University of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

# Low-Cost Approximate and Adaptive Monitoring Techniques

Demetris Trihinas

A Dissertation Submitted to the University of Cyprus

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

December, 2017

# VALIDATION PAGE

**Doctoral Candidate:** Demetris Trihinas

**Doctoral Dissertation Title:** Low-Cost Approximate and Adaptive Monitoring Techniques

*The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Department of Computer Science and was approved on December 18, 2017 by the members of the Examination Committee.*

**Examination Committee:**

Committee Chair _____
Andreas Pitsillides

Research Supervisor _____
Marios D. Dikaiakos

Research Supervisor _____
George Pallis

Committee Member _____
Rizos Sakellariou

Committee Member _____
Sarunas Girdzijauskas

Committee Member _____
Elias Athanasopoulos

# DECLARATION OF DOCTORAL CANDIDATE

*The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.*

..... [Full Name of Doctoral Candidate]

..... [Signature of Doctoral Candidate]

# Περίληψη

Με τη ευρεία διάδοση έξυπνων συσκευών και τη καθολική αναγνώριση που έχουν αποκτήσει στη συνείδηση των καταναλωτών, η υπολογιστική νοημοσύνη όπως τη γνωρίζουμε, εξελίσσεται με συνέπεια να επηρεάζει την καθημερινότητά μας και να διαμορφώνει αυτό που γνωρίζουμε, ως το Διαδίκτυο των Πραγμάτων. Για τον λόγο αυτό, βλέπουμε ότι ο υπολογισμός έχει αρχίσει να μεταφέρεται στις πηγές παραγωγής δεδομένων, που βρίσκονται κατανεμημένες στη άκρη του διαδικτύου, με σκοπό τη δημιουργία αναλυτικών γνώσεων σε πραγματικό χρόνο για όλους σχεδόν τους τομείς της βιομηχανίας. Ωστόσο, για την παραγωγή ενός τέτοιου πλούτου γνώσεων, απαιτείται έντονη επεξεργασία και διαρκής μεταφορά δεδομένων μέσω διαδικτύου. Αυτό έχει ως αποτέλεσμα την αυξημένη κατανάλωση ενέργειας για τις ίδιες τις πηγές, ενώ οι υπηρεσίες επεξεργασίας ροών δεδομένων, συνεχώς επιφορτίζονται και αγωνίζονται να είναι αποτελεσματικές. Παρά τις προσπάθειες αύξησης της ενσωμάτωσης έξυπνων συσκευών με τις δυνατότητες επεξεργασίας που παρέχονται από υπολογιστικές νεφέλες, εξακολουθούν να υπάρχουν ανασταλτικοί παράγοντες όπως περιορισμοί στο εύρους ζώνης και χρονικές καθυστερήσεις δικτύου λόγω απόστασης.

Σε αυτή τη διατριβή, αντιμετωπίζουμε την επεξεργασία ροών δεδομένων σε πραγματικό χρόνο και την καλύτερη ενεργειακή απόδοση σε κατανεμημένες πηγές δεδομένων, αναπτύσσοντας προσεγγιστικές και προσαρμοστικές τεχνικές παρακολούθησης χαμηλού κόστους ως μέσο αντιμετώπισης αυτών των προκλήσεων. Εάν μπορεί να γίνει ανεκτός ένας βαθμός ανακρίβειας στον υπολογισμό, τότε προσεγγιστικές τεχνικές παρακολούθησης, όπως η προσαρμοστική δειγματοληψία (sampling), το φιλτράρισμα (filtering) και η βασιζόμενη σε μοντέλα διάδοση δεδομένων (model-based dissemination), μπορούν να μειώσουν σημαντικά την κατανάλωση ενέργειας και τη ποσότητα των δεδομένων που

κατακλύζουν υπηρεσίες επεξεργασίας ροής δεδομένων. Αυτό επιτυγχάνεται προσαρμόζοντας δυναμικά, στη ίδια τη πηγή και με χαμηλό κόστος, το ρυθμό συλλογής και διάδοσης μετρικών με βάση την τρέχουσα διακύμανση και την κατανομή της ροής δεδομένων. Για να επιτευχθεί αυτό, εισάγουμε ένα προσεγγιστικό αλγοριθμικό μοντέλο μάθησης χαμηλού κόστους ικανό να παρακολουθεί την εξέλιξη και τη μεταβλητότητα της ροής δεδομένων σε πραγματικό χρόνο, επιτρέποντας έτσι στις τεχνικές παρακολούθησης, που αναπτύξαμε, να προσαρμόζουν το ρυθμό συλλογής και διάδοσης μετρικών με βάση τη ικανότητα του αλγοριθμικού μοντέλου να εκτιμήσει σωστά τι θα συμβεί στη συνέχεια στη ροή δεδομένων. Ιδιαίτερη προσοχή λαμβάνεται για τη διαρκή προσαρμογή του αλγοριθμικού μοντέλου, εισάγοντας στη εκτίμηση μεταβολής της ροή δεδομένων προσαρμοστική στάθμιση παραμέτρων (adaptive parameter weighting), ανίχνευση τάσεων (trend detection) και εμπλουτισμό συμπεριφοράς εποχικότητας (seasonality knowledge enrichment), έτσι ώστε να αναγνωρίζονται άμεσα απότομες αλλαγές στην διακύμανση της ροής και να ξεπερνιούνται τυχόν καθυστερήσεις στην διαδικασία εκτίμησης.

