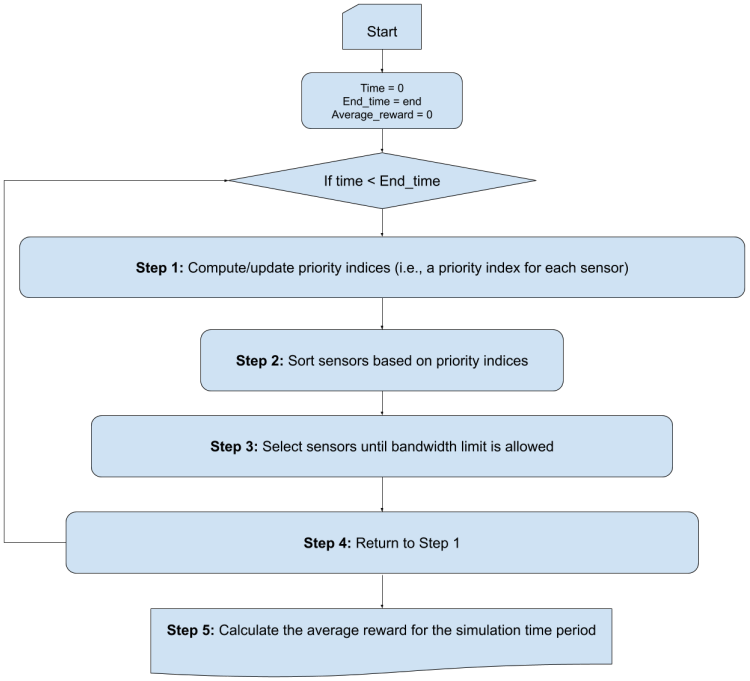


Figure 2.1: The flowchart of sensor polling process.

M. Figure 2.1 depicts the step by step flowchart of sensor polling process in our proposed approach. *Note that an alternative approach is to treat the decision at each epoch as a knapsack problem [17] after we have computed the indices but, as we discuss later, this approach does not result in significant benefits for the extra work involved.*

Computational Complexity: The Whittle-like index priority calculation in the previous section is very easy to compute and implement with only a linear increase in space and time complexity with the number of sources. Note that following the Whittle's approach, we decouple our problem into N sub-problems (one per sensor), each of which involves a constant-time

update. Therefore, the worst-case complexity of calculating indices is linear ($\Theta(N)$) in the number of resources. Once we have indices at each time step, the index policy requires that the indices be sorted ($\Theta(N \log N)$, in the worst case) such that it can select the best sensors. However, we need not perform a complete sorting at each epoch. There is a semi-periodic behaviour that we can exploit, and this allows us to select the top few sensors more efficiently in practice, often in sub-linear time.

2.5 Adaptive Estimation of Accrued Utility

In our initial analysis, we made the model of the utility accrued at sensor S_i during each period using (2.1), which we reproduce here for reference:

$$\upsilon_i = \sum_{j=1}^{M} q_j \mu_{i,j} \int_0^P \overline{\nu}_i e^{-\beta_i t}\, dt = \frac{\sum_{j=1}^{P} q_j \mu_{i,j}}{\beta_i} \overline{\nu}_i \left(1 - e^{\beta_i P}\right).$$

An insight that we can use into refining this estimate is as follows: suppose a sensor has not observed an event for t time units, what is the probability that this sensor not observe any event after $t + P$ time units? When events arrive with separation that is hyper-exponentially distributed, we can show – rather easily – not seeing an event can allow us to model the future with a *modified* hyper-exponential distribution. Let us suppose that t' is the inter-arrival time between two events.

$$
\begin{aligned}
P(t' > t+P | t' > t) &= \frac{P(t' > P | t' > t+P) P(t' > t+P)}{P(t' > t)} \\
&= \frac{P(t' > t+P)}{P(t' > t)} \\
&= \frac{\sum_{i=1}^{M} p_i e^{-\lambda_i (t+P)}}{\sum_{j=1}^{M} p_j e^{-\lambda_j (t)}} \\
&= \sum_{i=1}^{M} q_i e^{-\lambda_i (P)},
\end{aligned}
$$

where $q_i = p_i e^{-\lambda_i}(t) / \sum_{j=1}^{M} p_j e^{-\lambda_j (t)}$.

The derivation above illustrates that when we do not see arrivals under a hyper-exponential distribution, then we can make predictions using a modified hyper-exponential distribution.

We can use this insight as follows:

- If we poll a sensor and do not find an any useful data, then we can modify the arrival distribution. We can change our estimate for the expected utility in the next period according to the modified hyper-exponential distribution.

- If we do obtain useful data when we poll a sensor, we make our next estimate using the *original* hyper-exponential distribution associated with that sensor.

We also note that if we have multiple consecutive periods when a sensor does not produce useful data, then we can keep shifting the associated distribution based on the number of periods that have elapsed with no event.

One can comfortably accommodate this observation in the analysis we have shown so far. Therefore, we can derive an alternative index policy using this adaptive approach, and we denote this policy $\mathrm{IP_v}$ in our numerical study (Section 2.7) and compare it to the original policy that we denote $\mathrm{IP_f}$.

2.6 Explicit Analysis of Stochastic Arrivals

We now consider the case when the true discrete-event stochastic process governs observations at the sensors. In this case, the value of observations at a sensor may differ (and are unknown ahead of time), but the mean data value at each sensor is known.

Let $\{t_k^i\}$ represent the times at which sensor i records a new observation. The value of these observations is represented by $\{v_k^i\}$. The index k here is the observation count.

We assume that a sensor-dependent Poisson process governs the arrivals of observations. Further, we assume the observation values $\{v_k^i\}$ are also independent , and that they are identically distinguished for sensor i.

The value accumulated at source i during the jth epoch (between sampling instants $j-1$ and j) will be:

$$\upsilon_i(j) := \sum_{t_k^i:(j-1)P \le t_k^i < jP} v_k^i e^{-\mu_i(jP - t_k^i)}$$

The state of the system at $t = (j+1)P$ is then:

$$\Upsilon_i(j+1) = a_i(t) = \begin{cases} \alpha_i X_i(j) + \upsilon_i(j+1), & \text{if not polled} \\ \upsilon_i(j+1), & \text{if polled.} \end{cases}$$

The average expected reward can be defined as

$$\limsup_{t\to\infty} \sum_{i=1}^{N} \frac{1}{t} \sum_{m=0}^{t} \mathbb{E}\big[r\big(\Upsilon_i(m), a_i(m)\big)\big],$$

and we want to maximize this function subject to the cost/bandwidth constraint

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{i=1}^{N} r_i \mathbb{E}[a_i(t)] = M.$$

For the immediate discussion, we will not use the index i. We can focus on any one sensor and explicitly use i later, as needed.

$$V_\rho(s) := \sup_{\{V(t)\},\Upsilon(0)=s} \mathbb{E}\Big[\sum_{t=0}^{\infty} \rho^t V(\Upsilon(t), a(t))\Big]$$

is the discounted value function satisfying the following:

$$V_\rho(s) = \max\left\{\gamma\lambda + \rho\int V_\rho(\alpha s + \upsilon)\varphi(d\upsilon),\right. \tag{2.13}$$
$$\left. s + \rho\int V_\rho(\upsilon)\varphi(d\upsilon)\right\}.$$

In the expressions above, φ represents the function that governs $\upsilon(t), \forall\, t$.

Lemma 3. *The solution to* (2.13) *has the same properties as the solution to* (2.5)*:*

1. *It is unique and bounded;*
2. *It is continuous over* $\rho \in (0,1)$ *in the Lipschitz sense;*
3. *It increases monotonically and is convex.*

Proof of Lemma 3

Proof. Claim (i) can be derived from the standard theory of DTMCs, as stated before.

Claim (ii): Let $\Upsilon(t)$ and $\Upsilon'(t)$ be defined with identical processes for when the sensor records data and for the control (poll/not polled). With $a(\cdot)$ begin optimal for $\Upsilon(\cdot)$. Υ and Υ' only differ in their initial states, which are

s and s', respectively. Then,

$$\begin{aligned} V_\rho(s') - V_\rho(s) &\leq \mathbb{E}\left[\sum_{t=0}^{\infty} \rho^m \Big\{ V\big(s'(t), a(t)\big) - V\big(\Upsilon(t), a(t)\big) \Big\}\right] \\ &\leq \mathbb{E}\left[\frac{1-(\alpha\rho)^{t^-}}{1-\alpha\rho}\right](s'-s), \end{aligned}$$

where $t^- := \min\big\{t \geq 0 : \Upsilon(t) = \Upsilon'(t)\big\}$. t^- represents that instant at which we first poll this sensor. We can show that the value function is continuous as we did earlier.

To show claim (iii), we consider the processes $\big\{\Upsilon(t)\big\}$ and $\big\{\Upsilon'(t)\big\}$ generated by a common action sequence $a(t)$. Similar to the proof of Lemma 2, we will consider the supremum over all action sets but after taking expectations on both sides. This would establish monotonicity. Convexity follows in a fashion similar to the deterministic case.

□

Now, the dynamic program considering average costs can be written as

$$V(s) + \xi = \max\left\{\gamma\lambda + \int V(\alpha s + \upsilon)\varphi(d\upsilon), s\right\},$$

we can use the discounting approach from earlier and also make $V(\cdot)$ unique by forcing $\int V(s)\varphi(d\upsilon) = 0$.

We can show that $V(\cdot)$ is monotone and convex using pointwise limits.

When $f(\cdot)$ is convex, we have $s \mapsto \int f(bs+y)\omega(dy)$, for all bs and probability measures ω on $\mathbb{R}$.

We get

$$\mathbb{E}\big[V(\Upsilon(t))\big] + \xi = \mathbb{E}\big[V(\Upsilon(t), a(t))\big] + \mathbb{E}\big[V(\Upsilon(t+1))\big].$$

Using the above equation, we can then establish that:

$$a^{\star}(s) \in \operatorname{argmax}\left\{ as + (1-a)\left(\lambda + \int V(\alpha s + a)\varphi(d\upsilon)\right)\right\},$$

with $s \in S$, using a line of reasoning similar to the deterministic case.

Then, through iteration, we can show that V is the unique solution to the dynamic programming equation by representing $V(s)$ as follows. For passive s:

$$V(s) = \max \mathbb{E}_s\left[\big(\gamma\lambda - \xi(\lambda)\big)\theta + \alpha^{\theta} s + \sum_{t=1}^{\theta} \alpha^{\theta-t}\upsilon(t)\right] - \xi(\lambda) \tag{2.14}$$

$$= \max \mathbb{E}_s\left[\big(\gamma\lambda - \xi(\lambda)\big)\theta + \alpha^{\theta} s + \big(1 - \alpha^{\theta}\upsilon^{\star}\big)\right] - \xi(\lambda) \tag{2.15}$$

where θ is the time at which the sensor is polled for the first time. We take the maximum over all valid sequences of actions $\{\upsilon(t)\}$, where $\upsilon := \mathbb{E}\big[\upsilon(j)\big] + \upsilon^{\star} = \frac{\upsilon}{1-\alpha}$.

For active s, we have

$$V(s) = s - \xi(\lambda). \tag{2.16}$$

The second equality for $V(s)$ when s is passive is a consequence of the Optional Stopping Theorem (due to Doob) [38].

Now, considering only the interesting cases where $\xi(\lambda) > \min(\lambda\gamma, \upsilon)$, the r.h.s. of (2.15) will be less than $s - \xi(\lambda)$ for $s > \upsilon^\star$. This observation implies that $V(s)$ is smaller than the r.h.s. of (2.16), which suggests that s must have been active. Reasoning as we did in the deterministic or fluid-flow approximation situation, we can assume the state space as $[\upsilon, \upsilon^\star]$.

We can show that the optimal policy for polling sensors is a threshold policy exactly like in the deterministic case but with the appropriate change of definition for sets D and D^c.

$$D := \left\{ s \in S : \lambda\gamma + \int V(\alpha s + \upsilon)\varphi(d\upsilon) > s \right\}$$
$$D^c := \left\{ s \in S : \lambda\gamma + \int V(\alpha s + \upsilon)\varphi(d\upsilon) \leq s \right\}.$$

Lemma 4. *There is an index policy to solve the sensor selection problem with stochastic observations.*

Proof of Lemma 4

Proof. Consider $V(s) = \max \mathbb{E}_s\left[(\gamma\lambda - \xi(\lambda))\theta + \alpha^\theta s + (1-\alpha^\theta)\upsilon^\star\right] - \xi(\lambda)$.

The maximum is over every valid policy, and consequently, over every threshold policy too. Pick some threshold, and let the initial condition be $s \in [\upsilon, \upsilon^\star]$. Now, consider a process that uses the threshold policy. Then, θ is an r.v. That does not depend on λ. Now, $\xi'(\lambda) < \gamma$ so we can infer that the expression on the r.h.s. monotonically increases with λ. This property holds for when we maximize over every threshold-based policies. Let $s^\star(\lambda)$ be the optimal threshold with λ as the Lagrange multiplier (or subsidy) for passivity.

Now, (2.15) and (2.16) will continue to hold for $s = s^\star(\lambda)$.

Define $F(\lambda, s)$ as

$$F(\lambda, s) := \begin{cases} \max \mathbb{E}_s\big[(\gamma\lambda - \xi(\lambda))\theta + \alpha^\theta s + & \\ (1-\alpha^\theta)\upsilon^\star\big], & \forall s \in D, \\ s, & \forall s \in D^c. \end{cases}$$

Here the maximization is over all threshold policies. $F(\lambda, s)$ is a convex increasing function in s because V is convex and increasing in s. $F(\lambda, s)$ also increases with λ. $s^\star(\lambda)$ is a fixpoint of $F(\lambda, \cdot)$. The best action at $s = \nu$ is to be passive because $\xi(\lambda) > \upsilon$, hence $F(\lambda, \upsilon) > \upsilon, \forall \lambda$.

Next, we note that it is optimal to be active when $s = \upsilon^\star$. This gives us

$F(\lambda, \upsilon^\star) = \upsilon^\star$. The convex curve $s \mapsto F(\lambda, s)$ intersects $y = s$ at precisely one point in $[\upsilon, \upsilon^\star]$ and the intersection is at $s^\star(\lambda)$, by definition. This point increases with λ because $F(\cdot,\cdot)$ does.

Hence we conclude that an index policy must exist.

$\square$

Define $\overline{\tau}$ as the first time ≥ 1 when the state of an arm enters D^c with $\Upsilon(0) = u(0)$. This is the next polling time after 0.

$$\Upsilon(\tau) = \sum_{t=1}^{\overline{\tau}} \alpha^{\tau - t} \upsilon(t), \quad j \geq 1.$$

We can restrict our attention to stationary Markovian policies because of the underlying dynamic programming formulation.

Let $\lambda = g(s)$ be the index value at s. With $\lambda = g(s), \tau(s) = \mathbb{E}[\overline{\tau}]$, and $\overline{\xi}$ as the optimal set.

Using the standard theory for renewal-reward processes [62], we have

$$\overline{\xi}(s) = \frac{\lambda \gamma(\tau(s) - 1) + \mathbb{E}[\Upsilon(\overline{\tau}(s))]}{\tau(s)}.$$

The definition of $g(s)$ is that it is a subsidy needed to make an arm passive at s. Then, we can note that $s \in D^c$ and the best action would be to poll

the sensor. Using the definition of $g(s)$, $s = \gamma g(s) + \mathbb{E}\big[V\big(\alpha s + \upsilon(1)\big)\big]$, and therefore

$$\begin{aligned}
\gamma g(s) &= s - \mathbb{E}\big[V(\alpha s + \upsilon(1))\big] \\
&= s - \int \varphi(d\upsilon)\mathbb{E}_{\alpha s+\upsilon}\big[(\gamma g(s) - \overline{\xi}(s))\theta + \alpha^{\theta} x \\
&\quad + \sum_{t=0}^{\theta} \alpha^{\theta-t}\upsilon(t) - \overline{\xi}(s)\big] \\
&= s - (\gamma g(s) - \overline{\xi}(s)) \int \varphi(d\upsilon)\mathbb{E}_{\alpha s+\upsilon}[\theta] \\
&\quad + \int \varphi(d\upsilon)\mathbb{E}_{\alpha s+\upsilon}[\alpha^{\theta}]s - \overline{\xi}(s) + \\
&\quad \int \varphi(d\upsilon)\mathbb{E}_{\alpha s+\upsilon}\left[\sum_{t=0}^{\theta} \alpha^{\theta-t}\upsilon(t)\right].
\end{aligned}$$

From the equation above, we can solve for $g(s)$ by the observation that all the expectations are computable. To solve for $g(s)$, we could adopt the following computational procedure. Fix the threshold policy to use $\hat{s}$ as the threshold. V should satisfy the following conditions:

$$V(s) = \gamma\lambda - \xi + \int V(\alpha s + \upsilon)\varphi(d\upsilon), \quad s < \hat{s} \tag{2.17}$$

$$V(s) = s - \xi, \quad s > \hat{s} \tag{2.18}$$

$$V(u) = 0 \tag{2.19}$$

The index $g(\hat{s})$ must satify $g(\hat{s}) = \lambda$ and

$$\gamma\lambda + \int V(\alpha s + \upsilon)\varphi(d\upsilon) - s = 0,$$

which is derived from 2.18. We can write $V(s)$, the unique solution to (2.17), (2.18), and (2.19) to make an explicit connection to λ. We then learn $g(\hat{s})$ through a series of stochastic approximations:

$$g_{m+1} = g_m - b(m)\left[\gamma g_m + V\left(g_m + V(g_m, \alpha\hat{s} + \overline{\upsilon}_m) - \hat{s}\right)\right],$$

where g_m is forced towards (2.18) holding.

In this procedure, we assume that $\{\overline{\upsilon}_m\}$ are r.v.s that are independent and identically distributed according to distribution ϕ. This assumption leads to significant computational demands but can be reduced to relative value iteration algorithms where g_m is time-dependent [46].

g_m will, asymptotically, converge to the index. This computational approach is needed before each $\hat{s}$, but one could select a finite set of $\hat{s}$ values and then interpolate as a further approximation strategy.

Computational challenges with the index policy: When we consider the

fully stochastic nature of the sensing process, the computational complexity of the index policy is high and this approach, even though it is sub-optimal, may not be practical for large sensor deployments. Consequently, we can use the index policy derived using the fluid-flow approximation as a replacement. We applied the index policy that we first derived to the utterly stochastic scenario. Then, we compared our proposed index policy with the two other heuristics (greedy sensor selection and round-robin) that we have in our numerical evaluations (Section 2.7) and found that first index policy does outperform the other heuristics. We, therefore, believe that the fluid-flow approximation does preserve some of the essential problem characteristics and can yield satisfactory policies. We do not include this set of numerical evaluations in the next section because the results are similar to the other results we report.

2.7 Numerical Evaluation and Analysis

The index policies we propose are sub-optimal. The decomposition of the stochastic dynamic program into separate problems per arm results in the sub-optimality. On the other hand, we find that the index policies perform well in comparison to some other conceivable policies. We compare the two proposed policies (labelled as IP_v and IP_f) with another two policies:

- Greedy (GD): The greedy sensor selection strategy chooses the sensors that have highest value at the time of polling (epoch) until it

reaches the bandwidth limit.

- Round-robin (RR): The round-robin strategy selects sensors by turn until it reaches the bandwidth limit.

Our discussion here is restricted to the fluid-flow approximation (Section 2.4) since the application of the four policies (IP_v, IP_f, GD, RR) to the completely stochastic model yield similar results. We simulated our problem environment for our numerical evaluations in *python 2.7* on a PC with 8GB RAM, and core i5 intel cpu.

The greedy strategy chooses sensors based on their current accumulated reward (that is accumulated for each sensor since its last selection). The round-robin strategy selects sensors in turn and ignores accumulated values and other factors.

For the index policies (IP_v and IP_f), in general, we selected the arms with the highest indices until the bandwidth constraint was not exceeded. However, the hyper-exponential distribution varies in IP_v (unlike remaining fixed in IP_f) based on occurence of event(s) in each period of time (as discussed in Section 2.5).

To compare IP_v and IP_f with GD and RR, we carry out two types of experiments. In the first set of experiments, the bandwidth cost of each sensor/arm was kept identical (i.e. fixed); in the second, we selected these

costs at random (i.e. variable).

The other parameters in these experiments are chosen as follows:

- The value of a data item (v) at the sensor is selected from an exponential distribution with parameter 1.0. Note that when we use the fluid-flow approximation, all observations at a particular sensor provide the same value v_i at sensor i, and therefore we use the described distribution to select this value; the value of data may differ from one sensor to another.

- The rate at which the value of a data item decays (β) is chosen from a uniform distribution over $[0.01, 0.99]$. A different decay rate is selected for each sensor.

- The rate at which events arrive (μ) in each of the two distributions (in the hyper_exponential) is chosen from a uniform distribution over $[0.01, 25]$. Again, a different arrival rate is chosen for each sensor.

Workload intensity: We define a workload intensity metric as:

$$\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i \tag{2.20}$$

where M is the set of all messages arriving in a simulation run, $\mu_{arrival}$ is the arrival rate of events, and v_i and β_i are the value and decay rate for each message, respectively. We consider the decay rate as an approximation for

the deadline of each message. We could calculate this metric of "workload intensity" for each simulation over the total number of time steps. Such a metric essentially captures, to some extent, the workload intensity. Therefore, we report the performance of the policies concerning the "*average reward*" (on the Y-axis) along with the "*workload intensity*" (on the X-axis) when presenting the simulation results.

Evaluation setup		Simulations when IP_v/IP_f win				Average performance advantage			
#Bandits	Bandwidth	IP_v vs. RR	IP_f vs. RR	IP_v vs. GD	IP_f vs. GD	IP_v vs. RR	IP_f vs. RR	IP_v vs. GD	IP_f vs. GD
4°	2	98%	75%	100%	100%	12.1%	6%	15.6%	11.5%
8°	4	99%	85%	100%	100%	14.2%	9.1%	17.1%	12.3%
16°	8	99%	92%	100%	100%	15.6%	10.8%	17.3%	12.4%
32°	16	100%	98%	100%	100%	16.1%	11.8%	16.8%	12.4%
$4^\bullet$	2	89%	76%	97%	97%	11%	7%	22.1%	18%
$8^\bullet$	4	99%	89%	100%	100%	14.5%	10.2%	23.9%	20.1%
$16^\bullet$	8	99%	90%	100%	100%	16.2%	11.5%	25.7%	21.5%
$32^\bullet$	16	99%	93%	100%	100%	15.9%	11.3%	26.1%	21.7%

$\circ$ Bandits with fixed bandwidth costs.
$\bullet$ Bandits with various bandwidth costs.

Table 2.1: Comparison of algorithms in the cases of fixed and varied bandwidth costs. The table shows the percentage of simulation runs wherein the index policy outperforms other algorithms as well as the average performance advantage.

2.7.1 Identical bandwidth costs

First, we consider the simple case of equal sensor polling costs (γ=1 for all sensors).

We start with four bandit arms and a bandwidth limit of two ($M = 2$), which implies that we can pull two arms at any given epoch. We increase the number of bandits to 32 bandits and with a bandwidth of 16 to observe the performance of our algorithms at a slightly larger scale. We ran 1000 Monte Carlo trials for each policy and calculated the average reward attained by a strategy.

The experiments indicate that $\mathrm{IP_v}$ and $\mathrm{IP_f}$ dominantly outperform the other two algorithms (Figures 2.2, 2.3, 2.4, and 2.5). To be more specific, $\mathrm{IP_v}$ and $\mathrm{IP_f}$ always outperformed GD. However, in comparison with RR, both policies performed better in a majority of cases. $\mathrm{IP_v}$ outperformed RR in almost all cases (98% to 100%) with a performance margin of 12.1% to 16.1%. Also, $\mathrm{IP_f}$ outperformed RR in most cases (75% to 98%) with a performance margin of 6% to 11.8%. In both comparisons when RR did outperform IP, the performance difference was only 0.5%.

2.7.2 Varied bandwidth costs

In order to further evaluate our index-based approach, we considered the case of having different bandwidth costs among sensors/arms. We randomly assigned γ values to each sensor/arm; the value was selected from a

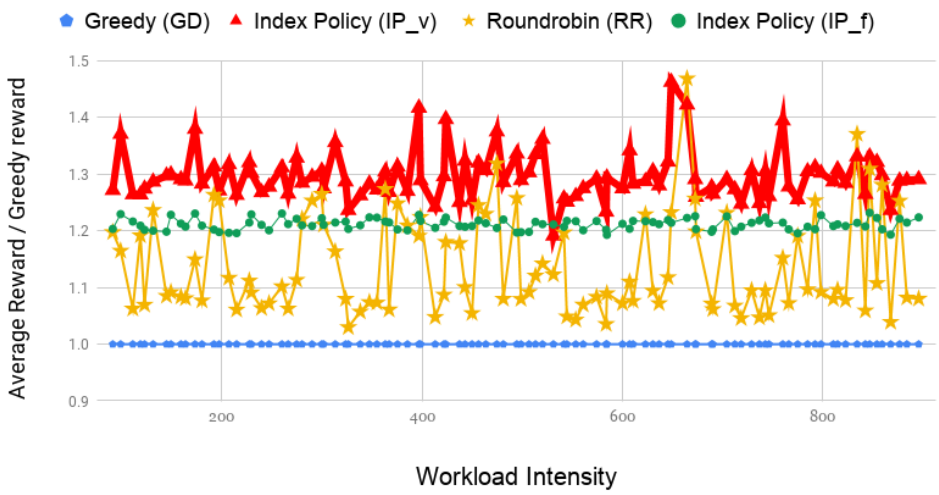


Figure 2.2: Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 4, Bandwidth Limit = 2. The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

uniform distribution between 0 and $M/2$, where M is the bandwidth limit. We did not see differences in performance relative to the earlier set of experiments with fixed bandwidth costs. Again, $\mathrm{IP_v}$ and $\mathrm{IP_f}$ almost always outperformed GD with higher performance margin with respect to the case of "*fixed bandwidth costs*" as shown in Table 2.1. In comparison with RR, both policies performed better in the majority of cases. $\mathrm{IP_v}$ outperformed RR in (89% to 99%) of the times with a performance margin of 11% to 15.9%. Also, $\mathrm{IP_f}$ outperformed RR in (76% to 93%) of the times with a

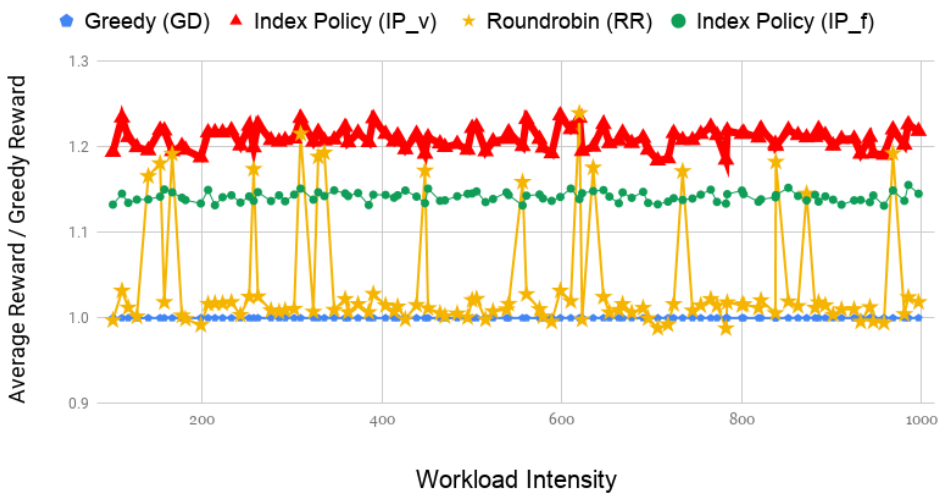


Figure 2.3: Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 8, Bandwidth Limit = 4. The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

performance margin of 7% to 11.3%. In both comparisons, when RR did outperform IP, the performance difference was less than 1.5%. The performance details of $\mathrm{IP_v}$ and $\mathrm{IP_f}$ relative to the other policies is also tabulated (second half of Table 2.1).

2.7.3 IP_v vs. IP_f

Besides the comparison results of our proposed policies to other policies, we compared $\mathrm{IP_v}$ and $\mathrm{IP_f}$ in terms of total rewards accrued over simula-

Figure 2.4: Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 16, Bandwidth Limit = 8. The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

tion runs. As implicitly implied from the above results in sections 2.7.2 and 2.7.1, $\mathrm{IP_v}$ outperformed $\mathrm{IP_f}$ in all simulation setups (as shown in Figure 2.10 and Figure 2.11). This observation suggests $\mathrm{IP_v}$ as the dominant policy.

2.7.4 Selecting arms using indices: top-k arms vs. knapsack packing

The index policy prioritizes the arms/sensors to be polled at an epoch. We can either select the arms with the highest indices until we exhaust that

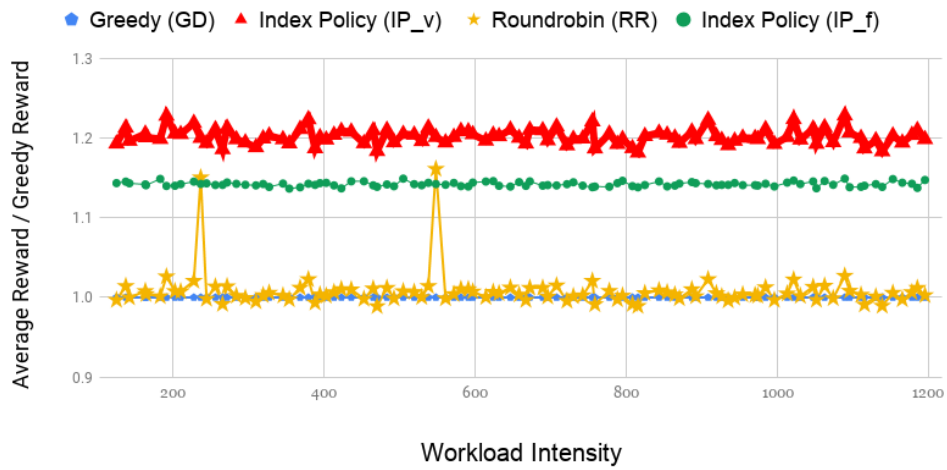


Figure 2.5: Performance of different policies with identical bandwidth costs: $\gamma = 1$ for all sensors where #Bandits = 32, Bandwidth Limit = 16 The proposed index policy typically outperforms greedy and round-robin sensor selection. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

bandwidth, or treat the problem as a knapsack [17]. We believed that there might be some gains in solving the knapsack problem (even though it is an NP-hard problem) and select arms. However, numerical results suggested that the more straightforward approach of selecting the arms with the highest indices performed well, and the more elaborate approach seems unnecessary. Note that we use IP_v for this comparison.

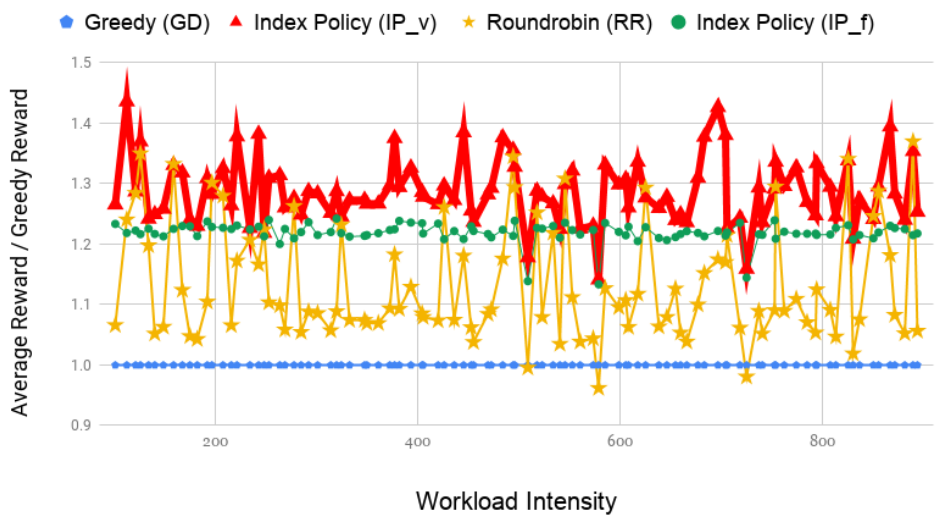


Figure 2.6: Performance of different policies with randomly selected bandwidth costs where #Bandits = 4, Bandwidth Limit = 2. The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

2.7.5 Insight from numerical evaluations

The main message that we want to communicate from our numerical evaluation is this: the index policy usually does better than the other policies, and when it does underperform another policy, the difference in performance is rather small. *These results suggest that IP_v is a policy that we can apply consistently and expect reasonable performance from.*

Figure 2.7: Performance of different policies with randomly selected bandwidth costs where #Bandits = 8, Bandwidth Limit = 4. The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

2.8 Related Work

We now position our work relative to prior research. Our work is closely connected with the work related to multi-armed bandits and, in particular, the model of restless bandits.

The model of restless bandits was presented by Whittle [69]. In the original formulation, there are many *arms* and pulling arm results in a reward.

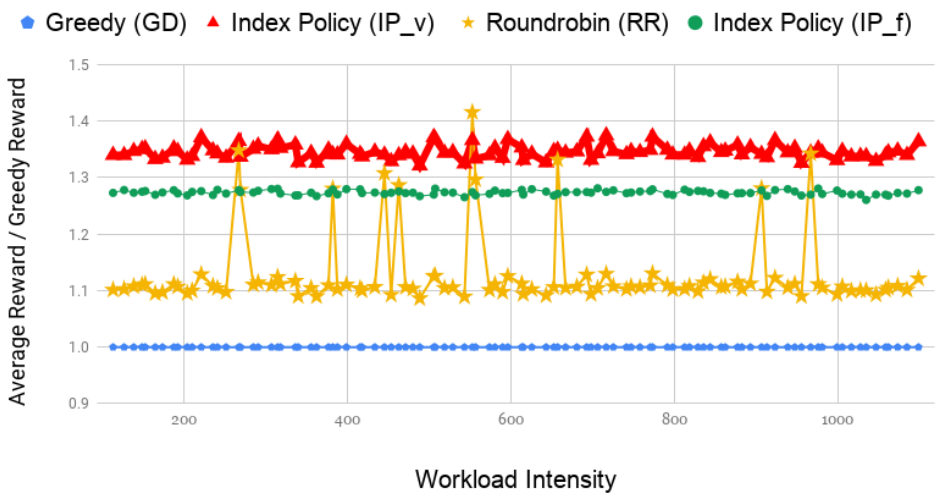


Figure 2.8: Performance of different policies with randomly selected bandwidth costs where #Bandits = 16, Bandwidth Limit = 8. The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

The underlying probability distribution for rewards is unknown and may change with time (hence the restlessness). The goal is to identify a policy that maximizes the long-term average reward. This model generalizes the multi-armed bandit model that was initially studied by Gittins and had a detailed presentation in a more recent monograph [21]. The solution approach is to define a priority/index computation that is efficient and helps determine the actions to take at each decision epoch.

Figure 2.9: Performance of different policies with randomly selected bandwidth costs where #Bandits = 32, Bandwidth Limit = 16. The index policy tends to outperform the greedy as well as the round-robin policy except for a few cases when round-robin selection has a small advantage. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

Most such problems can be tackled using *stochastic dynamic programming* [50], but such direct approaches suffer from the curse of dimensionality, leading to impractical solutions for a large number of sensors/bandit arms. The index approach is suboptimal–although asymptotically it is near-optimal–but avoids brute-force dynamic programming by reducing the problem to a set of more straightforward problems (one per arm). Prior work has focused on the state space of the arms being discrete, but, as we shall see, in our

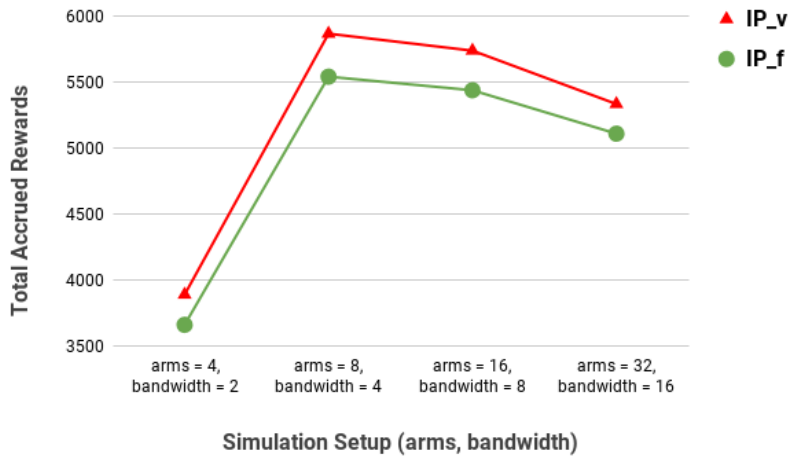


Figure 2.10: Comparison of IP_v and IP_f in terms of total accrued rewards with identical bandwidth. IP_v shows a consistent advantage with respect to IP_f in all simulations setups.

formulation, the state space is continuous (albeit closed) and requires that we establish the existence and suitability of priority/index policies.

Kleinberg and Immorlica have suggested the model of *recharging bandits* [31], wherein an arm that has not been played accrues rewards over time according to some concave function. The assumption of concavity in how an arm's rewards grow between pulls of the arm allows for a polynomial-time approximation scheme. The problem we study bears similarity with the work by Kleinberg and Immorlica, but that fact that a sensor's state may grow with time (as events are observed) and decay (as the data becomes stale) does not permit for the same analysis as in the case of recharging bandits. We may view the problem that we have presented as one involving *recharging-discharging bandits*, with the *recharge-only* model being a partic-

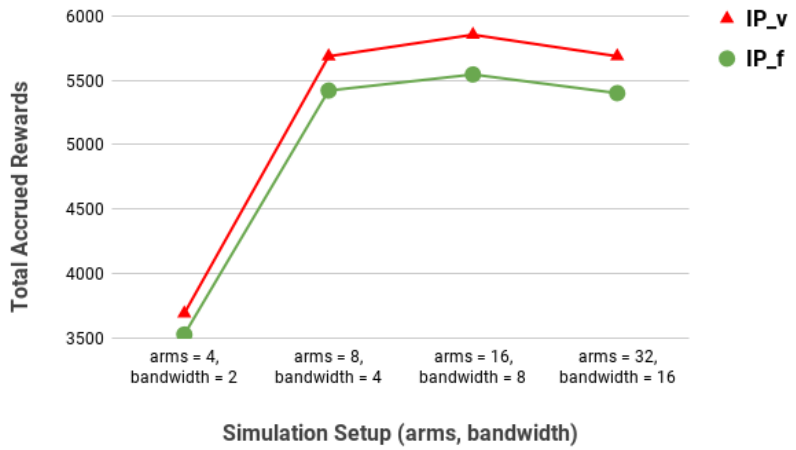


Figure 2.11: Comparison of IP_v and IP_f in terms of total accrued rewards with varied bandwidth. IP_v shows a consistent advantage with respect to IP_f in all simulations setups.

ular case. This difference seems to require the scheme we have discussed because the general restless bandit problem is PSPACE-Hard even to approximate [44] and work by Guha, Munagala and Shi [23] presents some approximation algorithms for some special cases. The work we present here is more general than the recharging bandits model but not as general as what was studied by Guha et al., and the results we present are relevant and interesting for a specific class of problems.