Στη συνέχεια, παρουσιάζουμε το AdaM, που είναι μια αυτόματα προσαρμοζόμενη βιβλιοθήκη παρακολούθησης που σχεδιάστηκε για να ενσωματώνεται στον πυρήνα λογισμικού πηγών δεδομένων (π.χ. συσκευές IoT) για να παρέχει προσαρμοστική παρακολούθηση ενσωματώνοντας τις χαμηλού κόστους προσεγγιστικές τεχνικές παρακολούθησης για προσαρμοστική δειγματοληψία και φιλτράρισμα, που αναπτύχθηκε στο πλαίσιο της διατριβής. Στη συνέχεια, παρουσιάζουμε το ADMin, ένα πρόσθετο (plugin) που αναπτύχθηκε για να διευρύνει τη λειτουργικότητα του AdaM, προσφέροντας προσαρμοστική διάδοση δεδομένων βασισμένη σε μοντέλα, για να προσαρμόσει αποτελεσματικά, στη ίδια τη πηγή, τον ρυθμό με τον οποίο οι πηγές παρακολούθησης διανέμουν μετρήσεις μέσω διαδικτύου. Κλείνουμε, με αξιολόγηση της αποτελεσματικότητας και ακρίβειας της βιβλιοθήκης μας, διεξάγοντας εκτενής σειρά πειραμάτων με πραγματικά δεδομένα από εφαρμογές υπολογι-

στικών νεφελών, φορητές συσκευές σώματος, φωτοβολταϊκά και έξυπνες υπηρεσίες μέσων μαζικής μετακίνησης. Τα αποτελέσματα δείχνουν ότι το AdaM είναι σε θέση να προσαρμόσει δυναμικά τον ρυθμό συλλογής και διάδοσης μετρικών, ενώ επιτυγχάνει ισορροπία μεταξύ απόδοσης και ακρίβειας. Ιδιαίτερα, για τις αξιολογούμενες εφαρμογές, το AdaM μπορεί να μειώσει την κατανάλωση ενέργειας κατά τουλάχιστον 83%, τον όγκο δεδομένων κατά 71%, τις καθυστερήσεις ανίχνευσης μετατόπισης στη μοντελοποίηση κατά 61%, διατηρώντας πάντοτε την ακρίβεια πάνω από 89% σε σύγκριση με άλλες προσαρμοστικές βιβλιοθήκες.

# Abstract

As consumers embrace the widespread of ubiquitously connected "smart" devices, computing as we know it evolves in surprising ways impacting our everyday life and forming what is known as the Internet of Things (IoT). With the prevalence of IoT we are seeing intelligence aggressively deployed at the edge to produce real-time analytic insights for almost all industry sectors. However, to produce such an unprecedented wealth of insights intense processing and constant data dissemination over the network is still required. This results in increased energy consumption for monitoring sources while cloud and streaming services consuming IoT data are constantly overwhelmed and struggling to be effective. Despite attempts of augmenting IoT devices with the power of the cloud, there still exist numerous inhibitors masked under constant data movement such as bandwidth limitations and network latencies.

In this thesis, we tackle real-time data processing and energy-efficiency on the edge of monitoring networks by developing low-cost approximate and adaptive monitoring techniques as the remedy to these challenges. If a degree of inaccuracy can be tolerated, approximate monitoring techniques such as adaptive sampling, filtering and model-based dissemination, can significantly reduce the energy consumption of monitoring sources and the amount of data flooding streaming services by dynamically adapting, in place and inexpensively, the metric collection and dissemination rate when stables phases in the data stream are detected. To achieve this, we introduce a low-cost approximate and probabilistic algorithmic learning model capable of capturing runtime knowledge from the monitoring data evolution and

variability, thus allowing our adaptive monitoring techniques to adjust the metric collection and dissemination rate of the monitoring source based on the confidence of the algorithmic model to correctly estimate what will happen next in the data stream. Specific consideration is taken to fine-tune the algorithmic model at runtime by introducing adaptive parameter weighting, trend detection and seasonality behavior enrichment so that our algorithms immediately identify abrupt transient changes in the metric stream evolution and overcome any lagging effects in the estimation process.

Afterwards, we introduce AdaM, a lightweight framework designed and developed to be embeddable in the software core of monitoring sources (e.g., IoT devices) to provide adaptive monitoring by incorporating the low-cost approximate monitoring techniques for adaptive sampling and filtering, developed in the scope of this thesis. Next, we introduce ADMin, a plugin developed to extend the functionality of the AdaM framework by offering model-based adaptive dissemination to efficiently adapt, in place, the rate at which monitoring sources disseminate metrics. We conclude by evaluating the efficiency and accuracy of our framework by performing a thorough experimentation study with testbeds using real-world data from cloud applications, wearables, photovoltaics and intelligent transportation services. Results show that AdaM is able to dynamically adapt both the collection and dissemination rate of a monitoring source; while achieving a balance between efficiency and accuracy. Particularly, for the evaluated testbeds, AdaM can reduce energy consumption by at least 83%, data volume by 71%, shift detection delays in model-based dissemination by 61% while maintaining accuracy always above 89% in comparison to other state-of-the-art adaptive frameworks.

# Acknowledgments

I would like to thank my thesis advisors, Professors Marios D. Dikaiakos and George Pallis for the opportunity they have given me to work under their supervision and the immense trust they keep showing towards me. I am sincerely grateful for their advise, guidance, patience and ample time devoted to not only the authorship of this thesis but for the knowledge and motivation you have given me to become a better researcher.

I feel the need to express gratitude to my friends and colleagues from the University of Cyprus and, in particular, the Laboratory for Internet Computing (LInC). One is only as great as the people next to him. There is no need to list them here one-by-one, they know who they are and how thankful I am.

I would also like to thank my family for their unconditional love and never-ending support which continues to provide me with the strength required to fulfill my dreams and ambitions. Without you, this would not be possible. Especially, I would like to thank my wife, Christa. Without you, the entire process would have been meaningless and intolerable, because success is only meaningful if you have someone to share it with.

Finally, i would also like to take the opportunity to acknowledge the support I have received from the following EU co-funded projects: CELAR FP7 (FP7- ICT-2011-8), PaaSport FP7 (FP7-SME-2013) and Unicorn H2020 (H2020-ICT-2016-1).

# Thesis Contributions

This thesis is founded on the knowledge acquired by my involvement in the authorship of the following journal articles and conference papers:

## Journal and Magazine Articles

1. **"[In Minor Revision] Low-Cost Adaptive Monitoring Techniques for the Internet of Things"**, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **IEEE Transactions on Services Computing**, 2017.

2. **"Monitoring Elastically Adaptive Multi-Cloud Services"**, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **IEEE Transactions on Cloud Computing**, vol. 4, 2016.

3. **"Evaluating cloud service elasticity behavior"**, G. Copil, **D. Trihinas**, H-L Truong, D. Moldovan, G. Pallis S. Dustdar, and M. D. Dikaiakos, **International Journal of Cooperative Information Systems (IJCIS)**, 2015.

4. **"[Invited Manuscript] Enabling Interoperable Cloud Application Management through an Open Source Ecosystem"**, N. Loulloudes, C. Sofokleous, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **IEEE Internet Computing**, vol. 19(3), 2015.

## Conference and Workshop Proceedings

5. **"Improving Rule-Based Elasticity Control by Adapting the Sensitivity of the Auto-Scaling Decision Timeframe"**, **D. Trihinas** and Z. Georgiou and G. Pallis and M. D. Dikaiakos, **In Third International Workshop on Algorithmic Aspects of Cloud Computing (AlgoCloud 2017)**, Vienna, Austria, Sep. 2017.

6. **"ADMin: Adaptive Monitoring Dissemination for the Internet of Things"**, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **2017 IEEE International Conference on Computer Communications (IEEE INFOCOM '17)**, Atlanta, GA, USA, May 2017.

7. **"AdaM: an Adaptive Monitoring Framework for Sampling and Filtering on IoT Devices"**, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **2015 IEEE International Conference on Big Data (IEEE BigData '15)**, pages 717-726, Santa Clara, CA, USA, Oct. 2015.

8. **"[Best Paper] ADVISE – a Framework for Evaluating Cloud Service Elasticity Behavior"**, G. Copil, **D. Trihinas**, H-L Truong, D. Moldovan, G. Pallis S. Dustdar, and M. D. Dikaiakos, **12th International Conference on Service Oriented Computing (ICSOC '14)**, Paris, France, 2014.

9. **"JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud"**, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '14)**, pages 226-235, Chicago, IL, USA, May 2014.

10. **"c-Eclipse: An Open-Source Management Framework for Cloud Applications"**, C. Sofokleous, N. Loulloudes, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **EuroPar 2014 Parallel Processing**, Porto, Portugal, 2014.

11. **"[Demo] Managing and Monitoring Elastic Cloud Applications"**, **D. Trihinas**, C. Sofokleous, N. Loulloudes, A. Foudoulis, G. Pallis and M. D. Dikaiakos, **International Conference on Web Engineering (ICWE '14)**, pages 523-527, Toulouse, France, 2014.

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

## 1.1 Thesis Motivation

Monitoring is the act of observing the behavior of a system for the purpose of ensuring that certain properties are maintained [1]. These properties may be *quantitative*, meaning that the observed behavior is measurable (e.g., system performance) or *qualitative*, meaning the observed behavior is evaluated towards its expected outcome (e.g., data integrity) [2]. The main purpose of monitoring is to gain *actionable insights* that enable the entity of interest –either this may be a developer, an operator, or even another system– to take appropriate actions to better manage and maintain the deployed system. For most enterprises, information is generated, stored and processed along familiar routes [3]. Information related to system utilization, customer transactions and social activity is monitored from its birth, lodged in small or large data stores and analysed to generate reports for decision-making across various management chains, to manage health and resource requirements, as well as, to investigate and recover from failures. For these reasons, monitoring is a crucial part of any system under deployment [4].

But the predictable pathways shaping information propagation are changing: *the physical world is becoming an information system itself* [5]. Sensors and actuators with "smart" processing capabilities embedded in internet-enabled physical devices are becoming the tools for understanding the complexity of the global inter-connected world we inhabit [6]. These devices capture and exchange continuous data streams with other network-enabled devices, services and human-beings, forming, what is known as, the Internet of Things (IoT) [7]. From smart transportation and home appliances, to retail innovations, surveillance, and manufacturing, we are starting

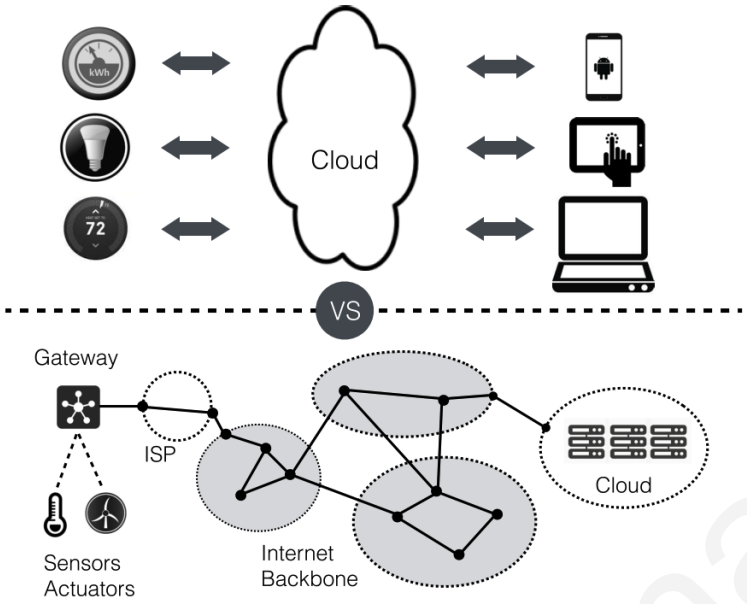



*Figure 1.1: The IoT developer perspective (upper image) and the reality (lower image) behind the cloud powering the Internet of Things*