Sensor scheduling has been modelled as a restless bandit problem but with different constraints and objectives [42]: to find specific elusive targets using imperfect sensors. Similarly, there has been an effort to use the restless bandit model for sensors with energy harvesting considerations [27]. Such work did not address the issues of data value and time sensitivity,

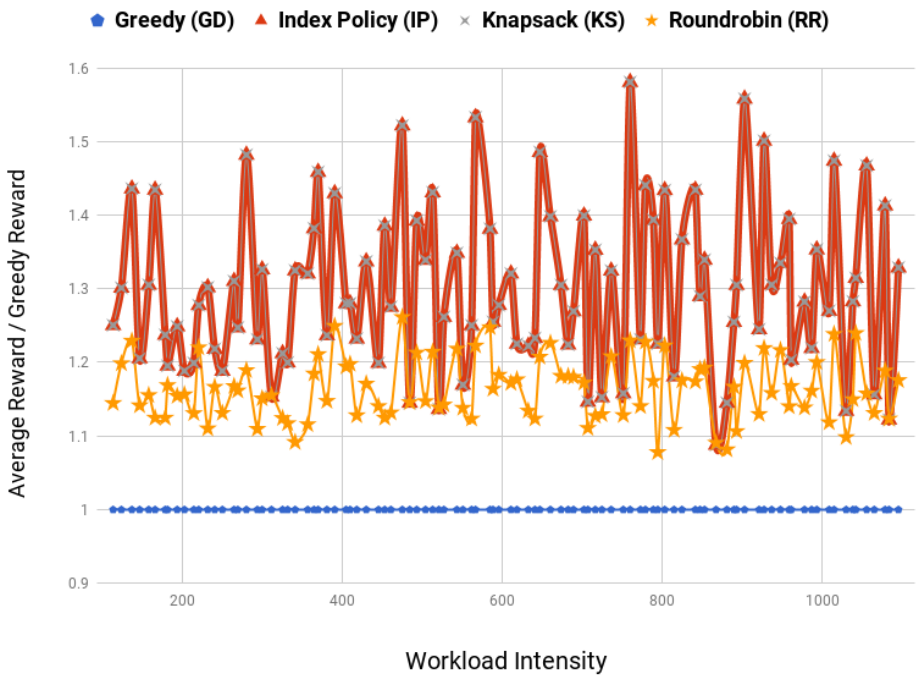


Figure 2.12: Comparing top-k selection with knapsack packing for index policy: We notice no significant difference between the two approaches so the top-k approach suffices. The Y-axis is the *Average Reward / Greedy Reward* (hence the Greedy policy always has the value of 1) and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} v_i \times \mu_{arrival} \times \beta_i$. According to the definition of the workload intensity, it increases with an increase in data value (v_i), an increase in data arrival ($\mu_{arrival}$) or an increase in decay rate (β_i) of messages.

which also require some analysis of the continuous state-space. Iannello and Simeone [26] studied the problem of optimally scheduling stochastically generated independent and time-sensitive tasks, where a centralized controller assigns at each time slot a node to a server for it to execute a task. The setting of their work is similar to ours in that a centralized decision-

making entity is assumed, task inter-arrival times are exponentially distributed, time-sensitivity is an explicit constraint, and the policy derived is a restless multi-armed bandit. One key difference, however, is that we consider continuous-time state dynamics, as opposed to the discrete-time state evolution model that the previous work considers. The consideration of continuous-parameter dynamics—and a continuous state-space—poses significant analytical challenges that are otherwise not present in discrete parameter/state-space models. In this model, sensors reinitialize state after some time steps were explored by Villar [66].

Optimal strategies for obtaining data from sensors with delay/freshness constraints have been studied in the context of a single sensor, to balance energy consumption with data freshness [19]. In these articles, a single sensor node was considered with the goal of maximizing a weighted function that accounted for sensor energy and data freshness. The problem of choosing which sensors to poll when there are multiple available sensors requires a different approach.

Heuristics for data collection have been explored in the IoT setting [25, 30], but these methods approach the problem from a broader system-building perspective, and the algorithms do not have proofs of optimality, or approximation ratios, or competitive ratios.

The sensor scheduling problem has been tackled recently in the specific

context of networked control systems, with the goal of state estimation. Weerakkody et al., for example, examine the multi-sensor scheduling problem to minimize mean squared error estimates [68]. Such work assumes specific knowledge of the underlying system state and its dynamics. Our formulation is relevant in situations where such system dynamics are not clearly defined, and the value of sensor data is measured by some exogenous processes such as data use. With similar information, Han et al. have also examined the stochastic sensor scheduling problem [24].

Clark et al. has used the general notion of adaptive delayed polling of sensors. [16], but not with near-optimal schedules.

2.9 Conclusions

We expect that as the deployment of the Internet of Things progresses, we will need to manage a massive data volume, and not all the data can be gathered in one place for processing. When some centralization is needed for data processing, sensor data selection is the first step in a pipeline of tasks that includes data management and analysis to support large-scale reasoning and decision making in a variety of applications [7]. We have concentrated on the first step alone; the amount of data produced by sensors alone can be overwhelming, and we need the right strategies for handling this deluge. Effective filtering of data in the early stages can reduce the pressure on other stages of IoT data processing systems that would

store and perform computation on the data.

Based on our analysis and evaluation, the index-based approach, which is computationally simpler than stochastic dynamic programming, is difficult to realize precisely when we consider the stochastic model for event arrivals and observation values. On the other hand, a fluid-flow approximation of the stochastic process may be sufficient to yield a simple and effective index policy for deciding which sensors to poll periodically. Although we poll sensors periodically, the set of sensors polled at each epoch may change and, depending on the underlying parameters; we find that an atomic structure may not exist for how this set of sensors changes from one epoch to the next.

We have chosen to model a polling approach and restricted our attention to the problem of deciding which sensors to poll at each epoch. Such polling systems are easy to implement, simplify system architecture, and can be suitable for satisfying specific data privacy requirements.

To conclude, our focus in this work is on the abstraction of polling sensors where we have some notions of delay sensitivity and limitations (such as privacy sensitivity as discussed in Section 2.1) that prevent a push-driven architecture and we showed that our proposed approach is effective in any such applications.

We have not answered the question of what is an ideal polling period. The

answer to this question will also depend on the characteristics of what is being sensed. We could remove the restriction of periodic polling and allow for adaptive polling intervals. This problem needs further study, although we believe that such adaptivity will lead to more fragile system architecture.

We want to emphasize, as we conclude, that sensors need not be physical sensors but could also represent feeds on services such as Twitter. The approach we present can be adapted to a variety of applications.

Concerning potential future directions, we envision multiple avenues to explore:

- Can we generalize this work to other distributions that govern arrival time? One idea would be to come up with multiple solutions and switch among a set of possible solutions appropriately based on the situation. For instance, one could explore coming up with a similar solution (i.e., deriving a priority index) for various assumptions concerning arrival distributions. On top of such solutions, we may be able to effectively select the best solution depending on the most recent trend of arrival times.

- How can we design strategies when at least some of the strategy (e.g., distributions) need to be learnt online? This situation will require that one incorporates parameter estimation as part of the solution. One

could investigate the feasibility of modelling the problem as an online learning problem over the environment. In this broad class of problems, agents aim to learn the true value of a parameter, often called the underlying state of the world. The state could represent a product, an opinion, a vote, or a quantity of interest in a sensor network. It is interesting to verify if such an approach could be applied to our problem with slight modifications of the model. For instance, if we assume each sensor has more resources available (e.g., to communicate with other sensors), we may consider each sensor as an agent. In this case, each agent observes feedback about the underlying state at each period (or the value sensed data) and communicates with her neighbours to augment her imperfect observations and dynamically learn about the environment. Since each agent aims to minimize its loss, comparison of each agent's loss at the end of each polling epoch could be equivalent to a priority policy.

Chapter 3

Balancing Message Criticality and Timeliness in IoT Networks: A Q-Learning Approach

Summary. We study the problem of balancing timeliness and criticality when gathering data from multiple sources using a two-level hierarchical approach. The devices that generate the data transmit them to a local hub. A central decision-maker then has to decide which local hubs to allocate bandwidth to, and the local hubs have to prioritize the messages they transmit when allowed to do so. Whereas an optimal policy does exist for this problem, such a policy would require global knowledge of messages at each

local hub, rendering such a scheme impractical. We propose a distributed reinforcement-learning-based approach that accounts for both the timeliness requirements and criticality of messages. We evaluate our solution using a *criticality-weighted deadline miss ratio* as the performance metric. The performance analysis is done by simulating the behaviour of the proposed policy as well as that of several natural policies under a wide range of system conditions. The results show that the proposed policy outperforms all the other policies – except for the optimal but impractical policy – under the range of system conditions studied and that in many cases it performs close (3% to 12% lower performance depending on the condition) to the optimal policy.

3.1 Introduction

With the proliferation of devices that can gather and transmit data about our world, we are confronted with the challenge of collecting and processing this data. More specifically, many such devices that we may deploy to observe and control aspects of our physical environment will use wireless communication links, and the available bandwidth for data transmissions can easily be saturated. To operate within these limits, we can perform a significant amount of data processing on the device that observes the data *and* we have to prioritize what data we choose to transmit. Some local processing is always possible, but certain decisions may require input from devices that are scattered quite widely in physical space. Therefore the

data from many devices will need to be collected at some centralized (or semi-centralized) location for joint processing.

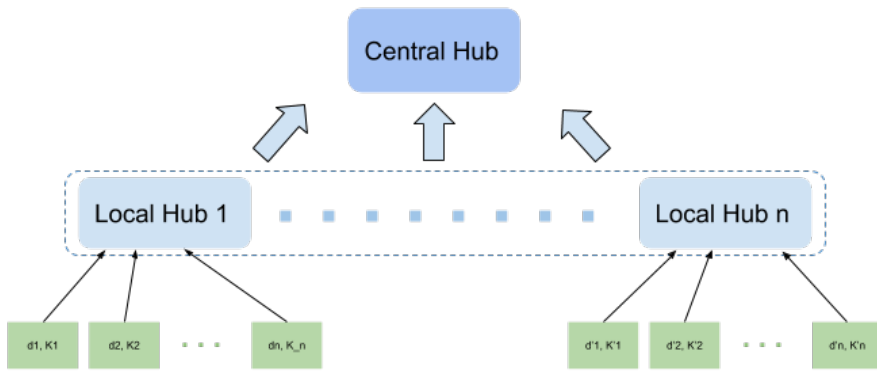
We discuss how we can prioritize and gather data from multiple devices when the data may have different timeliness and criticality requirements. In the system architecture, we consider, data is collected in a two-level setup: devices may communicate with a local hub, and local hubs communicate with a global entity. Our approach uses observations of system behaviour and reinforcement learning to identify suitable scheduling decisions. We present the mathematical analysis and simulation-based evaluation of scheduling policies for this problem.

3.2 System Model

We consider a system with hierarchical network architecture. At the lowest level are IoT devices such as sensors. These devices communicate with a local hub, which is a device that mediates communication with a central hub. We assume that the communication between the IoT devices and the local hub is not bandwidth-limited and that the local hubs use any suitable multiplexing mechanism to send and receive messages from the IoT devices. Local hubs collect messages that need to be transmitted to the central hub. They need to decide on how to prioritize these messages for transmission to the central hub.

We consider messages of unit length (equal-sized messages). We use the

Figure 3.1: The hierarchical model where unit-length messages with deadlines (d_i) and criticalities (κ_i) arrive at local hubs and then transmitted to the central hub (based on a policy).



following notation to describe a message and its characteristics: Message M_i has a relative deadline of d_i and a criticality value of κ_i. If a message is created at time t, then the absolute deadline for delivery of this message is $t + d_i$.

At the local hub, messages are assigned priorities from a fixed set $\{1, 2, \ldots, P\}$, with P representing the highest priority level.

At each scheduling epoch, the central hub selects one (or more) local hubs that can use the available bandwidth to transmit messages. A local hub will then transmit messages from its queues, starting with the highest priority queue that is not empty and moving to lower priority queues when higher priority queues are empty.

Performance Metric In the model that we have described, the system goal is to minimize the criticality-weighted deadline miss ratio that is defined as follows. Let N be the total number of messages generated during the time interval of interest that also have deadlines within that time interval. Let x_i be an indicator variable for whether message M_i missed its deadline or not. The criticality-weighted deadline miss ratio is

$$\rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j}. \tag{3.1}$$

With this performance metric, we can then state the problem that we want to solve.

Problem Statement We want to determine a priority assignment policy for use at the local hubs and a bandwidth allocation policy at the central hub to minimize the criticality-weighted deadline miss ratio for the online scenario. We do not know what messages will arrive until they arrive, and decisions need to be made for each new message.

3.3 Optimal Offline Policy

The problem of message prioritization can be solved efficiently, and optimally, when messages are all of equal length and information about all the messages is available at a central location. This offline and centralized policy is not realistic for two reasons: (i) In practice, messages arrive and

need to be prioritized as they arrive, and (ii) the cost of centralizing decision making imposes a high overhead on system operation. Nevertheless, we present the scheduling algorithm for this case because the performance of this approach is an upper-bound on the performance that any online and decentralized approach can achieve. We refer to this policy as OP.

Consider an offline setting, where there is a list of messages that need to be scheduled for transmission. Message M_i has criticality value κ_i, a relative deadline d_i, and unit length. The goal is to develop a scheduling policy (or algorithm) that schedules/ selects messages to get processed while minimizing the metric ρ (Section 3.2).

Formally, we can state the problem as follows:

- **Input**: (d_1, κ_1), (d_2, κ_2), ..., (d_n, κ_n)
- **Output**: Schedule $S = \{S(1), S(2), \ldots, S(i), \ldots, S(n)\}$ where $|S| \leq n$:
  - $S(i) = j$ means that the message M_i is scheduled in time slot j;
  - Any message is scheduled at most once;
  - If $x_i = 1$ when M_i misses its deadline and $x_i = 0$ otherwise then $\rho := \frac{\sum_{i=1}^{n} x_i \kappa_i}{\sum_{j=1}^{n} \kappa_j}$ is minimized.

In the offline scenario, minimizing ρ is equivalent to maximizing the criticality sum of the messages that meet their deadlines.

The proposed greedy algorithm (i.e., Algorithm 1) solves this problem optimally.

Algorithm 1 Optimal Greedy Policy

```
sort messages in non-increasing order of criticality values: κ_1 ≥ ... ≥ κ_n
for t ← 1 to n do
    S(t) ← 0
for i ← 1 to n do
    if there is any time slots left before d_i then
        Schedule M_i in the latest free slot that is possible before d_i
    else
        skip message M_i
```

Theorem 1. *Algorithm 1 maximizes the sum of criticalities of messages that meet their deadlines.*

Proof. We use induction for the proof as follows:

- **Base Case**: To see that $P(0)$ holds, consider any optimal schedule S_{opt}. Clearly, S_{opt} extends the empty schedule using only messages from $\{1,...,n\}$. So let $0 \leq i < n$ and assume $P(i)$ holds. We want to show $P(i+1)$. By assumption, S_i can be extended to some optimal schedule S_{opt} using only messages from $\{i+1,...,n\}$.

- **Induction Step**: Suppose that S is promising, and let S_{opt} be some optimal schedule that extends S. Let S_{i+1} be the result of one more iteration through the loop where message M_{i+1} is considered. We must prove that S_{i+1} continues to be promising, and therefore the

goal is to show there is an optimal schedule solution that extends S_{i+1}. Hence, we consider the following two cases:

- **Case 1**: Message M_{i+1} cannot be scheduled, so $S_{i+1} = S_i$. Since S_{opt} extends S_i , we know that S_{opt} does not schedule message M_{i+1}. Therefore, S_{opt} extends S_{i+1} using only messages from $\{i+2,...,n\}$.

- **Case 2**: Message M_{i+1} is scheduled by the algorithm, say at time t_0 (so $S_{i+1}(t_0) = i+1$ and t_0 is the latest free slot in S_i that is $\leq d_{i+1}$).

  * **Case 2-I**: Message M_{i+1} occurs in S_{opt} at some time t_1 (where t_1 may or may not be equal to t_0). Then $t_1 \leq t_0$ (because S_{opt} extends S_i and t_0 is as large as possible) and $S_{opt}(t_1) = i+1 = S_{i+1}(t_0)$.

    If $t_0 = t_1$, we are finished with this case, since S_{opt} extends S_{i+1} using only messages from $\{i+2,...,n\}$. Otherwise, we have $t_1 < t_0$. Say that $S_{opt}(t_0) = j \neq i+1$. Form S'_{opt} by interchanging the values in slots t_1 and t_0 in S_{opt}. Thus, $S'_{opt}(t_1) = S_{opt}(t0) = j$, and $S'_{opt}(t_0) = S_{opt}(t_1) = i+1$. The new schedule S'_{opt} is feasible (since if $j \neq 0$, we have moved message M_j to an earlier slot), and S'_{opt} extends S_{i+1} using only messages from $\{i+2,...,n\}$. We also have $P(S_{opt}) = P(S'_{opt})$, and

therefore S'_{opt} is also optimal.

* **Case 2-II**: Message M_{i+1} does not occur in S_{opt}. Define a new schedule S'_{opt} to be the same as S_{opt} except for time t_0, where we define $S'_{opt}(t_0) = i+1$. Then S'_{opt} is feasible and extends S_{i+1} using only messages from $\{i+2,...,n\}$. To finish the proof for this case, we must show that S'_{opt} is optimal.

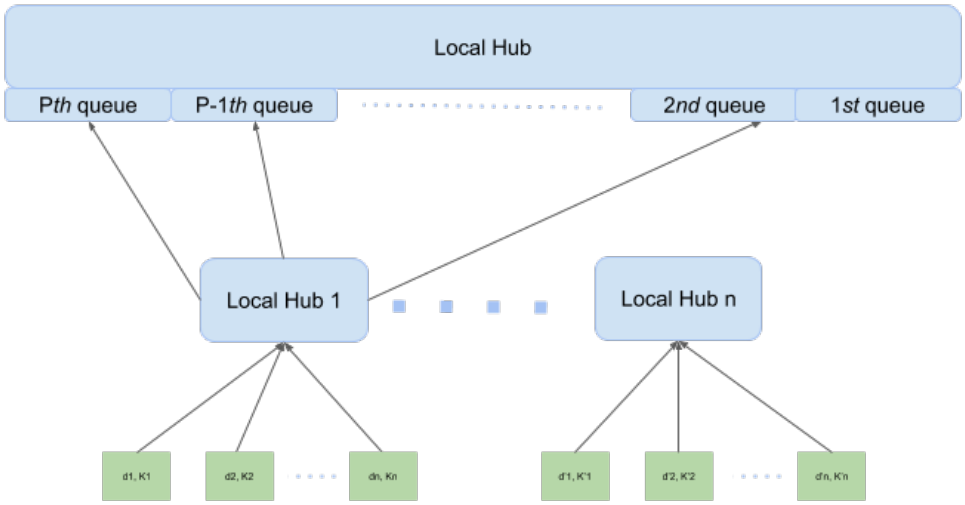
  If $S_{opt}(t_0) = 0$, then we have $P(S'_{opt}) = P(S_{opt}) + g_{i+1} \geq P(S_{opt})$. Since S_{opt} is optimal, we must have $P(S'_{opt}) = P(S_{opt})$ and S'_{opt} is optimal. So say that $S_{opt}(t_0) = j$, $j > 0, j \neq i+1$. Recall that S_{opt} extends S_i using only messages from $\{i+1,...,n\}$. So $j > i+1$, so $g_j \leq g_{i+1}$. We have $P(S'_{opt}) = P(S_{opt}) + g_{i+1} - g_j \geq P(S_{opt})$. As above, this implies that S'_{opt} is optimal.