to see intelligence aggressively deployed to produce real-time analytic insights. With intelligence comes the need to compute at the edge, and a variety of IoT offerings are opening up new and disruptive opportunities [8]. However, to produce such an unprecedented wealth of insights intense processing and constant data dissemination over the network are still required [9] [10]. This results in increased energy consumption for IoT devices while cloud and streaming services that are constantly overwhelmed with IoT data are struggling to be effective [11].

As IoT continues to spread across almost all industries it triggers a massive influx of big data [12] [13]. This data delivers knowledgeable insights that hold the potential to scrutinize, model and predict individual and collective behavior. According to Gartner [14], 8.4 billion IoT devices will be in use by the end of 2017, up 31% from 2016, and will reach 21 billion by 2020. As an illustrating example, the expectations for self-driving autonomous vehicles is that 4GB of telemetry data will be generated every day while requiring real-time processing for the vehicle to make correct and timely decisions [15]. Multiply this number by thousands, millions or even billions of vehicles and other sensing devices scattered across multiple locations in an urban environment and even powerful organizations equipped with high-performance distributed data engines such as Apache Spark and Hadoop reach their limits as the velocity and volume of sensed data keeps increasing [16] [17]. Therefore, it is no wonder that these billions of "things" already impact the digital universe with

| **Raspberry Pi (v2 model B)** | **Power** |
|---|---|
| Idle state | 420mA (2.1W) |
| Max CPU load (400%) | 800-1100mA (4W) |
| Max CPU load (400%) + disk I/O | 900-1200mA (4.5W) |
| Max CPU load (400%) + disk I/O + metric value dissemination over the network | 1250-1400mA (6.25W) |

*Table 1.1: IoT micro-controller power consumption benchmark test*

monitoring data generated from IoT devices accounting for 2% of the digital data in 2012 with a projection that by 2020 it will rise well above 12% [18].

With trends in cloud monitoring moving towards providing *Monitoring-as-a-Service* to ease end-user management of the monitoring infrastructure it is only natural that IoT follows [19]. However, despite attempts to augment IoT devices which have limited processing capabilities with the power of the cloud there still exist numerous inhibitors masked under constant *data movement* such as bandwidth limitations, network latencies, pricing and continuous location awareness [20]. This is particularly evident in Figure 1.1, where although IoT applications usually view the cloud as the center between all connected devices, the harsh reality is that the entire Internet backbone may sit between the cloud and the monitoring sources scattered across the logical extremes of the monitored IoT network [21]. Therefore, extending cloud monitoring beyond providing de-centralized endpoints (e.g., monitoring servers) in front of high performance data stores to accommodate the challenges of sensing the physical world while still enabling Monitoring-as-a-Service is a complex task that must be tackled by the next generation of IoT and cloud solutions [22].

Thus, it seems only inevitable that monitoring data must be processed at the edge of the network for shorter response times, more efficient processing and significantly less network pressure. Edge computing is a term coined to reflect the distribution of applications, services and (data) processing to sources constituting the logical extremes of a network [23]. Instead of IoT devices like activity monitors, drones and self-driven vehicles simply sensing the physical world and needing to constantly communicate over the network with a central service for instructions, monitoring data analysis is performed in place to accomplish timely vital tasks and reduce data movement. Hence, edge computing is, in turn, often referred to as *the brains fueling*

*the new generation of the Internet of Things* [24].

As IoT devices become "smarter" by embracing the capabilities provided by edge computing, battery life becomes a luxury as intense processing results in increased energy consumption [25] [26]. To date, a number of monitoring tools claim to be suitable for IoT settings as they are capable of running on limited resources while metric processing is left to powerful monitoring servers and analytic insight processing engines. For instance, traditional system (e.g., Nagios [27], Ganglia [28], Zabbix [29]) and cloud monitoring tools (e.g., CloudWatch [30], AzureWatch [31], Ceilometer [32]) have been offering for physical hosts and virtual machines, performance metrics such as CPU, memory and network utilization for years now. Collecting these metrics, for server monitoring, is merely the task of parsing OS written files (e.g., from /proc/* for Linux OS's) while popular application probing APIs (e.g., New Relic [33], JCatascopia [34]) extend the reach of cloud monitoring tools to provide application-level monitoring as well.

However, this assumption is far from true in cases where the monitoring task must sense the physical world, collect external stimulus and then perform, in place, costly analysis [35]. Table 1.1 presents such a case where power consumption triples as processing and data dissemination load are added to an idle IoT device. This is even more evident in IoT settings comprised of mobile and edge devices equipped with small batteries [36]. For example, the new generation of activity monitoring wearables differ from their predecessors in providing heartrate monitoring. Heartrate monitoring is based on photoplethysmography (PPG) signal analysis where green LEDs scatter light on the wrist which is reflected by the arteries as the heart pumps blood. This results in an AC signal processed by the wearable with peak detection algorithms to estimate the current heartrate. This intense process is the main reason battery life on the aforementioned devices has dropped from 5-7 days to 3-5 days [26] [37]. Therefore, monitoring tools without self-adapting capabilities to reduce energy consumption and network traffic are unsuitable for IoT settings and it is no wonder why taming data volume and velocity, as well as, energy efficiency are considered as great challenges transitioning from the big data era to IoT [38] [39].

## 1.2 Thesis Scope

In this thesis, we tackle *real-time data processing* and *energy-efficiency* on the edge of monitoring networks by developing **low-cost approximate and adaptive monitoring techniques** as the remedy to these challenges. It is simply inefficient to sense the physical world and propagate all monitoring data for analysis to the cloud, especially in periods where the monitoring data stream volatility is low; doing so requires a great deal of processing and energy consumption on the monitoring source itself. In turn, all the back-and-forth communication between monitoring sources and the cloud, negatively impacts performance while quickly saturating the monitoring network bandwidth and thus, preventing scalability. Hence, if a degree of inaccuracy can be tolerated, approximate monitoring techniques can significantly reduce the energy consumption of monitoring sources (e.g., IoT devices) and the amount of data vastly streamed to cloud consuming services.

In this thesis we explore the use of low-cost approximate and adaptive techniques such as **adaptive sampling, filtering and model-based dissemination**, to tackle the challenges introduced in monitoring networks with processing conducted at the edge. Thus, particular interest is put in dynamically adapting the *metric collection and dissemination rate* of monitoring sources which are the two prime energy drains for embedded and mobile sensing devices. With "approximate" we refer to techniques basing their decision mechanisms on an estimation model capturing and predicting the runtime evolution of the monitoring data stream within certain accuracy guarantees. In turn, with "adaptive" we refer to techniques capable of adapting the properties of a monitoring source (e.g., IoT device) based on the predicaments of the estimation model. Emphasis is given on presenting techniques that are designed for smart and battery-powered devices with limited processing capabilities and which can be used *in place* and *inexpensively*.

**Adaptive sampling** is the process of dynamically adjusting the sampling rate[1] based on a runtime estimation model following the current monitoring data stream evolution, such that when stable phases in the data stream are detected, the sampling rate is reduced to ease on device processing and energy consumption. Hence, in the case of a wearable device featuring heartrate monitoring, if stable phases exist in the

[1] Also referred throughout the literature as the sensing rate, collection rate and monitoring rate

data stream featuring user heartrate data, then the device will spend more time in idle state instead of performing costly analysis that drains limited processing and energy resources. In turn, when the evolution of the monitoring data fluctuates in time, the sampling rate is increased to immediately capture and notify users of event violations and possible risks.

On the other hand, **adaptive dissemination** is the process of applying approximation techniques to sensed monitoring data to reduce the communication overhead by suppressing the dissemination of consecutive datapoints with "little" change in their metric values. Hence, energy required to transmit values over the network is reduced in favor of exact precision and, at the same time, the volume and velocity of data streamed to cloud consuming services is regulated to ease processing. A low-cost approximate technique tailored for this task is *filtering*. A monitoring source featuring filtering capabilities, does not transmit the latest collected datapoint(s) if their values have not "changed" more than a range of values since last reported. How much "change" is considered as "little" depends on the type of filter with **adaptive filtering** being the process of dynamically adapting the range of accepted values to achieve certain filtering guarantees while at the same time obeying the accuracy guarantees given by the user. Another approach is **model-based dissemination** where, instead of disseminating metric values, we favor modeling the monitoring data stream so that consuming entities infer metric values from the model, triggering a model update only when shifts in the stream evolution render it as inaccurate.

Despite advancements in the field, current approximate and adaptive techniques for monitoring networks with processing conducted at the edge still require mature-ness as in many cases these techniques: (i) require excessive profiling to configure optimal framework parameters, a task difficult for users; (ii) present large runtime footprint in regards to energy consumption or resource utilization reducing the benefits of introducing adaptiveness in the end; (iii) fail to acknowledge abrupt and transient shifts in the evolution of monitoring data or assume that once determined there will be no distribution shifts in the monitoring stream value base; or (iv) require coordination from server-side components.

In light of striking a *balance between efficiency and accuracy*, for the adaptive techniques introduced in the scope of the thesis, we have developed a low-cost approximate and probabilistic algorithmic learning model capable of capturing runtime knowledge from the metric stream evolution and variability. With this, the metric

collection and dissemination rate of a monitoring source will be dynamically adjusted based on the confidence of each technique to correctly estimate what will happen next in the metric stream. Specific consideration is taken to fine-tune each algorithmic approximate technique, at runtime, by introducing *adaptive parameter weighting*, *trend detection* and *seasonality behavior enrichment* so that our algorithms immediately identify abrupt transient changes in the monitoring data stream evolution and overcome any lagging effects in the estimation process. To increase the applicability of our adaptive techniques, we introduce **AdaM**, a lightweight framework embeddable in the software core of monitoring sources (e.g., IoT devices) that provides adaptive monitoring by incorporating the low-cost approximate monitoring techniques for adaptive sampling and filtering. Next, we introduce **ADMin**, a plugin developed to extend the functionality of the AdaM framework by offering model-based adaptive dissemination to efficiently adapt, in place, the rate at which monitoring sources disseminate metrics. After undergoing a thorough experimentation study on testbeds using real-world and publically available data from cloud applications, wearables, internet security services, photovoltaics, and intelligent transportation services; results show that, in comparison to State-of-the-Art adaptive techniques, AdaM is able to achieve a balance between efficiency and accuracy.