□

Running Time The initial sorting can be done in $\Theta(n \log n)$ time in the worst case, and the remaining two loops take $\Theta(n)$ time in the worst case. Therefore, Algorithm 1 runs in $\Theta(n \log n)$ time in the worst case.

Although this greedy algorithm is an optimal polynomial-time algorithm, it requires complete and centralized information about all messages in the system. This centralization will impose a high overhead, and therefore we seek decentralized solutions to the problem at hand.

Figure 3.2: The details of processing messages at local hubs. First, sensors detect events and send messages to local hubs. Second, messages are routed to appropriate queues based on the priority assigned to each messages. Messages in the highest priority queue, are emptied first before messages at queues with lower priority level.



3.4 Using Reinforcement Learning in a Decentralized Policy

Given the two-level model that was described in Section 3.2, the central hub and local hubs have to make decisions on message transmission. Therefore, all hubs need to have policies for making such decisions. Here we elaborate on a policy that uses reinforcement learning to achieve near-optimal performance.

3.4.1 At Local Hubs

For each message, as shown in Figure 3.2, the local hub has to decide which queue to place the message. We assume that there are P priority levels $\{1,2,\ldots,P\}$, each of which is associated with a queue. Messages in the highest priority queue, associated with priority level P, are emptied first before messages at priority level $P-1$ are transmitted, and so on. The local hub needs to decide on an action $a \in \{1,2,\ldots P\}$ for each message where a represents the queue a message is assigned to.

The state s_t of a local hub is the state of each of the queues at time t. We will use $n(s_t)$ to represent the number of messages queued at time t. A decision made to assign a message to a queue changes the state of the system.

We can represent the value of a specific message M as a function of its deadline and criticality:

$$v_M = \kappa_M{}^{\beta} \left(\frac{1}{d_M}\right)^{\alpha}. \tag{3.2}$$

The message value is related to its criticality proportionally. In other words, the higher the criticality, the higher is the value. The message value is also related to its relative deadline (shorter the relative deadline higher the value).

α and β are parameters that help a system architect achieve a balance

between criticality and timeliness. These may be adapted on a per-local-hub basis, but in this discussion, all local hubs use the same settings for these two parameters.

We define the cost of taking action a_t at time t for new message M, when the system is in state s_t, as follows:

$$c(a_t, s_t) = \frac{\sum_{M \in s_{t+1}} w(M)v(M)}{n(s_{t+1})}. \tag{3.3}$$

We use s_{t+1} to indicate the state we reach after taking action a_t at state s_t. $w(M)$ is the weight associated with each message, which represents the absolute priority of M at the local hub. For the highest priority message in the system (the message at the head of the highest priority queue with any message in it), $w(M) = n(s_t)$ at state s_t and $w(M)$ for the lowest priority message (at the end of the lowest priority queue with any message in it) is 1.

$c(a_t, s_t)$ can be interpreted as the (weighted) average cost of a message queued at a local hub after taking action a_t.

Priority assignment. An incoming message is assigned a priority p^* based on the immediately perceived cost and the future cost associated with the decision:

$$p^* = \arg\min_{p \in P} E[c(s_t, a(p)) + \gamma Q(s_{t+1})]. \tag{3.4}$$

$Q(s_t)$ is the *value function* for the local hub in state s_t and we use $a(p)$ to denote the action of assignment priority p to the message.

Updating the value function. As each local hub evolves by the arrival and priority assignment of new messages, the value function (i.e., Q) gets updated:

$$Q(s_t) = Q(s_t) + \delta[c(s_t, a_t) + \gamma Q(s_{t+1}) - Q(s_t)], \tag{3.5}$$

where γ is the discounting factor and δ is the learning rate.

3.4.2 At the Central Hub

We assume that at the start of each scheduling epoch, the central hub gathers a snapshot of the state of the local hubs. This snapshot, for local hub L at time t, is the value $Q(s_t^L)$ and the number of queued messages, $n(s_t^L)$ (as explained in Section 3.4.1). This information is relatively small in size compared to the actual messages.

The central hub computes a weight for local hub L as follows:

$$w_L(s_t) = Q(s_t^L) n(s_t^L) \sum_{M_i \in T_L} \kappa_i, \tag{3.6}$$

where T_L is the set of messages that L can transmit in the one-time slot. If m messages can be transmitted in the one-time slot, then T_L is the set of the

m highest priority messages at L. For the rest of this discussion, we assume that a local hub transmits only one message in a time slot, but our work can be extended to situations when multiple messages can be transmitted in the same time slot.

The central hub selects the local hub with the highest weight. We call this policy at the central hub, the **Value-Weighted Policy (VWP)**.

3.5 Alternative Policies at the Central Hub

Having described the policy, based on reinforcement learning, that we have proposed, we discuss other heuristics that we can use to compare with our proposal.

Recall that the performance goal is to reduce the criticality-weighted deadline miss ratio. To this end, we can consider four other decentralized policies. In each of these policies, the local hubs assign priorities to messages using reinforcement learning (Section 3.4.1), but the central hub uses a simple heuristic.

1. Deadline-Greedy (DG): The central hub then selects a local hub using the earliest deadline among the highest priority messages at each local hub. Ties can be broken arbitrarily.

2. Criticality-Greedy (CG): The central hub then selects a local hub using the highest criticality among the highest priority messages at each

local hub. Ties can be broken arbitrarily.

3. Criticality Density Greedy (CDG): We define the criticality density of message M_i as $\frac{\kappa_i}{d_i}$. The central hub selects the local hub that has the message of the highest criticality density.

4. Random (RA): The central hub selects a local hub at random.

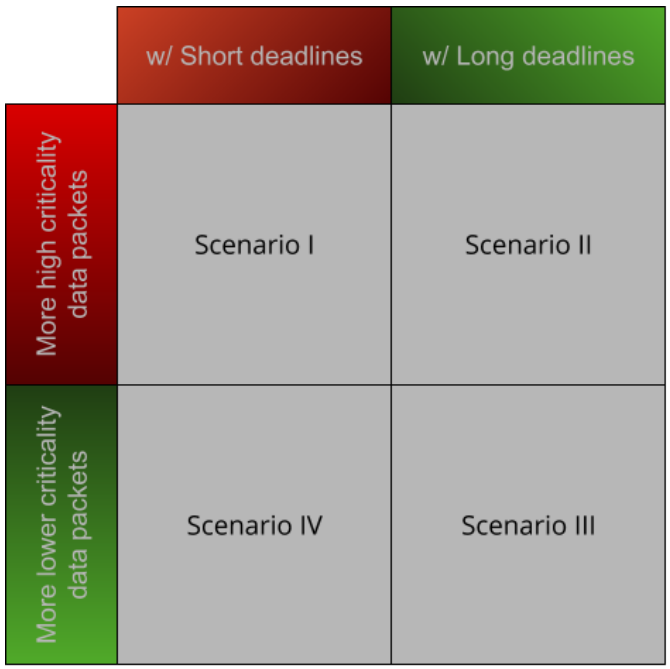
A fifth alternative policy that we consider, which is entirely centralized, is the Global Random (GRA) policy. We assume that the central hub has information about all messages at all the local hubs and selects a message to be transmitted at random from a uniform distribution. This policy is impractical because it is centralized, but we mention it here as another policy to compare to our proposed policy.

3.6 Quantitive Evaluation

We evaluate the policies via simulation. We set up a system with eight local hubs and a central hub, and each local hub has three priority queues ($q = 3$) representing *low, medium*, and *high* priority levels. We assume each queue has a capacity of $p = 50$ messages. We model message arrivals according to a Poisson process with rate $\lambda = 0.15$.

We assume that the processing time to calculate the weight (w_l, as shown in Section 3.6) for each local hub, is negligible.

Figure 3.3: We use four evaluation scenarios based on varying the distribution of messages in terms of criticality and deadline values. Scenario V, not in the figure, is where each the deadline and criticality of each message is chosen uniformly at random from a range of values.



3.6.1 Experimental Parameters

There are four parameters involved in the system model. Two parameters are related to the value function (the Q function): the discounting factor (γ) and the learning rate (δ). The other two are local hub balancing parameters for the deadline (α) and the criticality (β).

We assume, in our evaluation, that each local hub uses the same choice of parameter values to balance timeliness and criticality. For all four param-

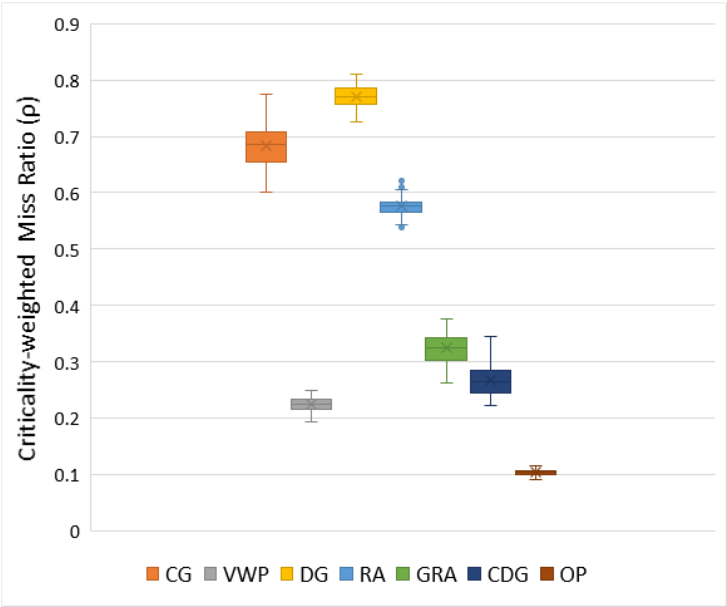


Figure 3.4: Range of criticality-weighted miss ratio (ρ) values for Scenario I. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.

eters, we assigned one of three values: low (0.1), medium (0.5) and high (0.9). This choice resulted in $3^4 = 81$ possible settings, and we simulated each setting for 1000 time steps.

3.6.2 Evaluation Scenarios

We have identified an optimal offline policy (Section 3.3) that provides an upper-bound on the performance of any of the decentralized policies. We compare the proposed policy (Section 3.4.1) and Section 3.4.2) with the five alternative heuristics (Section 3.5) as well as the optimal offline policy.

To understand the performance differences, we consider five different sce-

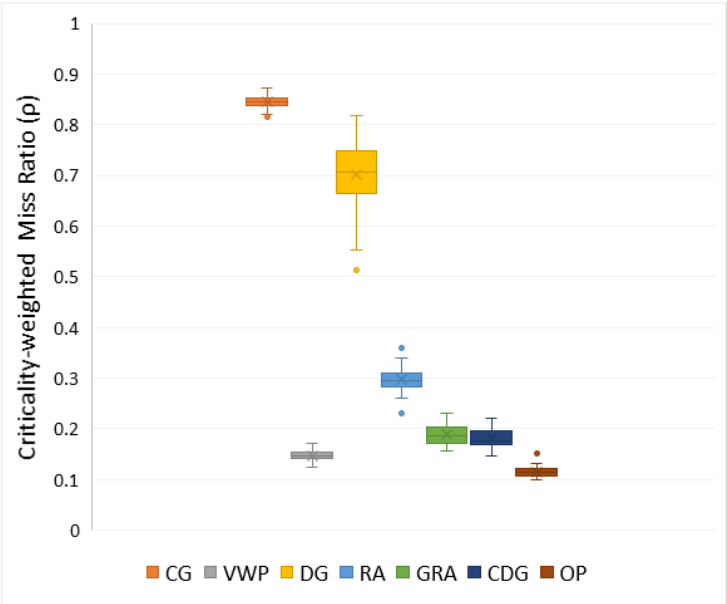


Figure 3.5: Range of criticality-weighted miss ratio (ρ) values for Scenario II. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.

narios concerning message deadlines and message criticalities:

- **Scenario I:** Majority of messages have *higher criticality* values and *shorter deadlines*.

- **Scenario II:** Majority of messages have *high criticality* values. On the other hand, those messages with *lower criticality* values have *shorter deadlines*.

- **Scenario III:** Majority of the messages have *low criticality* values. On the other hand, those messages with *high criticality* values have

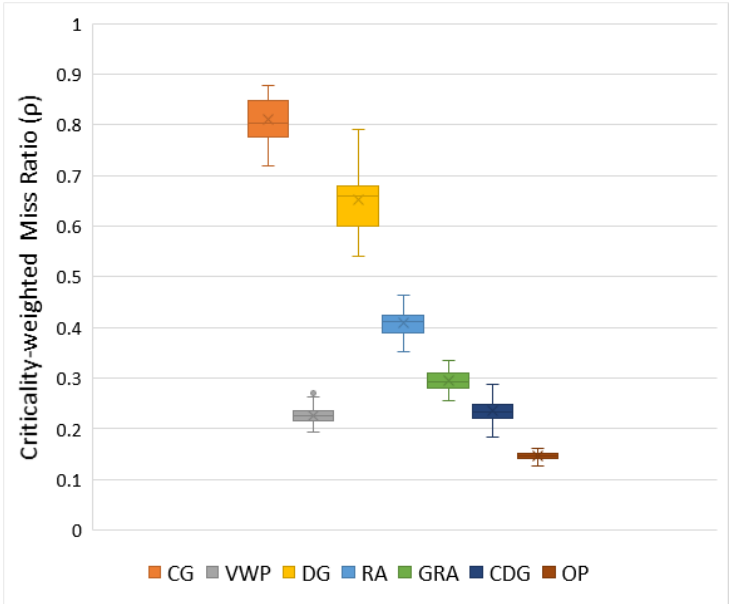


Figure 3.6: Range of criticality-weighted miss ratio (ρ) values for Scenario III. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.

shorter deadlines.

- **Scenario IV:** Majority of the messages have *low criticality* values. On the other hand, those messages with *low criticality* values have *shorter deadlines*

- **Scenario V:** For each message, the deadline and criticality values are chosen from a uniform distribution. Therefore, we chose deadlines of messages from the uniform distribution $U_d(1,50)$, and criticality values from the uniform distribution $U_\kappa(1,25)$.

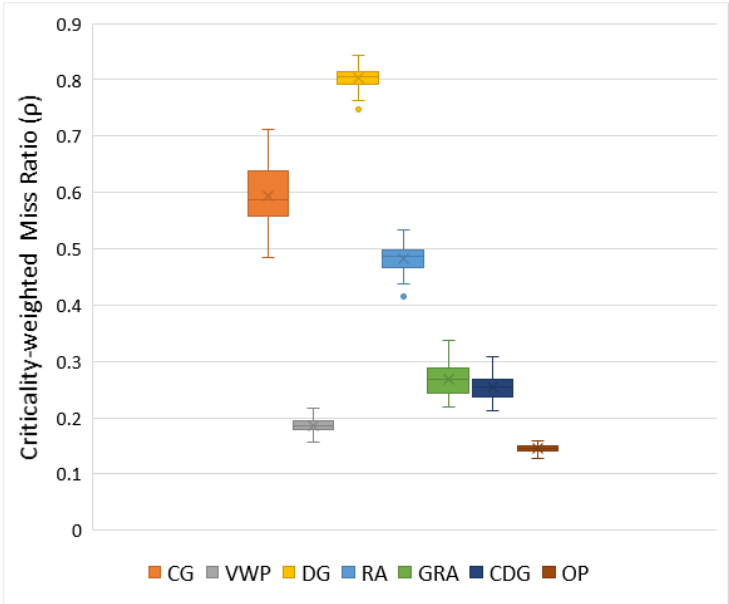


Figure 3.7: Range of criticality-weighted miss ratio (ρ) values for Scenario IV. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.

In these scenarios, we assumed $d_{short} = 25$ as the shorter deadline, and $d_{long} = 100$ as the longer deadline. Similarly, we considered $\kappa_{low} = 1$, and $\kappa_{high} = 2$ as the lower and higher criticality values, respectively. In Scenarios I to IV, we used a triangle distribution to generate 75% and 25% of messages from the specified majority and minority groups, respectively.

3.6.3 Results

Our observations, based on the results from the simulations (Figures (3.4, 3.5, 3.6, 3.7, and 3.8) and Table 3.1), can be broken down across the five scenarios we considered. In each scenario and for each policy, the ag-

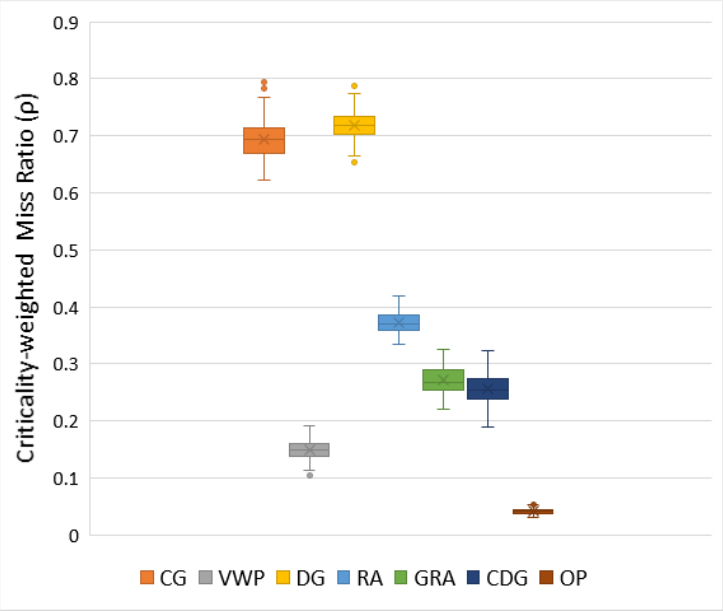


Figure 3.8: Range of criticality-weighted miss ratio (ρ) values for Scenario IV. The box plots indicate the range of miss ratio values using the same data reported in Table 3.1. The proposed reinforcement learning approach (VWP) is surpassed only by the offline optimal policy.

gregate results of the 81 settings are considered for the evaluations. *Across all scenarios, we find that the proposed policy competitively performs when compared with the optimal performance and outperforms all other policies. The proposed approach offers within 88% and 97% of the upper-bound on performance.*

We note that in Scenario II, the GRA and RA policies that pick local hubs uniformly at random perform relatively well because a majority of messages are of high criticality and with extended deadlines, which means that selecting messages at random also leads to reasonable performance.

Average Criticality-Weighted Miss Ratio ($\overline{\rho}$)							
Scenario	OP	VWP	CDG	DG	CG	RA	GRA
Scenario I	0.1046	0.2250	0.2679	0.7699	0.6840	0.5762	0.3239
Scenario II	0.1148	0.1475	0.1810	0.7004	0.8452	0.2972	0.1879
Scenario III	0.1460	0.2263	0.2467	0.6505	0.8094	0.4079	0.2938
Scenario IV	0.1443	0.1865	0.2559	0.8029	0.5943	0.4820	0.2680
Scenario V	0.0413	0.1488	0.2565	0.7171	0.6932	0.3721	0.2725

Table 3.1: The table shows the average Missed Criticality Ratio for the policies in each of the scenarios. As shown in green colored cells, the proposed policy (VWP) consistently performs as the second best policy after the optimal policy.