## 1.3 Thesis Statement and Contributions

In this thesis we argue that,

> *if a degree of inaccuracy can be tolerated, low-cost approximate and adaptive monitoring techniques that are capable of dynamically adjusting the metric collection and dissemination rate, can effectively tackle real-time data processing and energy-efficiency on the edge of monitored networks.*

Towards advancing the State-of-the-Art, this thesis makes the following contributions:

- We present a comprehensive overview of the monitoring tool landscape with focus on monitoring and managing distributed application deployments at scale. Particular emphasis is given in presenting the limitations the current monitoring tools have when introduced to the unique settings of the IoT

world. Monitoring at the edge implies significant costs which affects monitoring sources due to increased resource and energy demands for data sensing and processing, and network scalability due to increased data movement, network latencies, and limited bandwidth. In light of these challenges, we explore the limitations of following a fixed periodicity approach for both metric collection and dissemination which are considered the prime energy drains for mobile edge devices. We then, formally define the problem along with a common notation and introduce the concepts of approximate and adaptive monitoring.

- We develop three low-cost approximate and adaptive monitoring techniques to tackle the challenges introduced in monitoring networks with processing conducted on the edge. **Adaptive Sampling** dynamically adjusts the intensity of the metric collection process depending on the evolution and variability of the monitoring data stream to reduce the overhead imposed to a monitoring source; **Adaptive Filtering** discards consecutive metric values that do not differ more than a range of values that is adjusted to given accuracy guarantees in order to reduce the volume of data disseminated over the network; while rather than transmitting metric values, **Model-Based Dissemination** favors modeling the monitoring data stream so that consuming entities infer metric values from the model, triggering a model update only when shifts in the stream evolution render the model inaccurate.

- We introduce the design of a low-cost approximate and probabilistic learning model to capture runtime knowledge of the monitoring data stream evolution and variability. In order for the model to immediately identify abrupt transient changes in the data stream evolution and overcome any lagging effects in the estimation process adaptive parameter weighting, trend detection and seasonality behavior enrichment are introduced to the estimation model.

- We design and develop AdaM, a lightweight framework embeddable in the software core of monitoring sources (e.g., IoT devices) that provides adaptive monitoring by incorporating the developed techniques introduced in this thesis for adaptive sampling and filtering. Next, we introduce ADMin, a plugin developed to extend the functionality of the AdaM framework by offering model-based adaptive dissemination. Thus, AdaM is able to dynamically ad-

just both the rate at which monitoring metrics are collected and disseminated to interested receiving entities based on extracted runtime knowledge capturing the evolution and variability of the metric stream. By accomplishing this, energy consumption and data volume are reduced, allowing monitoring sources, such as edge devices, to preserve energy and resource utilization, while at the same time ease processing on monitoring data receiving services. AdaM has been initially developed in Java for IoT devices (e.g., Raspberry Pi's, Android devices) but as it features no external source code dependencies, we show that, it can also be ported to other real-world settings and popular programming frameworks.

- We evaluate the efficiency and accuracy of the developed adaptive monitoring techniques by performing a thorough experimentation study that compares AdaM to other State-of-the-Art adaptive frameworks. All deployed testbeds use real-world and publically available data from cloud applications, wearables, internet security services, photovoltaics and intelligent transportation services. To the best of our knowledge, this is the only study extending the experimentation to include, beyond an accuracy evaluation, both an on device performance and energy consumption comparison. For the evaluated testbeds, our results show that AdaM can reduce energy consumption by at least 83%, data volume by 71%, shift detection delays in model-based dissemination by 61%, while always maintaining accuracy above 89% in comparison to other state-of-the-art adaptive frameworks. Thus, big data services consuming IoT data can truly benefit in terms of lower monitoring costs, achieve greater scalability and efficiently utilize underlying resources when embedding AdaM in the software core of their monitoring sources.

## 1.4 Thesis Organisation

The remainder of this thesis is structured as follows:

**Chapter 2** provides context for the thesis by describing background information and concepts related to the Internet of Things and Edge Computing, while highlighting advancements in Monitoring Tools relevant to the thesis contributions.

**Chapter 3** formally states and defines the research challenges tackled by the thesis and introduces the concepts of low-cost approximate and adaptive monitoring.

**Chapter 4** discusses relevant work in relation to approximate and adaptive monitoring by introducing the current State-of-the-Art after trawling the literature and industry advancements.

**Chapter 5** introduces the AdaM framework developed to provide adaptive monitoring for IoT devices. Particular focus is given in presenting the algorithmic logic behind AdaM's adaptive sampling and filtering techniques, while afterwards a thorough experimentation study is conducted to evaluate AdaM efficiency and accuracy towards other adaptive frameworks.

**Chapter 6** introduces the ADMin plugin developed to extend the functionality of the AdaM framework to support model-based adaptive dissemination. Focus is given to show how the AdaM estimation model can be extended to support model-based dissemination while afterwards an evaluation is conducted to compare ADMin to other adaptive dissemination frameworks.

**Chapter 7** concludes the thesis and outlines future work as a direct product of the thesis contributions.

## 1.5 Thesis Assumptions

In this thesis we focus on adapting the collection and dissemination rate of a monitoring source to allow the sensing and network controlling units of the monitoring source to remain for longer periods of time in idle state when the workload does not change. While monitoring data collection and dissemination constitute the primary energy for mobile and embedded devices, certain control functions prevent either the sensing unit and/or the network controlling unit to remain in idle state. For example, physical devices may consume energy not only for metric transmission, but also when conducting control operations driven by network layer protocols (e.g., maintaining routing tree, probing ongoing communication at the MAC layer) [9]. This is

often not mentioned when introducing adaptive monitoring techniques, leading to a misguided and overly optimistic view of software-oriented adaptive techniques. Adapting the network layer control functionality, which is tightly coupled to the underlying hardware and network, is out of the thesis scope. Therefore this assumption holds for this thesis as well. Nonetheless, *when evaluating our adaptive monitoring techniques, real-world testbeds will be deployed and extensive experimentation related to device energy consumption will always be present*.

There exist cases in the edge computing realm where for low-capability devices, a single thread is dedicated to the monitoring task. This means the same periodicity is used to collect datapoints for all metrics reported by the device. In this case, adapting the periodicity per metric, will result in unexpected behavior. For example, let us consider a device monitoring the production of a photovoltaic (PV) panel with a single thread used to collect both the current ($I_{DC}$) and voltage output ($V_{DC}$). An adaptive sampling technique may suggest that for the current, the periodicity should be 2s while for the voltage, 5s is adequate. At this point multiple strategies may be considered: should 2s or 5s be used?, should the average be used?, or, should a particular metric be prioritized? This specific case is often not mentioned when trawling the literature. It can be advocated that this is usually evident in remote sensing devices where metric probing (e.g., PV panel production) is not energy or compute intensive and therefore will not greatly benefit from adapting the metric collection periodicity. Thus, a fixed periodicity is often used. This assumption will hold for this thesis as well. However, *it is this type of case where enabling (adaptive) filtering and model-based adaptive dissemination, with or without (adaptive) sampling, can be very beneficial for metric digesting and storage entities, which without data volume reduction are quickly overwhelmed with data and struggle to be effective*.

When trawling the literature, a number of advanced adaptive monitoring techniques assume a one-dimensional (univariate) data stream, while in the case where more than one metrics of interest are collected by the monitoring source, each is published as a separate and independent data stream. For instance, an IoT device reporting weather conditions may publish the following metrics as independent streams: temperature, humidity and air pollution. Nonetheless, not all metric classes can be described in a single dimension (e.g., GPS coordinates, displacement, velocity and acceleration), and considering each dimension as a separate metric stream is both meaningless and error prone. However, support for high-dimensionality

(multivariate) metric classes is not straightforward and as dimensionality increases the complexity of the algorithmic process increases as well, and may harm the introduction of adaptivity in the end. Assuming univariate data streams as input for the introduced adaptive techniques holds for this thesis as well. Therefore, although we will not bound the number of data streams published by a monitoring source, we will limit the discussion to univariate metric classes. However, we note that the formal definition of each introduced low-cost adaptive monitoring technique holds for multivariate monitoring data streams as well.

Finally, it must be noted that in addition to energy consumption and the "big data" generated by edge devices, security and data privacy are another set of challenges that must be addressed to increase the wider adoption and applicability of IoT. Protecting user privacy is vital, especially, in cases where data is collected from device owners (e.g., heartrate, blood pressure, activities, location) and the actions performed by the device (e.g., operating machinery in production chain, vehicle steering) are sensitive. The current State-of-the-Art for privacy preserving in data publishing is differential privacy where sensitive data is perturbed by a randomized algorithm, so that the output remains roughly the same even if any single tuple in the input data is arbitrarily modified. It has been shown that the addition of adaptive sampling and filtering to differential privacy can potentially increase protection, or at least provide the same level of privacy guarantee, by removing consecutive datapoints with high correlation [40] [41]. This prevents adversaries from inferring both datapoint values and data expectancy (when datapoints are released) from the protected data, while also reducing the privacy cost by applying differential privacy to a subset of the data stream. Nonetheless, privacy preserving alone cannot limit all potential attacks without appropriate security enforcement measures (e.g., secure channel establishment). For instance, with the addition of adaptive techniques to remove correlated data, adversaries via side-attacks may not be able to infer datapoint values but can assume datapoint significance and network traffic patterns [42]. Therefore, while this thesis does not deal with security and privacy-preserving, it goes without mentioning that *any developed adaptive technique for monitoring data sources in the IoT and Edge Computing realm must complement and not compromise enforced security and privacy-protecting mechanisms*.

# Chapter 2

# Background

## 2.1 The Internet of Things

The *Internet of Things* is a novel network paradigm referring to the pervasive presence of a variety of everyday objects, denoted as "things", that are inter-connected to achieve certain design goals [43]. Such goals usually envision automating certain jobs or tasks (e.g., smart lighting, self-tracking) to improve and impact our everyday lives and work. The concept of "things" in the network infrastructure refers to any physical, or virtual, participating actors such as real world objects, human beings, and intelligent software agents [44]. The purpose of the IoT is to create an environment in which the basic information from inter-connected autonomous actors can be efficiently shared with others in real-time.

There are many definitions of the "Internet of Things" in the research and relevant industrial communities, with the term first coined by Kevin Ashton (1999) to describe how the Internet will, someday, be connected to the physical world [45]. However, Ashton was simply inspired by Mark Weiser's vision (1988) for *ubiquitous computing*, who stated that the time where computing power will be made available by any device, at any location and in any format, is fast approaching [46]. Another popular definition is by Atzori et al. [47], denoting that merging together the words "Internet" and "Things" semantically means a world-wide network of inter-connected objects uniquely addressable, based on standard communication protocols. Nonetheless, although not a new concept, IoT is only now becoming a reality due to recent advances in *microelectronics* (e.g., nano bio-sensors [48], printed electronics [49]), *telecommunications* (e.g., RFID, NFC, Bluetooth 4.0 [38]) and *data-*