In Scenario III, CDG has a similar performance to our proposed policy. This result may reflect the fact that most low criticality messages have an extended deadline, so it may be useful to prioritize messages with a high criticality-density.

In Scenario IV, our proposed policy is significantly better than most other policies because there are more low criticality messages with short deadlines, and reinforcement learning adapts to this scenario, but other heuristics do not.

3.7 Related Work

Our research brings multiple areas such as Reinforcement Learning (RL), performance analysis, and job scheduling together. Therefore, the related work section is mostly about the area of scheduling and its relation to RL techniques, especially in performance analysis of distributed systems appli-

cations.

In distributed environments such as grid and cloud, job scheduling is considered an NP-hard problem given the problem setup [64]. Therefore, the optimization targets are around metrics such as makespan and load balance.

Lucas-Estan and Gozalvez have investigated load balancing for industrial IoT networks but without specific attention to messages with deadlines and criticalities [36]. Maguluri et al. [37] assumed to have unknown job sizes with the same criticality levels and optimized throughput using load balancing/scheduling algorithm. Shi et al. [55] developed an algorithm for provisioning and scheduling jobs under deadline constraints to address unpredictable issues in large-scale scientific computing. Seno et al. have developed soft real-time scheduling algorithms for industrial wireless networks with timeliness constraints alone [54]. Our work and these efforts relate to timing constraints, but we focus on striking criticality and overall timeliness in a distributed IoT-like setup.

Tordsson et al. proposed a scheduling and resource allocation method for cloud jobs based on particle swarm optimization, taking the scheduling deadline, and scheduling budget into account [63]. Classic heuristic optimization techniques such as parallel genetic algorithms have been often employed to solve the job scheduling problems in cloud computing [12, 56]

and particle swarm optimization algorithm [28, 72]. Although there has been a large amount of research done regarding job scheduling in the distributed computing environment, such work does not directly apply to the message scheduling problem in hierarchical networks.

The performance analysis of infrastructure-as-a-service (IaaS) cloud platforms has been studied extensively under various configurations and use cases. Although the environment that we target is different regarding resource constraints, we share similarities such as preserving service level agreement (SLA) that we refer to as a criticality-weighted miss ratio (Section 3.2). For instance, Salah et al. proposed an analytic model based on Markov chains to predict the number of cloud instances or VMs needed to satisfy a given SLA performance requirement such as response time, throughput, or request loss probability [52]. Khazaei et al. proposed a general analytic model for end-to-end performance analysis of a cloud service [29]. They illustrated their approach using the IaaS cloud with three pools of servers: hot, warm, and cold, using service availability and provisioning response delays as the key QoS metrics. The proposed approach reduces the complexity of performance analysis of cloud centers by dividing the overall model into sub-models and then obtaining the overall solution by iteration over individual sub-model solutions. Our work in this chapter is similar to that of Khazaei et al. [29] in the sense of having a hierarchy such that local hubs partially handle prioritization of messages.

Reinforcement learning has been used for scheduling for several applications of distributed computing. Peng et al. proposed an effective RL-based scheduler scheme for cloud computing under SLA constraint. Quan et al. [47] developed a two-layered RL method offload task. In this method, the first layer is in charge of selecting the appropriate cluster of machines, and the second layer selects a physical machine to execute the task. Chang et al. [13] tackled the data forwarding problem in Under Water Wireless Sensor Networks (UWSN) using an RL-based method that factors in the challenge of timeliness and energy constraints.

The model we have studied is different from the work on mixed-criticality scheduling that has been studied in the context of traditional real-time systems [11]. In the mixed-criticality real-time scheduling problems, the goal is to ensure that high criticality jobs always meet their deadline and the criticality-weighted deadline miss ratio is not of the metric of importance.

To the best of our knowledge, we believe that ours is the first attempt at applying reinforcement learning to the soft real-time scheduling problem with message deadlines and criticalities, and to minimize the criticality-weighted deadline miss ratio.

3.8 Conclusions and Future Work

We have demonstrated that a reinforcement learning approach to message scheduling in a hierarchical system with many nodes can help us strike a

balance between timeliness and criticality. The approach we have proposed outperforms many other heuristics that one could use in this context. We believe that these ideas can be of value in the context of the Internet of Things, with particular relevance to factory automation and medical systems.

We have tackled this problem in a specific setting where all messages are of equal (or unit) length. **We have shown that our solution has performance that is near-optimal by determining an upper-bound using an optimal offline algorithm.** The optimal offline algorithm for the problem we studied is a greedy centralized algorithm, which is not practical because the centralization will come with significant overhead.

As a future direction, one would like to understand the problem when messages can be of varying lengths. This change to the problem is significant because the offline problem (when we know all messages ahead of time) is NP-Hard and can be solved by a pseudo-polynomial time dynamic program formulation.

Chapter 4

Handling the Message Criticality vs. Timeliness Tradeoff in Complex IoT Environments: A Deep RL Approach

Summary. We study the problem of handling timeliness and criticality trade-off when gathering data from multiple resources in complex environments. We use the term "complex" for environments, where data resources (e.g., sensors) may be correlated in time or space. In IoT environments, where several sensors transmitting data packets - with various criticality

and timeliness, the rate of data collection could be limited due to associated costs (e.g., bandwidth limitations and energy considerations). Besides, environment complexity regarding data generation could impose additional challenges to balance criticality and timeliness when gathering data. For instance, when data packets (either regarding criticality or timeliness) of two or more sensors are correlated, or there exists temporal dependency among sensors, incorporating such patterns can expose challenges to trivial policies for data gathering. Motivated by the success of the Asynchronous Advantage Actor-Critic (A3C) approach, we first mapped vanilla A3C into our problem to compare its performance in terms of *criticality-weighted deadline miss ratio* to the considered baselines in multiple scenarios. We observed degradation of the A3C performance in complex scenarios. Therefore, we modified the A3C network by embedding long short term memory (LSTM) to improve performance in cases that vanilla A3C could not capture repeating patterns in data streams. Simulation results show that the modified A3C reduces the criticality-weighted deadline miss ratio from 0.3 to 0.19.

4.1 Introduction

Connected devices are part of IoT environments in which every device talks to other related devices - by gathering and transmitting data - to *timely* communicate *important/critical* sensor data to interested parties for further usage. With the enormous amounts of data generation, selecting only *important/critical/relevant* data to be *timely* used for various usages

is still an issue. Besides, environment complexity regarding data generation could impose additional challenges to balance criticality and timeliness when gathering data. We consider an environment to be "complex" when:

- Correlation exists between the arrival of events for two or more data resources/sensors.
- Temporal dependence exists among sensors due to the physical arrangement of devices.
- Correlation exists between (sensors) data.

For instance, when data or arrival of data for two or more sensors correlate, or there exists temporal dependency among sensors, incorporating such patterns can expose challenges to trivial policies for data gathering. Examples of environments with such attributes could be a network of sensors for capturing marine temperature, where there is temporal dependence among data [33] or intelligent buildings where deployed sensors generate data that is temporally and spatially dependent [14].

Capturing patterns such as temporal dependency or correlation on the data arrival of sensors could help to collect data more efficiently concerning the desired goal. For example, consider a network of four sensors (S_A, S_B, S_C, and S_D) where at each time step, we can collect data from only one sensor. The goal is to collect data packets in such a way that we have the highest

accumulated criticality values over a time horizon, where each data packet is characterized by a pair of criticality and deadline value (e.g., data of sensor S_i is referred as (Cr, d)). As an example (shown in Table 4.1 and Table 4.2), imagine that there is a temporal dependency between sensors S_A and S_B due to their close location. In case of not capturing the dependency mentioned above (detailed in Table 4.1 at the time step t, we have data from sensors S_A and S_C, where data at S_C has a higher criticality value and deemed as the better choice for transmission despite having larger deadline value compared to S_A. However, other data packets will be expired by the time that we want to collect data again at step $t+1$, and the total criticality value collected at the end of $t = 4$ would be 8 (equivalent to criticality-weighted deadline miss ratio of $\frac{13-8}{13} = \frac{5}{13} = 0.38$). Alternatively, as shown in Table 4.2, by knowing and capturing the temporal dependency between sensors S_A and S_B, one could transmit data at S_A since a data packet will arrive for S_B soon after S_A. So, the total amount of criticality for the transmitted data packets from S_A and S_B can be as large as the criticality of data at S_C (if it was alternatively selected), and we may be able to collect data of S_C still if it is not expired yet. In this case, the total criticality value collected at the end of $t = 4$ would be 11 (equivalent to criticality-weighted deadline miss ratio of $\frac{13-11}{13} = \frac{2}{13} = 0.15$).

Time step	Current available data	Selected (sensor) data	Total accumulated criticality
t=1	S_A: (3,1), S_B: (-), S_C: (5,5), S_D:(-)	S_C: (5,5)	5
t=2	S_A: (-), S_B: (3,2), S_C: (-), S_D:(2,1)	S_B: (3,2)	5+3 = 8
t=3	No available data (all expired and no new arrivals)	-	8
t=4	No available data (all expired and no new arrivals)	-	8

Table 4.1: Example of sensor selection in a 4-step time window, when dependency **cannot** be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor i is shown as $S_i = (cr, d)$. As the table shows, the total accumulated criticality at the end of the time step $t = 4$ is 8.

Time step	Current available data	Selected (sensor) data	Total accumulated criticality
t=1	S_A: (3,1), S_B: (-), S_C: (5,5), S_D:(-)	S_A: (3,1)	3
t=2	S_A: (-), S_B: (3,2), S_C: (5,4), S_D:(2,1)	S_B: (3,2)	3+3 = 6
t=3	S_A: (-), S_B: (-), S_C: (5,4), S_D:(-)	S_C: (5,3)	3+3+5 = 11
t=4	No available data (all expired and no new arrivals)	-	11

Table 4.2: Example of sensor selection in a 4-step time window, when dependency **can** be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor i is shown as $S_i = (cr, d)$. As the table shows, the total accumulated criticality at the end of the time step $t = 4$ is 11.

We discuss *how we can prioritize and gather data from multiple devices when the data may have different timelines and criticality requirements - especially in complex environments*. We use the term "complex" to refer to scenarios where the sensors may be correlated in time or space. Such correlations make sensor polling decisions harder. Given the challenges in capturing dependencies among sensors, and motivated by the success of Deep Reinforcement Learning (DRL) in policy estimation, we explore leveraging DRL techniques to tackle our problem. DRL techniques have shown to outperform alternative methods (e.g., traditional Q-learning) in terms of handling large state spaces, which is a requirement for deploying large IoT networks. Also, DRL techniques can handle both continuous and discrete state spaces. We propose an approach based on the *Asynchronous Advantage Actor-Critic (A3C)* [39] by improving the network structure such that it captures the likely case of having temporal dependency and correlation within data streams. We achieved this improvement by adding an LSTM layer to consider some previous states (rather than only one state) to learn recurring patterns within data. Access to previous states implies the notion of embedding memory into the model. Besides the addition of memory to the model, the A3C itself offers benefits. It outperforms some other RL alternatives (e.g., Q-learning methods based on Q-table and DQN) concerning resource requirement and time performance [39]. We provide the formal presentation of the problem as well as the simulation-based evaluation of scheduling policies for this problem.

4.1.1 Context

The Internet of Things (IoT) refers to the vast number of things (i.e., electronics-infused devices) connected to the internet, which acts as sensors in their hosting environment, generating massive volumes of data. In such settings, everything from data acquisition to processing and analysis can leverage Machine Learning (ML) techniques to preserve efficiency and performance. As all the steps (i.e., data acquisition, processing, and analysis) mostly involve decision making of some sort, ML techniques would help in making informed decisions by capturing patterns in data and sensors behaviour. In a sense, the integration of ML into the IoT world would transform simple sensor-actuator devices into ambient intelligent devices.

In different application settings such as factory automation, hospital, and more environments [48], the notions of message criticality and timeliness co-exist. Imagine the case of factory automation, where sensory messages that require immediate actions compete with less urgent messages (i.e., no need for instant actions) though being critical. In such cases, there needs to be a decision-maker as to act upon sufficiently reasonable concerning the desired objective function.

While the ultimate goal is to deploy applications that can identify changes such as the one described above in the environment and appropriately adapt to them, sometimes such changes could occur in a more challenging way:

- There could be cases that correlation exists between the arrival of events for two or more sensors. For instance, in a factory, temperature and pressure sensors may generate data mostly at about the same time, different from optical sensors.

- Furthermore, there could be cases where *temporal dependence* exists among sensors due to the physical arrangement of devices. For instance, imagine that two optical sensors (sensor A and sensor B) installed in separate locations in a factory. Sensors A and B may observe the same event, but at different points in time.

- It is more challenging to capture the *correlation* among sensor data. In the context of factory automation, there could be cases where the criticality/timeliness of messages for some sensors correlate with some other sensors. This situation may happen because of reasons such as related functionalities of the operating devices there. For example, two temperature sensors that are installed in the same factory warehouse sense correlated temperature values, whereas their sensed data may not correlate with temperature data of sensors in other warehouses.

From a system architecture perspective, we consider a network of devices (i.e., sensors) that sense events in the environment. There is a central unit managing the transmission of data from sensors by selecting a sensor at

a time. Sensors communicate their sensing data to the central unit upon selection. The system that we study deals with the challenge of possessing minimal resources (e.g., in terms of energy and memory) as a likely setting for many networks ([15, 18, 67]), which emphasizes the essence of having an effective decision-making policy. For efficiency of energy and memory consumption, sensors would hold a limited number of messages. Such architecture makes it possible to be extended and creates clustered networks with multiple hierarchy levels. For instance, in a more extensive network, each central unit can operate as a relay device, such as the model described by Rashtian et al. [48]. In the scope of our work here, we consider a central unit responsible for deciding during each decision-making epoch.

4.1.2 Contributions

We propose and evaluate a scheduling mechanism for the described architecture where the environment adds complexity by causing correlations among messages and arrival dependency messages that need to be transmitted. We start by exploring the applicability of the A3C method [39] as a successful Deep Reinforcement Learning method and proposing our approach by improving upon that. The contributions of the work have four folds:

- Mapping the A3C approach to our problem and establishing a baseline method that showed consistent performance in less complex scenarios.

- Showing that the A3C performs no better than the greedy baselines in complex scenarios.

- Embedding memory to the network in the vanilla A3C to improve the performance in complex scenarios with high dependencies.

- Showing that based on the simulations, the modification to A3C did not negatively affect the performance in other scenarios where the vanilla A3C had already performed well. Such observation confirms the advantage of the proposed solution over the vanilla A3C in all studied scenarios.

4.2 System Model

We consider a system with a centralized architecture, as shown in Figure 4.1. The IoT devices (i.e., sensors) are communicating to a central unit. Sensors sense the environment events and capture them as messages. The central unit transmits one message at a time from one of the sensors.

We consider messages of equal-sized lengths arriving at sensors. For energy consumption and memory costs, we assume each sensor would hold one data message. Sensors have enough resources to maintain only one massage at the time. We use the following notation to describe a message and its characteristics: Message M_i has an applicable deadline d_i and criticality κ_i. If a message arrives at time t, then the absolute deadline for delivery

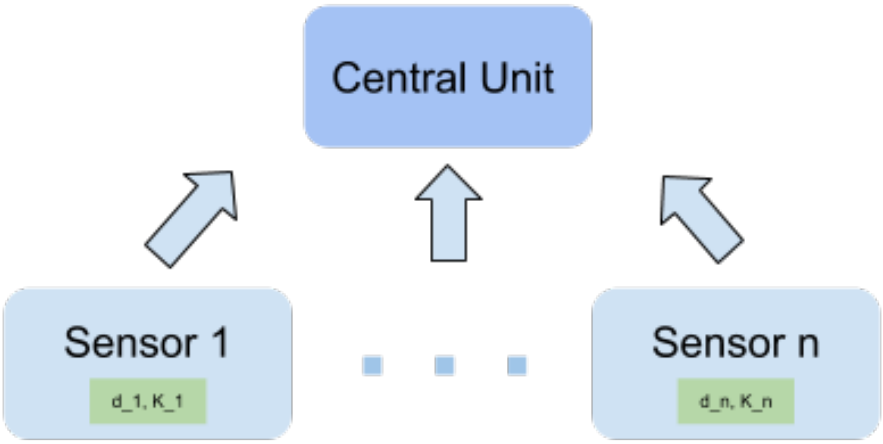


Figure 4.1: The system model where the central unit selects a sensor at a time to transmit data. Sensors hold a message with deadline (d_i) and criticality (κ_i). The model makes no assumptions about the arrival rate of events sensed by the sensors.

of this message is $t + d_i$. Until the message reaches its expiration time, it will provide its highest value if selected. If the message is selected after its expiry, it will provide a discounted value (as will be discussed in Section 4.14). If a new message arrived at the sensor, and the sensor already has a message, the older message would be replaced by the new message.

At each scheduling epoch, the central unit selects one sensor to use the available bandwidth to transmit its associated message. Also, no assumptions are being made about the event arrival rate.

4.2.1 Performance Metric

In the model that we have described, the system goal is to minimize the *criticality-weighted deadline miss ratio* that is defined as follows: Let N be

the total number of messages generated during the time interval of interest that also have deadlines within that time interval. Let x_i be an indicator variable for whether message M_i missed its deadline or not. The criticality-weighted deadline miss ratio is

$$\rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j}. \tag{4.1}$$

With such performance, we can define the problem that we want to tackle.

4.2.2 Problem Statement

The problem that we want to tackle is to determine an efficient bandwidth allocation policy at the central unit to minimize the miss criticality ratio over a finite time interval as the optimal policies may be intractable:

$$\min\{\rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j}\} \tag{4.2}$$

We do not know what messages will arrive until they arrive, and decisions regarding the selection of messages need to be made at each scheduling epoch.

4.3 Background

We have chosen to take advantage of Reinforcement Learning (RL) to tackle our problem. We provide a brief background on RL techniques in general, A3C, and later discuss our proposed method.

4.3.1 Overview of Deep Reinforcement Learning

In this section and before discussing the A3C network, we first introduce the basic concepts of RL and Deep Learning (DL), based on which Deep Reinforcement Learning (DRL) is defined.

Reinforcement Learning (RL)

RL is a class of algorithms in machine learning that can achieve optimal control of a Markov Decision Process (MDP) [32, 58, 70]. There are generally two entities in RL - an agent and an environment. The environment evolves in a stochastic manner within a state-space at any time. The agent operates as the action executor and interacts with the environment. When it acts within a particular state, the environment will generate a reward/penalty to indicate how good was the action taken by the agent. The agent learns from this generated response from the environment over time. The policy determines the strategy for an agent to take action when being in a state. The agent task is to learn from its already taken actions such that in the future, it will take actions that are optimal concerning the value function $V_\pi(s_0)$. We define the value function as the expected reward from

actions taken by a policy (π) over a finite time horizon:

$$V_\pi(s_0) = E_{\tau_{s_0} \sim \pi}[R_{total}(\tau_{s_0})], \tag{4.3}$$

where τ_{s_0} denotes a chain of states selected by adopting π policy starting from s_0. R_{total} is the reward accumulated from traversing such sequence of states.

Apart from value function, another important function is Q function $Q^\pi(s_0, a_0)$, which is the expected reward for taking action a_0 in state s_0 and thereafter following a policy π. When policy π is the optimal policy $\pi^\star$, value function and Q function are denoted by $V^\star(s)$ and $Q^\star(s,a)$, respectively. Note that $V^\star(s) = \max_a Q^\star(s,a)$. If the Q functions $Q^\star(s,a)$, $a \in A$ are given, the optimal policy can be easily found by $\pi^\star = \arg max_a Q^\star(s,a)$. In order to learn the value functions or Q functions, the Bellman optimality equations can usually help. Taking the discounted MDP with a discount factor of γ for example, the Bellman optimality equations for the value function and Q function is

$$V^\star(s_t) = \max_{a_t}[r_{t+1} + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^\star(s_{t+1})], \tag{4.4}$$

and

$$Q^\star(s_t, a_t) = r_{t+1} + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}), \tag{4.5}$$

respectively.

Bellman equations represent the relation between the value/Q functions of the current state and the next state.

Usually, a large amount of memory is required to store the value functions and Q functions. In some cases, when only small, finite state sets are involved, it is possible to store these in the form of tables or arrays. This method is called the tabular method. However, in most of the real-world problems, the state sets are large, sometimes infinite, which makes it impossible to store the value functions or Q functions in the form of tables. Therefore, the trial-and-error interaction with the environment is hard for learning the environment dynamics due to formidably computation complexity and storage capacity requirement. Even if we can learn the dynamics, it imposes massive consumption of computing resources. In this case, one can approximate some functions of RL such as Q functions or policy functions with a smaller set of parameters by the application of DL. ***The combination of RL and DL results in the more powerful DRL****.*

Deep learning (DL)

DL refers to a family of machine learning algorithms that leverage artificial neural networks (ANN) to learn patterns from a large amount of data. It can perform well in tasks like regression and classification. The weights and bias of every node in a neural network (NN) are the parameters of the

NN. Usually, a neural net with two or more hidden layers is called a Deep Neural Network (DNN). A loss function $L(\theta) = g(\hat{Y}(\theta), Y)$ is used in deep learning, which is a function of the output $\hat{Y}(\theta)$ from the network and the desired output (Y) . The loss function evaluates the performance of the desired NN (in terms of learning the corresponding parameters till this point) in terms of modelling the given data (i.e., $Y = f(X)$). Depending on the task type, various loss functions can contribute. For instance, the standard regression loss functions include Mean Square Error (MSE), Mean Absolute Error (MAE), Mean Bias Error (MBE). Concerning the classification task, loss functions such as Cross-Entropy loss and Support Vector Machine (SVM) loss perform well. After that, gradient descent methods are used to update θ parameters in NNs and consequently minimize the loss function. Given a loss function $L(\theta)$, the parameters get updated by a gradient method such as the simple gradient:

$$\nabla_\theta L(\theta) = \frac{\partial L(\theta)}{\partial \theta} \tag{4.6}$$

Such gradient descent methods start from an initial point of θ_0 . As the input data is fed to NN, the average loss function over all input data is calculated and used to minimize $L(\theta)$ by taking a step along the descent direction, i.e.,

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta), \tag{4.7}$$

where α is a hyper-parameter named step size and indicates how fast the parameter values move towards the optimal direction. The above process is repeated iteratively by inputing more data to NN until convergence.

Deep Reinforcement Learning (DRL)

As discussed earlier, DRL refers to a family of methods that combine RL and DL to approximate either Value or Q functions (or even both) via a deep NN. In general, the DRL approaches can be categorized into two main groups: *Value-based* and *Policy Gradient*.

In *Value-based* methods for DRL the states $s_t \in S$ or state-action pairs $(s_t, a_t) \in S \times A$ are inputs of NNs, while Q functions $Q^{\pi}(s_t, a_t)$ or value functions $V^{\pi}(s_t)$ are approximated by parameters θ of NNs. An NN returns the approximated Q functions or value functions for the input states or state-action pairs. There can be a single output neuron or multiple output neurons. For the former case, the output can be either $V^{\pi}(s_t)$ or $Q^{\pi}(s_t, a_t)$ corresponding to the input s_t or (s_t, a_t). For the latter case, the outputs are the Q functions for state s_t combined with every action, i.e., $Q^{\pi}(s_t, a^1)$, $\cdots$, $Q^{\pi}(s_t, a^{|A|})$.

In *Policy Gradient* methods, NNs can directly approximate a policy as a function of the state, i.e., $\pi^{\theta}(s)$. The states are used as inputs to the NNs, while policy π is approximated by parameters θ of NNs as π^{θ}. *In contrast to value-based DRL methods, the policy gradient methods for DRL is a direct*

mapping from state to action, which leads to better convergence properties and higher efficiency in high-dimensional or continuous action spaces [20]. Therefore, we chose to take leverage of policy gradient methods for our research problem. Specifically, we focused on A3C as the basis of our solution and improved upon it concerning our problem.

4.3.2 A3C Network

A3c networks consist of multiple independent agents (i.e., neural networks) with their weights, which interact with a different copy of the environment in parallel. Therefore, they can explore a more significant part of the state-action space in much less time. The agents (or workers) are trained in parallel and periodically update a globally shared neural network, which holds shared parameters. The updates are not happening simultaneously, and that is where the asynchronous notions come from. After each update, the agents reset their parameters to those of the global network and continue their independent exploration and training until they update themselves again.

We shall now briefly elaborate on *Asynchronous Advantage Actor-Critic (A3C)* as the underlying structure in our approach.

We define a value function $V(s)$ of a stochastic policy $\pi(s)$ (that returns dis-

tribution of probabilities over actions) as an expected discounted reward:

$$V(s) = E_{\pi(s)}[r + \gamma V(s')] \tag{4.8}$$

where $V(s)$ is the weight-average of $r + \gamma V(\prime(s)$ for every action that can be potentially taken in state $s \in S$.

We also define the action-value function $Q(s,a)$ as:

$$Q(s,a) = r + \gamma V(s') \tag{4.9}$$

where we emphasize that the action is given and there is only one following s'.

We define the advantage function as:

$$A(s,a) = Q(s,a) - V(s) \tag{4.10}$$

A(s, a) is the advantage function as it informs how good it is to take action a in a state s compared to the average performance. In case the action a is better than the average, the advantage function has a positive value. It gets a negative value when the action is worse than average.

Furthermore, we define ρ as the distribution of states, which indicates the probability of being in states. ρ^{s_0} and ρ^{π} denotes the distribution of begin-

ning states in the environment and states under policy π, respectively.

Since policy π is only a function of state s, we can approximate it directly. In this case, a neural network (with θ as the weights) would take a state s and output an action probability distribution π_θ. We shall use π and π_θ interchangeably as the policy parametrized by the network weights θ.

On the other hand, we want to optimize the policy. We define a metric function $J(\pi)$ as an averaged discounted reward that a policy π can accumulate over possible beginning states s_0:

$$J(\pi) = E_{\rho^{s_0}}[V(s_0)] \tag{4.11}$$

Now, we use the gradient of $J(\pi)$ to improve it. The gradient of $J(\pi)$ is derived in the *Policy Gradient Theorem* ([59, 61]) and has the following form:

$$\nabla_\theta J(\pi) = E_{s\sim p^\pi, a\sim\pi(s)}[A(s,a).\nabla_\theta \log \pi(a||s)] \tag{4.12}$$

where the first part (i.e., $A(s,a)$) informs the advantage of taking action a in state s. The second part of it (i.e., $\nabla_\theta \log \pi(a||s)$) informs a direction in which logged probability of taking action a in state s rises. Since both terms are together, the equation 4.12 increases that the likelihood of actions that are better than average performance while decreases the likelihood of actions worse than average performance. Since it is not feasible to compute

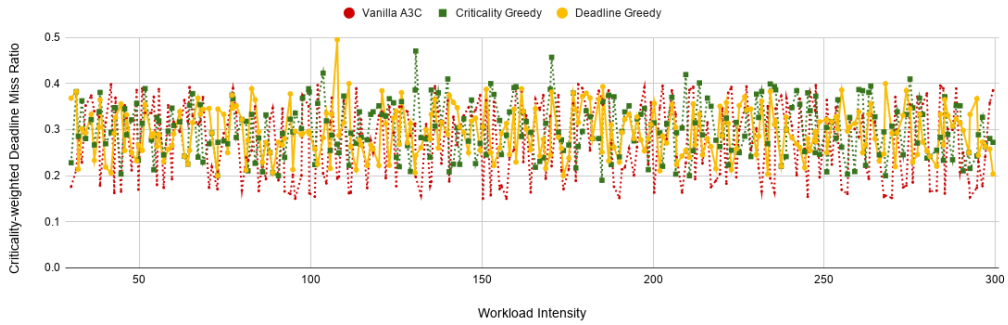


Figure 4.2: An example of A3C's limitations, where its performance degrades in complex scenarios with 8 sensors. The Y-axis is the *Criticality-weighted Deadline Miss Ratio* and the X-axis represents the *Workload Intensity* that we define as: $\sum_{i \in M} \frac{Cr_i \times \lambda_{arrival}}{d_i}$. According to the definition of the workload intensity, it increases with an increase in criticality (Cr_i), an increase in data arrival ($\lambda_{arrival}$) or a decrease in deadline (d_i) of messages.

the gradient over every state and every action, we use sampling for this computation (as the mean of samples lays near the expected value).

The advantage function also needs to be computed. Let us expand the definition as:

$$A(s,a) = Q(s,a) - V(s) = r + \gamma V(s') - V(s) \tag{4.13}$$

where we can see that running an episode with a policy π would provide us with an unbiased estimate of the $Q(s,a)$. In other words, it is sufficient to know the value of $V(s)$ to compute $A(s,a)$. Therefore, we can also approximate $V(s)$ by a neural network (similar to approximating action-value in DQN [22]).

Moreover, we can combine the two neural networks for estimating $V(s)$ and $\pi(s)$ to learn faster and more effectively. Also, on the negative side, separate networks are likely to learn very similar low-level features. Besides, combining the networks would act as a regularizing element that results in better stability. In the case of the two networks, our neural network share all hidden layers and outputs two set of results - $\pi(s)$ and $V(s)$. The part that optimizes the policy is the ***actor*** and the one that estimates the value function is the ***critic***. *In fact, the* ***critic*** *provides* ***actor*** *with insights about its action.*

The asynchronous notion in A3C originates from the fact that the running of an agent would result in gathering samples with high correlation. To avoid such an issue in DQN, a method called *experience replay* is used by storing the samples in memory and form a batch by retrieving them in random [8]. However, the way A3C handles this issue is to run multiple agents simultaneously. In this case, each agent has its copy of the environment and would use its samples as they arrive. The advantage of this approach is two folds: First, it avoids the correlation as agents would have their unique experience (i.e., various states and transitions). Second, this method requires less memory compared to the experience replay method.

Limitations of A3C: Given all the properties of the vanilla A3C, we examined its performance (quantified using the *criticality-weighted deadline miss ratio*) in the presence of temporal dependencies and correlation of sensor

data values as an example of complex environments. Specifically, we experimented with 8 sensors in the setup explained in detail later (Section 4.4.2), where there is temporal dependence concerning the arrival of data as well as the correlation among data values for two or more sensors as described later in Scenario 4 at Section 4.4.1.

Figure 4.2 shows the *criticality-weighted deadline miss ratio* of policies concerning the *workload intensity* as defined further in equation 4.16. Our numerical evaluation reveals that the performance of A3C is not better than some of the naïve greedy policies (Figure 4.2). With this observation, we hypothesized that such turbulence in the A3C performance relates to the lack of any memorization mechanism to capture recurrent patterns in data streams. Therefore, we decided to examine our hypothesis by proposing a modified version of A3C.

4.3.3 Proposed Approach

We propose an architecture for determining policy at the central hub for deciding on data streams in complex environments. Aligned with the examples provided in Section 4.1.1, we consider studying the following cases, where:

- There exists a correlation among the arrival of data for two or more sensors in the system.
- There is temporal dependence for the arrival of data for two or more

sensors in the system.

- There is temporal dependence concerning the arrival of data as well as the correlation among data values for two or more sensors.

We chose the above cases as a representative set of scenarios for a complex environment inspired by real-world examples (e.g., in [33, 34]). We acknowledge that such a set of cases can get extended to represent more complex environments. We provide more details about each case when we discuss the evaluation more completely (Section 4.4).

We make no assumptions concerning the similarity of information when there is any data arrival correlation or temporal dependency. It means that in such scenarios, if we collect data from two correlated/dependent sensors at two consecutive times (e.g., at t and $t+1$), it does not mean that we have collected "redundant" data. We are solely focusing on collecting data packets in such a way that we have the highest accumulated criticality values over a time horizon. However, in some environment settings, it could be the case that such data arrival correlations/dependence is lead to redundant data. As mentioned above, the proposed model does make this assumption and consequently does not consider having redundant data/messages.[1]

As shown in Figure 4.3, we can abstract the network in two parts: input

[1]As an extension to this work, we can model messages that have related information using a metric such as mutual information, and incorporating this metric into our objective will allow us to optimize the system behaviour suitably.

and learning model.

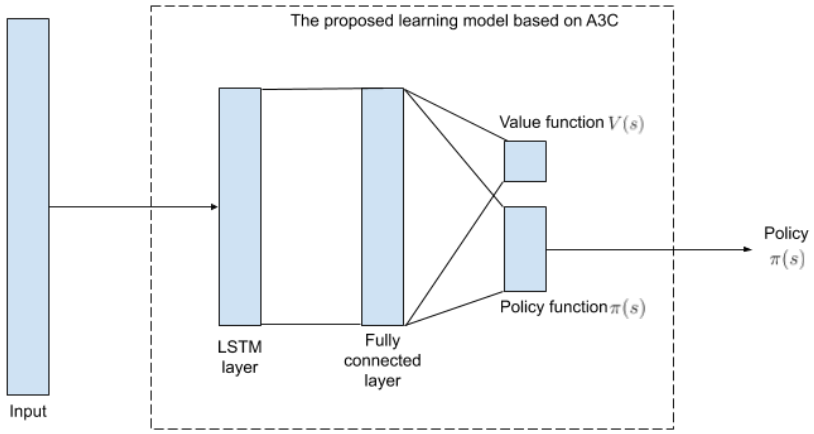


Figure 4.3: The proposed A3C-based network with embedded memory (i.e., LSTM layer).

At the input layer, the system state is an array of $n = n(s)$ tuples ($n(s) = |S|$, S is the set of sensors in the system), each tuple representing the available data-packet (criticality, deadline).

Concerning the learning model, we use the Asynchronous Advantage Actor-Critic (A3C) algorithm [39], which leverages a deep neural network to learn the policy and value functions while running parallel threads to update the parameters of the network. Regularization with policy entropy improves exploration by limiting the premature convergence to sub-optimal policy [39]. The core of our network contains an LSTM layer (output space= $n(S)$) followed by a fully connected network (output space=$2 \times n(S)$) to perform the both required estimations. The LSTM layer is intro-

duced to arm the agent such that it can have some memory of previous states. We propose such embedding of memory to the model as it is likely for an agent in complex environments to encounter recurring patterns in states. In these cases, the model is capable of making efficient decisions only if it can distinguish such patterns. Therefore, we chose to embed memory (i.e., by adding the LSTM layer) to enrich the model with this feature. Alternatively, one could argue about embedding memory to the sensors. However, we believe that such a decision would lead to a more fragile architecture and would negatively impact the scalability of IoT deployments.

Similar to any Reinforcement learning technique, we are concerned with how to take actions in the environment to maximize some notion of cumulative reward. It is essential to define a concrete and reasonable optimization goal, i.e., the reward. We define the reward in such a way that if a message is chosen after the expiry, a penalty is incurred:

$$r(t) = \kappa^{\alpha} - I(\frac{\iota}{t-d})^{\beta} \tag{4.14}$$

where α and β are parameters that help a system architect achieve a balance between criticality and timeliness. These may be adapted on a per-sensor basis, but in this discussion, all sensors use the same settings for these two parameters. t denotes the current time step, and d is the corre-

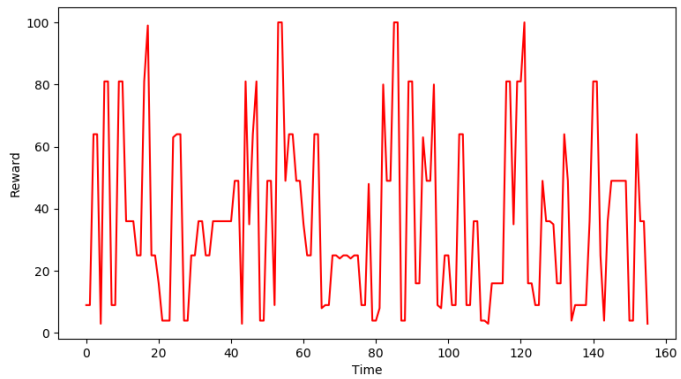


Figure 4.4: An example graph of the reward function for some parameter choices ($\iota = 1$, $\alpha = 2$, $\beta = 4$, and κ and d are randomly selected from interval of $(1,5)$. The peaks correspond to the cases where no penalty is incurred (i.e., $I = 0$), whereas the troughs correspond to the cases with a penalty (i.e., $I = 1$).

sponding deadline of a message. ι is the parameter that characterizes the penalty for a message, and it may be adapted per-message or per-class of messages. For simplicity of exposition, we use the same value of ι for all messages. An example graph of the reward function for some choices of the parameters mentioned above is shown in Figure 4.4.

In terms of the loss function, we define it as follows:

$$L = L_\pi + c_v L_v + c_{reg} L_{reg} \tag{4.15}$$

where L_π is the loss of the policy, L_v is the value error and L_{reg} is a regularization term. The constants c_v and c_{reg} can show which part we want to emphasize on.

4.3.4 Environment

We created an environment for a complex IoT environment where data-packet streams arriving at sensors may have not only different distributions but also temporal dependency and correlation concerning their arrival times and data-packet values. Such properties are known in IoT environments such as intelligent buildings [14], marine environments [33], and multiple mobile sensing and computing applications [71]. In our environment, as mentioned in Section 4.2, data packets arrive at sensors and the central unit chooses a sensor to fetch its data at each polling turn. Then, it verifies whether the deadline of the chosen sensor is still valid. If so, it calculates a reward without any extra penalty; otherwise, it still generates a reward (like a soft scheduler [48]) but subtracts a penalty to reflect the deadline expiry for the chosen data packet. This iteration continues at each time step until the episode finishes.

4.4 Quantitative Evaluation

Having described the proposed approach based on deep reinforcement learning, we discuss other heuristics that we can use to compare with our proposal. To recap the performance goal, we want to reduce the criticality-weighted deadline miss ratio.

We evaluate the policy from the proposed approach along with three other policies, namely: the vanilla A3C, critical greedy, and deadline greedy.

Given the explanations for the A3C (Section 4.3.2) and the proposed approach (Section 4.3.3), we briefly elaborate on the greedy policies:

- **Critical greedy:** In this policy, the central unit selects a sensor using the highest criticality value among the messages at each sensor.

- **Deadline greedy:** In this policy, the central unit selects a sensor using the earliest deadline among the messages at each sensor.

4.4.1 Evaluation Scenarios

As mentioned above, we have four different approaches (i.e. policies) to select a sensor each time. We compare these policies concerning their criticality-weighted deadline miss ratio. To understand the performance differences, we consider four different scenarios, each representing an environment concerning data arrival and values. We chose *Scenario 1* to study our proposed solution in less complex environments (where there is no dependency among sensors). Inspired by the scenarios reported in the literature ([14, 33]), we considered *Scenarios 2, 3, and 4* to explore the performance of our approach in complex environments to overcome the A3C limitation as discussed earlier and shown in Figure 4.2:

- **Scenario 1:** The arrival of data to sensors follows different distributions: a uniform distribution and multiple Poisson distributions - without dependencies.

- **Scenario 2:** The arrival of data to half of the sensors is dependent on each other in a pairwise manner. In other words, sensors in each pair are dependent on each other. In the case of having eight sensors in the system, two pairs (i.e., half of the sensors) correlated the arrival of data. For example, the arrivals of data for the (s_1, s_2) pair of sensors were correlated. The same correlation existed between the arrival of data for the (s_3, s_4) pair. We did not consider any such correlation for the rest of the sensors.

- **Scenario 3:** The arrival of data to half of the sensors is temporally dependent on each other in a pairwise manner. Similar to the details provided in Scenario 2, for half of the sensors, pairs of sensors had temporal dependence on each other. For example, when we had eight sensors, there was temporal dependence for the (s_1, s_2) pair of sensors, and similarly for the (s_3, s_4) pair while there was no dependence between other sensors.

- **Scenario 4:** The arrival of data to half of the sensors is temporally dependent on each other in a pairwise manner. Also, the state value (i.e., data value) of those sensors is dependent on each other. In the case that of eight sensors, for the pair of (s_1, s_2), the data arrivals were temporally dependent, and data values were correlated. The same setting holds for another pair of sensors (e.g., (s_3, s_4)), and the rest of the sensors did not have a dependency of any kind.

4.4.2 Experimental setup

We perform our experiments in all of the above scenarios, where we tried multiple numbers of sensors ($n = 4, 8$, and 16). We chose to report the results for $n = 8$ here due to the similarity of results and the results for the other two cases of 4 and 16 sensors are available in the Appendices (Section B.1 and Section B.2), respectively. To be consistent, we fixed the values of temporal dependencies and correlations (regarding arrival time or state values) wherever they existed across the scenarios. For temporal dependency, we used $t_w = 5$ to denote the temporal difference with correlation value of $corr = 0.9$ (i.e., to hold high correlation). This correlation value was also used for the cases that only correlation exists (no temporal dependency). As mentioned in Section 4.4.1, half of the sensors (i.e., equivalent to four sensors as $n = 8$) have dependencies in scenarios $2, 3,$ and 4, which means two pairs of sensors have either or both of temporal dependency or correlation. We used the arrival rate of $\lambda = 0.15$ for these scenarios. For the first scenario, we chose the arrival time for one sensor from a uniform distribution over the time horizon of the study, and the other sensors from multiple Poisson processes ($\lambda = [0.05, 0.15, 0.3, 0.35, 0.5, 0.65, 0.85]$). Also, for each message, the deadline and criticality values are chosen from a uniform distribution. Therefore, we chose deadlines of messages from the uniform distribution $U_d(1, 10)$, and criticality values from the uniform distribution $U_\kappa(1, 10)$. In the case of dependencies among data values (i.e., criticality and deadline values), the chosen values are still within the ranges

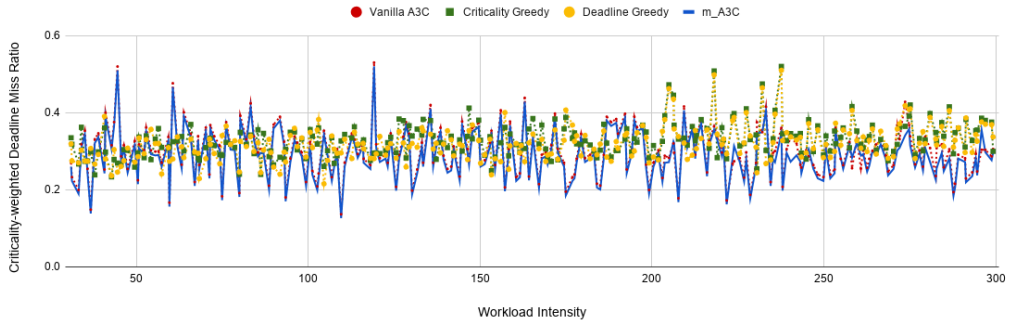


Figure 4.5: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C.

of the above uniform distributions.

We ran the simulations with eight threads as it directly impacts the performance due to affecting the quality of gradient. We tweaked the hyperparameters and ended up having simulation runs through 250 episodes each 5000 steps, the minimum batch size of $n_{batch} = 32$, and the learning rate of $\delta = 0.005$. Also, we chose the values for the constants of the loss function as $c_v = 0.5$ and $c_{reg} = 0.01$.

4.4.3 Results

We elaborate on our observations from the simulations. In each scenario, we considered the aggregate results of the 250 episodes for the evaluations.

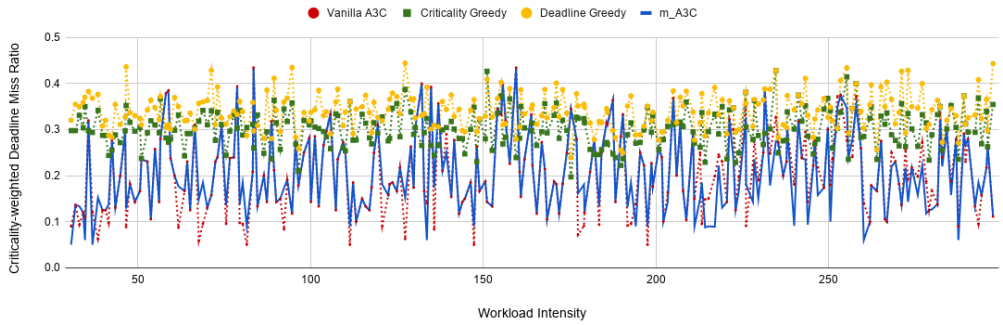


Figure 4.6: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C.

Workload intensity: We define a workload intensity metric as:

$$\sum_{i \in M} \frac{Cr_i \times \lambda_{arrival}}{d_i} \tag{4.16}$$

where M is the set of all messages arriving in a simulation episode, $\lambda_{arrival}$ is the arrival rate of events, and Cr_i and d_i are the criticality and deadline for each message, respectively. We could calculate this metric for each episode of simulations over 5000 steps. Such a metric essentially captures, to some extent, the workload intensity. Therefore, we report the performance of the policies concerning the "*criticality-weighted deadline miss ratio*" (on the Y-axis) along with the "*workload intensity*" (on the X-axis) when presenting the simulation results.

As depicted in Figures 4.5, 4.6, 4.7, and 4.8, each data point represents the overall "*criticality-weighted deadline miss ratio*" in terms of the workload-intensity metric (as defined earlier) over 5000 steps during an episode. Table 4.3 summarizes the results of experiments for the case of 8 sensors. *Across all scenarios, we find the proposed approach performs well. In two of the scenarios (III and IV), it outperforms all others, which confirms the idea of adding memory to capture temporal relations. In scenario I, it still performs slightly better than the vanilla A3C and outperforms the other two policies. In scenario II, it performs competitively when compared with vanilla A3C while still outperforming the two greedy baselines.* Table B.1 (Section B.3) provides a more comprehensive set of results by including the experiments for the cases of having 4 and 16 sensors.

Observing that the modification to the A3C did not negatively affect the performance in less complex scenarios suggests the advantage of the proposed solution over the vanilla A3C as a generalized solution.

In addition to generalizability, our proposed solution also offers scalability compared to the traditional RL approaches, such as the tabular Q-learning. As described earlier (Section 4.3.1), with the proposed DRL solution, we do not deal with updating the Q-table of states and actions. Instead, we have the states as the input of the NN and approximate the value function and action values. Therefore, we alleviate the massive consumption of computing resources for updating a Q-table. In this way, it is easier to increase

Figure 4.7: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) outperforms other policies.

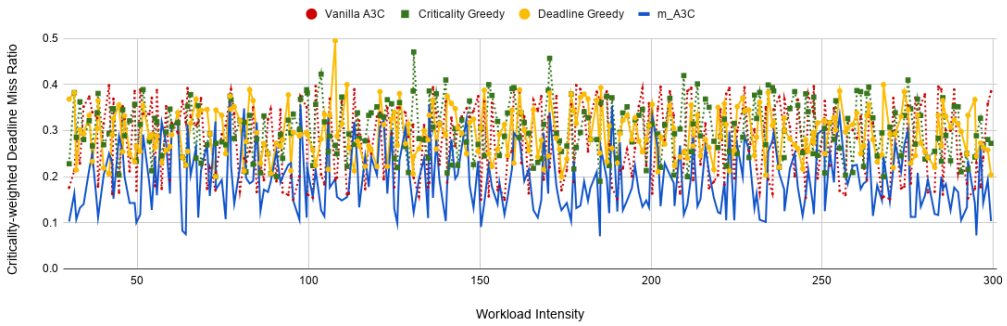


Figure 4.8: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 8 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table 4.3. The proposed approach (m_A3C) outperforms other policies.

the number of sensors as it only changes the size of the input array in the NN, which requires considerably lower computing resources than Q-tables that grow exponentially. One possible bottleneck towards the large scale implementation of our approach would be the training time that could in-

crease in environments with more sensors and complexity. This issue, however, could be addressed by leveraging GPUs for computational speedup, as shown to be effective in improving A3C performance by Babaeizadeh et al. [9].

4.5 Related Work

Our research brings multiple areas such as DRL, performance analysis, and job scheduling together. Therefore, the related work section is mostly about how DRL has helped to solve problems of scheduling specifically about the performance of sophisticated sensors deployment that capturing data patterns is non-trivial [60, 73].

In the scheduling of distributed tasks in grid and cloud, several optimization works are available around metrics such as lifespan and load balance [35, 55]. However, such works reflect on either of criticality/importance or timeliness, which leaves studying the trade-off of the two parameters open - as we do.

In terms of performance analysis, our work shares similarities such as the notion of the metrics between what we study as the criticality-weighted deadline miss ratio (Section 4.1) and other metrics such as the one explained by Salah er al. [52]. Aside from the performance metrics, different works consider hierarchical models, such as the ones studied by Khazaei et al. [29] and Rashtian et al. [48]. Our work, though, is focusing on

the case of a centralized model as a standard model in infrastructure-as-a-service (IaaS) platforms.

On the RL side, our work compares to the body of work done via policy gradient methods. Policy gradient methods include a large family of reinforcement learning algorithms. They have a long history [70], but only recently were backed by neural networks and had success in high-dimensional cases. A3C algorithm was published in 2016 and can do better than DQN [43] with a fraction of time and resources [39]. Although there are several papers leveraging policy gradient methods, our work is distinguished from many of them by both the application and network structure.

4.6 Conclusions and future work

We have demonstrated that a deep reinforcement learning approach to balancing message criticality and timeliness in a complex IoT environment is effective. We proposed an approach by improving upon the A3C algorithm via embedding memory to the model such that it can capture recurring patterns of data in complex environments. Our solution outperforms the studied alternative heuristics that one could use in this context. We envision the applicability of such an idea in the context of IoT environments that host temporally dependent and correlated data arrival. We have tackled this problem in the specific settings where messages have two essential properties of criticality and timeliness.

We have shown that our solution is effective in four scenarios. The proposed approach outperforms the rest of the policies in complex scenarios, where we have data arrival correlations, data arrival temporal dependency, and data arrival temporal dependency plus correlation of data (concerning criticality and deadline values). Also, our solution remains effective in the more straightforward, where the arrival of data to sensors follows different distribution without dependencies. The observation of the results both in complex and non-complex scenarios suggest the generalizability of the proposed solution.

Concerning model scalability, the proposed model's training time may vary depending on the input size (i.e. the number of sensors deployed in the environment). However, we found that decreasing the training time would result in a slightly less optimal policy. In order to choose between model-based and model-free approaches for deriving a policy (e.g., DRL techniques), one may consider that DRL methods require a large number of samples from the environment to perform well. While such methods provide more generalized solutions (with minimal assumptions about the environment), in case of severe limitation for training time, model-based approaches may still be reasonable if a comprehensive understanding of the environment is available.

As future work, we envision multiple avenues to explore. One natural extension to this work is to explore the effectiveness of the approach with

more scenarios. This path may lead to the modification of network architecture. Another possible plan could be to compare the proposed approach with solutions based on some other DRL algorithms, such as PPO [53]. Such explorations could shed light on the trade-off of convergence vs. stability across DRL-based approaches. Lastly, another direction for future work would be to answer the question of "How should we model information correlation to preventing redundant message collection?". In this case, we may be able to use mutual information as a metric to capture the value of different messages; we could ignore messages with high mutual information correlation in relation to messages that have been scheduled. As our proposed model here does not assume and address collecting redundant data, it would be interesting to explore potential modification to the framework, to prevent collecting repetitive data while still capturing recurring patterns within data. We envision that addressing this problem in our proposed framework would require updates to the reward function and performance metric, but the overall solution structure is likely to remain the same. Finally, it would be interesting to solve a similar case when the assumption is to have two or more classes of tasks (e.g., critical and regular/normal tasks), and the goal is to prioritize critical tasks over other classes of tasks. Such a study would complement our work in this chapter as the current problem settings do not assume such a clear distinction among tasks and solely focuses on minimizing the *weighted-criticality deadline miss ratio* metric over a finite horizon.

Average Criticality-weighted Deadline Miss Ratio ($\overline{\rho}$)				
Scenario	m_A3C	Vanilla A3C	Criticality Greedy	Deadline Greedy
Scenario 1	0.2885	0.2989	0.3277	0.3201
Scenario 2	0.2073	0.2063	0.3012	0.3398
Scenario 3	0.2077	0.3139	0.3145	0.3056
Scenario 4	0.1938	0.2779	0.3018	0.2942

Table 4.3: The table summarizes the results for the case of 8 sensors. It reports the average Criticality-weighted Deadline Miss Ratio for the policies in each of the scenarios. As shown in green-coloured cells, the proposed policy (m_A3C) consistently performs as the best policy. Even when it is not the best (i.e., Scenario II), it still performs reasonably well with only one percent of difference (concerning ρ) compared to the best result. Also, the red-coloured cells represent the worst performance (by the greedy policies) across all the scenarios.

Chapter 5

Conclusions & Future Work

With the proliferation of data-rich environments such as IoT networks, there is an urgent need for the development of proper mechanisms to manage massive volumes of data. A natural pipeline for the data within an IoT environment is to have data management as the first step, followed by analysis to support large-scale reasoning and decision making in a variety of applications [7]. However, one cannot gather all data for further processing in such a pipeline due to likely limitations concerning resources. In this dissertation, we have concentrated on the first step of the pipeline alone; the amount of data produced by sensors alone can be overwhelming, and we need the right strategies for handling this deluge. Effective filtering of data in the early stages can reduce the pressure on other stages of IoT data processing systems that would store and perform computation on the

data.

Studying statistical models of data-rich environments such as IoT networks brings us closer to understanding what prioritizing the data means in the context of such systems. It makes clearer how to model and interpret such prioritization. We were successful at providing reasonable policies in cases that we studied. We anticipate–and hope– that our proposed techniques have implications on how data prioritization should be developed for various systems.

Providing high quality of service is a challenging endeavour, especially in light of resource constraints and even more when environments impose increased complexity on patterns in data streams. The models that we proposed in the three sections use performance metrics that enable one to reason about and improve the system functionality concerning the quality of service in the presence of the challenges above.

Numerous modern computing systems integrate intelligent computational entities, such as machine learning-based approaches, to improve the analysis of data that were somehow collected. However, we applied reinforcement learning techniques to develop methods such that with no - and sometimes - very little knowledge available about their timeliness and also importance (i.e., criticality), one can control system performance. Moreover, in Chapter 4, we tried to improve the work in earlier chapters (Chapters 2

and 3) by making them more generic to avoid high adaptivity concerning system settings as we believe this will lead to a more fragile system.

Before discussing potential future directions, we wish to emphasize that the ultimate goal of our efforts is the development of tools that would assist the system designer/architecture in making design decisions relating to data collection aspects of IoT systems. One such tool is an automatic data scheduler (or collector) for systems with minimal knowledge of the arriving data streams depending on the system architecture, either in a centralized or decentralized (e.g., hierarchical) design. In these cases, the data scheduler/collector combines sensor-based information with machine learning capabilities to find patterns in data from sensors. It pulls relevant information together and passes them to other entities in the system for further analysis, understanding, and insights. Considering the whole pipeline in IoT systems, the focus in this dissertation is on the first part that is *data collection*.

5.1 Limitations

There are limitations to this work that we would like to acknowledge. While we envision the three studied problems as a representative set in terms of system architecture, some of our assumptions would still be limited to the specific cases that we studied and that they could impact on the overall analysis involved in the final solutions.

Although we chose to focus on three properties of data streams and environments (i.e., timeliness, criticality, and environment complexity) as essential factors, there are potentially other factors that could contribute to characterizing IoT systems. Examples are environmental factors that could directly affect the sensor quality of data or potential biases in data sampling by sensors. While such factors can be integrated into our proposed solutions, we avoided having too many of them as we believe such adaptivity will lead to more fragile system architectures.

Finally, our solutions were around performance metrics that rely on the criticality value of the data streams/packets. Another way of envisioning criticality/importance is to infer criticality from the data itself adaptively. For instance, imagine that the data scheduler collects data to serve a classification task via *boosting*(i.e., using a series of classifiers rather one classifier to decrease bias). In this case, the more critical data packets are the ones that the previous classifiers have failed (i.e. harder for classifiers to get it right), in the hope that the next classifier will focus on these harder examples - rather than criticality value of the data itself. Although following such a notion of criticality may result in missing some critical data packets, it would be beneficial in the long run as it improves the analytics capabilities (e.g., classification accuracy in this example).

5.2 Potential Future Directions

Besides the future work directions we identified in the individual chapters, we would like to point out a few other relevant research directions. In this dissertation, the focus is on IoT networks as an example of data-rich environments. As an interesting area of exploration, one would explore the adaptation of data prioritization techniques applicable to a broader range of cases. For instance, Bandit algorithms have been useful in systems dealing with unknown stochastic environments and seek to optimize a long-term reward by learning and exploiting the unknown environment simultaneously. However, such techniques may not be applicable in the systems, where there exist safety guarantees that have to be met at every single round. Therefore, it is crucial to come up with new bandit algorithms that account for critical safety requirements.

Another exciting direction would be to create a taxonomy of characteristics in data-rich environments (not limited to IoT), which could ultimately shed light on developing a unified framework for data prioritization with minimal change of settings. The area of Model-Agnostic machine learning [49] seems to be promising to achieve this goal.

Another likely direction that can improve generic solutions is to figure out how to interpret models. In the majority of cases, understanding of why ML models work is limited intuitive heuristics and does not translate into a

precise interpretation of the process. To overcome this issue, “interpretable machine learning” [40], provides appropriate methods to understand models that are already performing well even to improve them.

Being able to interpret generic models results in pinpointing the potential areas for improvements. In this case, one can explore the effectiveness of learning to learn (i.e., “meta-learning” [65]), where the idea is to build self-adaptive learners that improve their bias dynamically through experience by accumulating meta-knowledge.

Bibliography

[1] Facebook statistics available at http://rob.social5.net/post/how-to-reach-more-customers-without-adding-more-minutes/. → page 2

[2] IBM report available at https://www-01.ibm.com/software/in/data/bigdata/. → page 1

[3] https://www.wired.com/insights/2014/11/the-internet-of-things-bigger/. → page 1

[4] Pokemon Go Engagement Trend: http://www.bloomberg.com/news/articles/2016-08-22/these-charts-show-that-pokemon-go-is-already-in-decline. → page 3

[5] Twitter statistics available at http://www.internetlivestats.com/twitter-statistics/. → page 2

[6] Youtube statistics available at http://expandedramblings.com/index.php/youtube-statistics/. → page 2

[7] C. C. Aggarwal. *Managing and mining sensor data*. Springer Science & Business Media, 2013. → pages 62, 132

[8] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba.

Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017. → page 113

[9] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Ga3c: Gpu-based a3c for deep reinforcement learning. *CoRR abs/1611.06256*, 2016. → page 127

[10] P. Billingsley. *Weak convergence of measures: Applications in probability*. SIAM, 1971. → page 23

[11] A. Burns and R. I. Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys*, 50(6), January 2018. → page 90

[12] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya. Ga-eti: An enhanced genetic algorithm for the scheduling of scientific workflows in cloud environments. *Journal of computational science*, 26:318–331, 2018. → pages 6, 88

[13] H. Chang, J. Feng, and C. Duan. Reinforcement learning-based data forwarding in underwater wireless sensor networks with passive mobility. *Sensors*, 19(2):256, 2019. → page 90

[14] M. Charfi, Y. Gripay, and J.-M. Petit. Spatio-temporal functional dependencies for sensor data streams. In *International Symposium on Spatial and Temporal Databases*, pages 182–199. Springer, 2017. → pages 7, 94, 119, 120

[15] H. Cho and J. Jeong. Implementation and performance analysis of power and cost-reduced opc ua gateway for industrial iot platforms. In *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–3. IEEE, 2018. → page 100

[16] L. L. Clark, C. E. DeArdo, and B. A. Shaffer. Adaptive delayed polling of sensors. US Patent 4,598,363, 1986. → page 62

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. McGraw-Hill Science/Engineering/Math, 2009. ISBN 0262033844. → pages 31, 53

[18] D. G. Costa. Visual sensors hardware platforms: A review. *IEEE Sensors Journal*, 2019. → page 100

[19] D. Dhar, S. Gopalakrishnan, and K. Rostamzadeh. Optimal sensing using query arrival distributions. In *Proceedings of the ACM International Symposium on Design and Analysis of Intelligent Vehicular Networks and applications*, pages 39–46, 2012. → page 61

[20] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018. → page 109

[21] J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed Bandit Allocation Indices*. John Wiley and Sons, 2011. → page 56

[22] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016. → page 112

[23] S. Guha, K. Munagala, and P. Shi. Approximation algorithms for restless bandit problems. *Journal of the ACM*, 58(1), Dec. 2010. → page 59

[24] D. Han, J. Wu, Y. Mo, and L. Xie. On stochastic sensor network scheduling for multiple processes. *arXiv CoRR*, abs/1611.08222, 2016. → page 62

[25] S. Heo, S. Song, J. Kim, and H. Kim. RT-IFTTT: Real-time IoT framework with trigger condition-aware flexible polling intervals. In *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2017. → page 61

[26] F. Iannello and O. Simeone. On the optimal scheduling of independent, symmetric and time-sensitive tasks. *IEEE Transactions on Automatic Control*, 58(9):2421–2425, Sept 2013. ISSN 0018-9286. doi:10.1109/TAC.2013.2258791. → page 60

[27] F. Iannello, O. Simeone, and U. Spagnolini. Optimality of myopic

scheduling and whittle indexability for energy harvesting sensors. In *Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, 2012. → page 59

[28] B. Jana and J. Poray. A hybrid task scheduling approach based on genetic algorithm and particle swarm optimization technique in cloud environment. In *Intelligent Engineering Informatics*, pages 607–614. Springer, 2018. → pages 6, 89

[29] H. Khazaei, J. Misic, and V. B. Misic. A fine-grained performance model of cloud computing centers. *IEEE Transactions on parallel and distributed systems*, 24(11):2138–2147, 2012. → pages 89, 127

[30] J. E. Kim, T. Abdelzaher, A. Bar-Noy, and R. Hobbs. Sporadic decision-centric data scheduling with normally-off sensors. In *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2016. → page 61

[31] R. Kleinberg and N. Immorlica. Recharging bandits. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, Oct. 2018. → pages 5, 58

[32] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000. → page 104

[33] J. Liu, T. Zhang, G. Han, and Y. Gou. Td-lstm: Temporal dependence-based lstm networks for marine temperature prediction. *Sensors*, 18(11):3797, 2018. → pages 7, 94, 115, 119, 120

[34] M. Á. López Medina, M. Espinilla, C. Paggeti, and J. Medina Quero. Activity recognition for iot devices using fuzzy spatio-temporal features as environmental sensor fusion. *Sensors*, 19(16):3512, 2019. → page 115

[35] M. C. Lucas-Estañ and J. Gozalvez. Load balancing for reliable self-organizing industrial iot networks. *IEEE Transactions on Industrial Informatics*, 2019. → page 127

[36] M. C. Lucas-Estan and J. Gozalvez. Load balancing for reliable

self-organizing industrial IoT networks. *IEEE Transactions on Industrial Informatics*, 2019. → pages 6, 88

[37] S. T. Maguluri and R. Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transactions On Networking*, 22(6): 1938–1951, 2013. → pages 6, 88

[38] P. A. Meyer. *Probability and potentials*, volume 357. Blaisdell Waltham, 1966. → page 39

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. → pages 97, 100, 116, 128

[40] C. Molnar. *Interpretable machine learning*. Lulu. com, 2019. → page 137

[41] T. Moon, W. Chu, L. Li, Z. Zhang, and Y. Chang. An online learning framework for refining recency search results with user click feedback. *ACM Transactions on Information Systems*, 30(4), November 2012. → page 15

[42] J. Niño-Mora and S. S. Villar. Sensor scheduling for hunting elusive hiding targets via Whittle's restless bandit index policy. In *Proceedings of the International Conference on Network Games, Control and Optimization*, pages 1–8, 2011. → page 59

[43] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016. → page 128

[44] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of optimal queuing network control. *Mathematics of Operations Research*, 24(2): 293–305, May 1999. → page 59

[45] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Trans. Emerg. Telecommun. Technol.*, 25(1):81–93, Jan. 2014. ISSN

2161-3915. doi:10.1002/ett.2704. URL http://dx.doi.org/10.1002/ett.2704. → page 12

[46] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York, 5 edition, 2005. → page 43

[47] L. Quan, Z. Wang, and F. Ren. A novel two-layered reinforcement learning for task offloading with tradeoff between physical machine utilization rate and delay. *Future Internet*, 10(7):60, 2018. → page 90

[48] H. Rashtian and S. Gopalakrishnan. Balancing message criticality and timeliness in iot networks. *IEEE Access*, 7:145738–145745, 2019. → pages 98, 100, 119, 127

[49] M. T. Ribeiro, S. Singh, and C. Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016. → page 136

[50] S. M. Ross. *Introduction to stochastic dynamic programming*. Academic Press, 2014. → pages 5, 57

[51] H. L. Royden and P. Fitzpatrick. *Real analysis*, volume 198. Macmillan New York, 1988. → page 23

[52] K. Salah, K. Elbadawi, and R. Boutaba. An analytical model for estimating cloud resources of elastic services. *Journal of Network and Systems Management*, 24(2):285–308, 2016. → pages 89, 127

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. → page 130

[54] L. Seno, G. Cena, A. Valenzano, and C. Zunino. Bandwidth management for soft real-time control applications in industrial wireless networks. *IEEE Transactions on Industrial Informatics*, 13 (5):2484–2495, October 2017. → page 88

[55] J. Shi, J. Luo, F. Dong, and J. Zhang. A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud. In *International Conference on Computer Supported Cooperative Work in Design*, pages 672–677. IEEE, 2014. → pages 88, 127

[56] H. Y. Shishido, J. C. Estrella, C. F. M. Toledo, and M. S. Arantes. Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds. *Computers & Electrical Engineering*, 69:378–394, 2018. → pages 6, 88

[57] W. J. Stewart. *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994. → page 20

[58] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. → page 104

[59] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. → page 111

[60] A. A. Szpiro, P. D. Sampson, L. Sheppard, T. Lumley, S. D. Adar, and J. D. Kaufman. Predicting intra-urban variation in air pollution concentrations with complex spatio-temporal dependencies. *Environmetrics*, 21(6):606–631, 2010. → page 127

[61] P. S. Thomas and E. Brunskill. Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. *arXiv preprint arXiv:1706.06643*, 2017. → page 111

[62] H. C. Tijms. *Stochastic models: an algorithmic approach*, volume 303. John Wiley & Sons Inc, 1994. → page 41

[63] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of

virtual machines across multiple providers. *Future generation computer systems*, 28(2):358–367, 2012. → pages 6, 88

[64] D. C. Vanderster, N. J. Dimopoulos, R. Parra-Hernandez, and R. J. Sobie. Resource allocation on computational grids using a utility model and the knapsack problem. *Future Generation Computer Systems*, 25(1):35–50, 2009. → page 88

[65] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002. → page 137

[66] S. S. Villar. Indexability and optimal index policies for a class of reinitialising restless bandits. *Probability in the Engineering and Informational Sciences*, 30(1):1–23, Jan. 2016. → page 61

[67] A. A. Visheratin, M. Melnik, and D. Nasonov. Workflow scheduling algorithms for hard-deadline constrained cloud environments. *Procedia Computer Science*, 80:2098–2106, 2016. → page 100

[68] S. Weerakkody, Y. Mo, B. Sinopoli, D. Han, and L. Shi. Multi-sensor scheduling for state estimation with event-based, stochastic triggers. *IEEE Transactions on Automatic Control*, 61(9):2695–2701, September 2016. → page 62

[69] P. Whittle. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability*, 25:287–298, 1988. ISSN 00219002. URL http://www.jstor.org/stable/3214163. → pages 14, 18, 55

[70] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4): 229–256, 1992. → pages 104, 128

[71] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, pages 351–360. International World Wide Web Conferences Steering Committee, 2017. → pages 7, 119

[72] S. Zhan and H. Huo. Improved PSO-based task scheduling algorithm in cloud computing. *Journal of Information & Computational Science*, 9(13):3821–3829, 2012. → pages 6, 89

[73] J. Zhang, Y. Zheng, and D. Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. → page 127

Appendix A

Supplementary material for Chapter 2

The following table summarizes the symbols used in Chapter 2.

Symbol	Description
S_i	i_{th} sensor
v_i	The value associated with data from S_i
β	Rate of data value decay
P	Polling period
μ	Rate at which new events occur in the environment
q	Probability of sensing a new event
M	Number of exponential distributions in a hyper-exponential process
$\Upsilon_i(t)$	Utility accrued at sensor S_i at time t
$a_i(t)$	Action taken at time t concerning sensor S_i
γ_i	Bandwidth required by sensor S_i
B	Given bandwidth limit for all sensors
ρ	Discount factor for reward calculation
$V_\rho(S_i)$	Value function associated with choosing sensor S_i
ξ	Average reward for the chosen control policy

Appendix B

Supplementary results of Chapter 4

Besides the experiment results reported for the case of 8 sensors in Chapter 4, we provide the extended experiments results for the cases of 4 and 16 sensors as well as the complete table of results.

B.1 Experiments results for 4 sensors

In this section, we present the experiment results for the case of having 4 sensors:

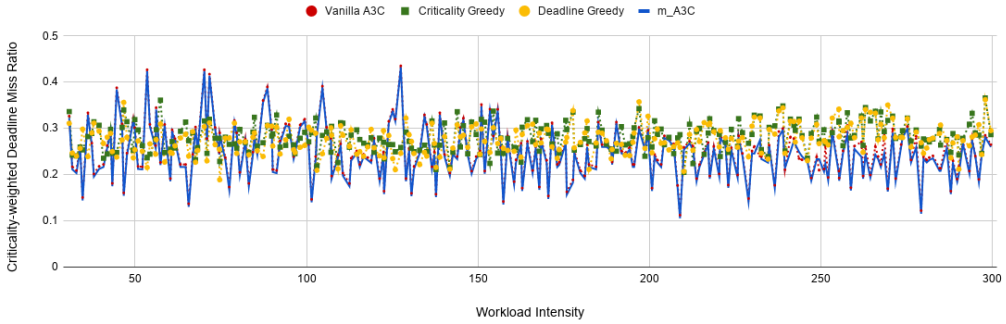


Figure B.1: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

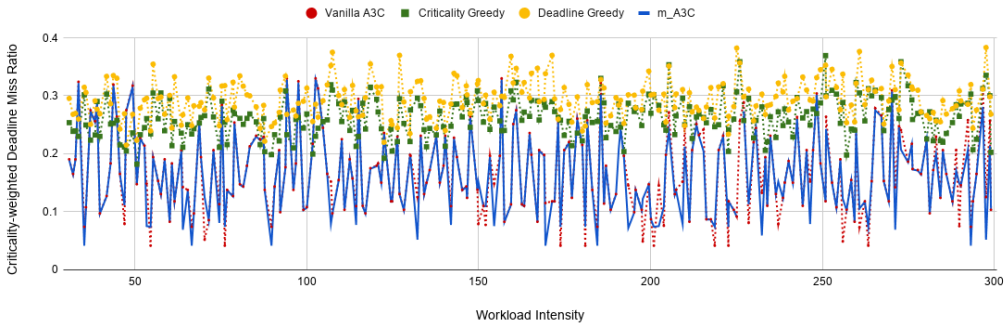


Figure B.2: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

Figure B.3: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

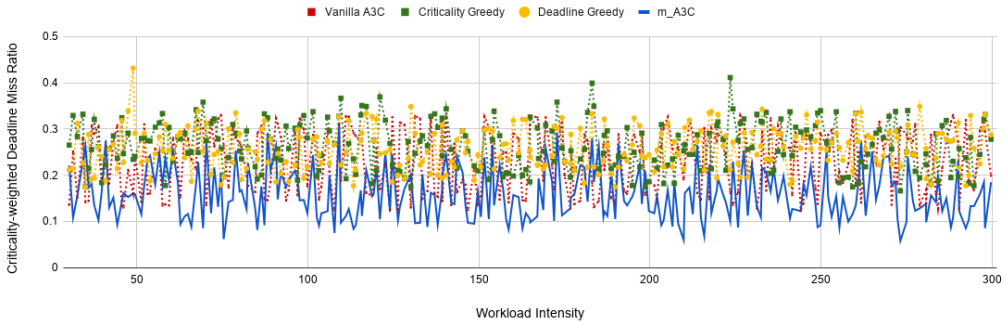


Figure B.4: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 4 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

B.2 Experiments results of 16 sensors

In this section, we present the experiment results for the case of having 16 sensors:

Figure B.5: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 1 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C.

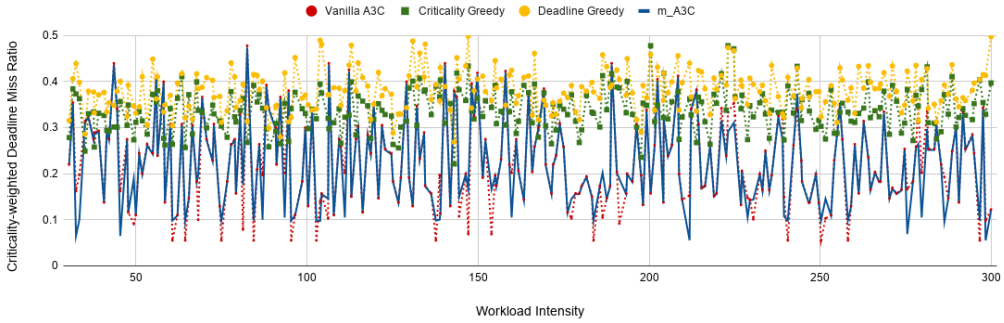


Figure B.6: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 2 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

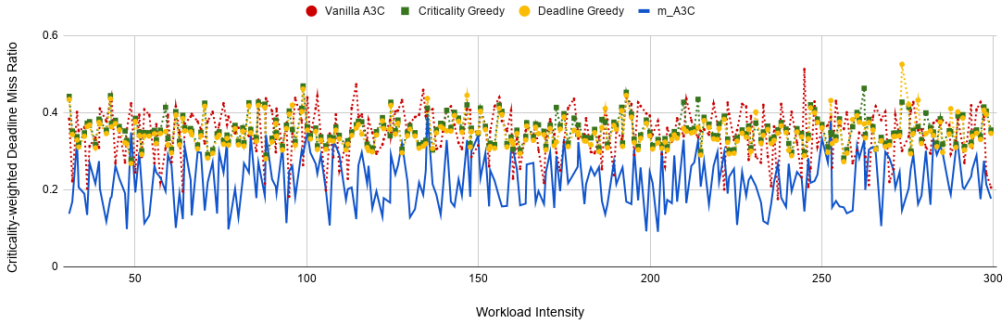


Figure B.7: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 3 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

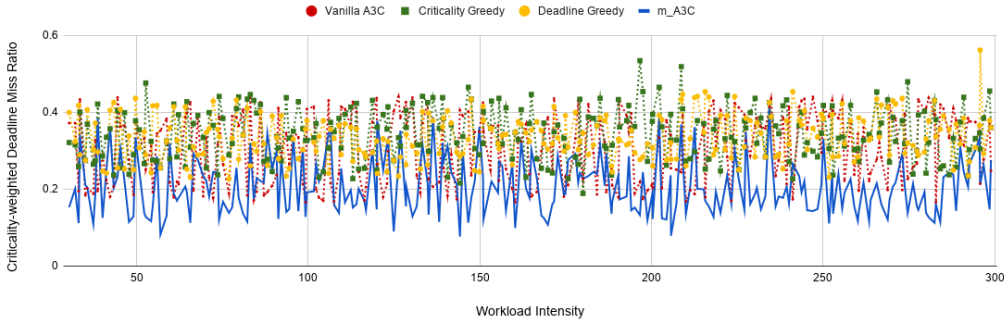


Figure B.8: Range of criticality-weighted deadline miss ratio (ρ) values concerning the workload intensity for scenario 4 with 16 sensors. The plot indicates the range of *criticality-weighted deadline miss ratio* values using the same data reported in Table B.1. The proposed approach (m_A3C) outperforms the vanilla A3C.

B.3 Table of results summary

The following table summarizes the results for the cases of $4, 8$, and 16 sensors. It reports the average Criticality-weighted Deadline Miss Ratio for the policies in each of the scenarios. As shown in green-coloured cells, the proposed policy (m_A3C) almost consistently performs as the best policy. Even when it is not the best, it still performs reasonably well with only one percent of difference (concerning ρ) compared to the best result. The red-coloured cells show the worst performance among the policies, which is dominantly shared between the two greedy policies.

Average Criticality-weighted Deadline Miss Ratio ($\overline{\rho}$)					
#Sensors	Scenario	m_A3C	Vanilla A3C	Criticality Greedy	Deadline Greedy
4	Scenario 1	0.2436	0.2493	0.2806	0.2744
	Scenario 2	0.1643	0.1683	0.2616	0.2934
	Scenario 3	0.1710	0.2586	0.2749	0.2674
	Scenario 4	0.1585	0.2306	0.2636	0.2572
8	Scenario 1	0.2885	0.2989	0.3277	0.3201
	Scenario 2	0.2073	0.2063	0.3012	0.3398
	Scenario 3	0.2077	0.3139	0.3145	0.3056
	Scenario 4	0.1938	0.2779	0.3018	0.2942
16	Scenario 1	0.3234	0.3335	0.3550	0.3494
	Scenario 2	0.2222	0.2249	0.3367	0.3805
	Scenario 3	0.2254	0.3482	0.3542	0.3458
	Scenario 4	0.2084	0.3076	0.3431	0.3350

Table B.1: The summary of all experiments concerning the Criticality-weighted Deadline Miss Ratio.