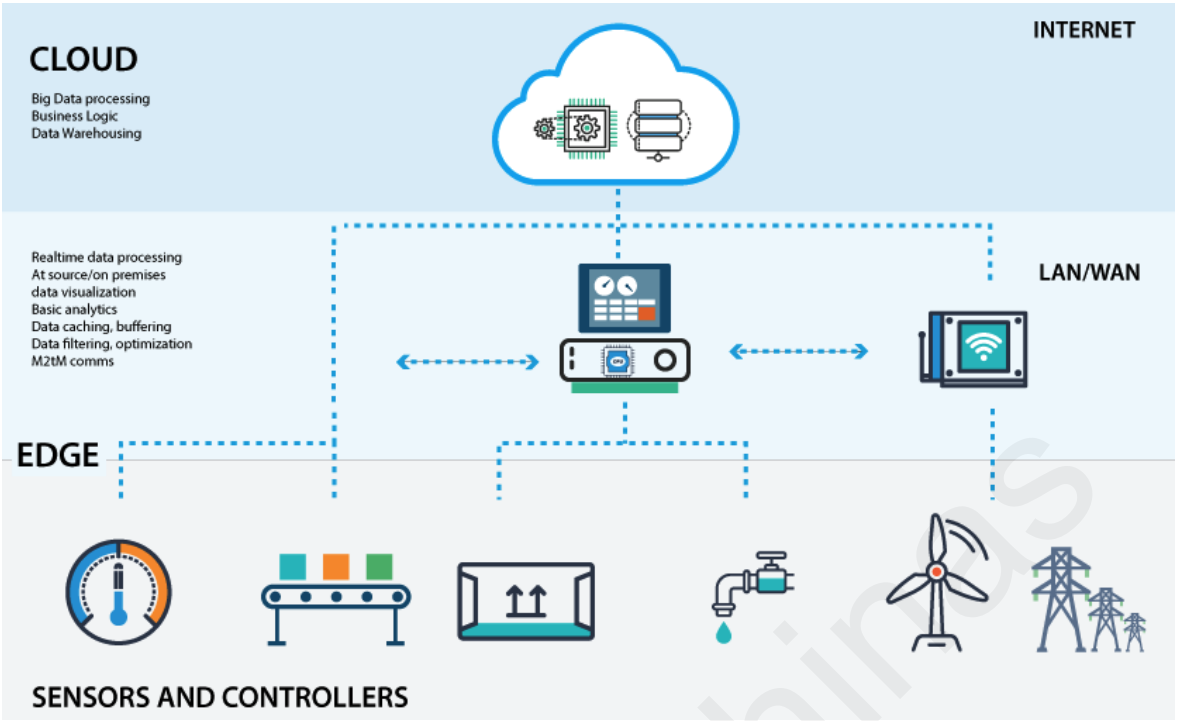



*Figure 2.1: The "Edge" in cloud computing*

*mining* (e.g., distributed AkNN [50], PCA on streams [51], location-based pattern matching [52], outlier/event detection [16], timeseries dimensionality reduction [53]).

Examples highlighting these advances in the multiple and different domains that the Internet of Things has infiltrated, include:

- **Intelligent transportation services (ITS)**, where smart wireless sensors and cameras embedded in roads, traffic lights, parking spots, public transportation and even citizen vehicles can help in reducing traffic congestion, carbon dioxide emissions, time to park, accident/incident reporting time but most importantly prevent incidents by reporting abnormalities and mechanical deficiencies [54]. Early ITS adopters include the cities of Dublin with its smart bus service used to detect and alert operators of road congestion and traffic incidents [55]; Toronto with its experimental smart traffic flow service reducing travel times by 26%; and Los Angeles with its smart parking service reducing, on average, for a 15 block radius carbon dioxide emissions by 730 tons [56].

- **The medicine and heath sector**, where wearable devices are monitoring human activity and biosignals with self-tracking data also capable of assisting doctors in taking more accurate and patient-centralized decisions [57]. For example, such is the case of a patient rushed to the ER with atrial fibrillation (a fast

and irregular heartbeat) where the ER doctors could only decide on the right treatment after tracker data confirmed his abnormal heart rate happened the same time the patient had a seizure [58]. Also pill-shaped microcameras are now capable of traversing the human digestive tract, sending back thousands of images to pinpoint sources of illness [59].

- **Smart metering and (renewable) power grids**, where low-power and bi-directional connectivity (e.g., zigbee, bluetooth 4.0, LORA) along with edge analytics embedding cognitive capabilities into meters, are being integrated by utility companies, manufactures and distributors to move beyond capturing operational costs towards ensuring resource efficiency, accurate diagnostics and optimized billing [60]. In particular, smart meters, feeding into power grids, are allowing to better manage bi-directional power flows in and out of grids, and give consumers the insights needed to understand energy infrastructure investments, such as in the case of micro-controllers monitoring the production and health of PV panels to detect and predict consumption patterns and quality degradation due to external conditions (e.g., the weather) [61].

- **Agriculture**, where precision farming equipment connected through wireless links to data collected from remote satellites and ground sensors can take into account crop conditions and adjust the way each individual part of a field is farmed - for instance, by spreading extra fertilizer on areas that need more nutrients [62]. In turn researchers have created devices to track the movement, biosignals and interactions between cows to improve cattle health and increase profitability through breeding between cows in narrow estrus windows lasting only a few hours [63].

- **Retailing**, where customer profile data (e.g., interests, shopping trends, recent purchases) along with readings from smart sensors, RFID tags and bluetooth beacons interacting with customers through monitors or their smartphones to assist, increase sales, provide additional product information, offer discounts at points of sale and close purchases.

| Device | CPU Speed | Memory | Price |
|---|---|---|---|
| Intel NUC | 1.3 GHz | 16 GB | $300 |
| Typical Phones | 2 GHz | 2 GB | $300 |
| Discarded Phones | 1 GHz | 512 MB | $40 |
| BeagleBone Black | 1 GHz | 512 MB | $55 |
| Raspberry Pi (model B) | 900 MHz | 512 MB | $35 |
| Arduino Uno | 16 MHz | 512 MB | $22 |
| mbed NXP LPC1768 | 96 MHz | 32 KB | $10 |
| Activity Wearable (Fitbit) | 32 MHz | 128 MB | $150 |

*Table 2.1: The wide spectrum of IoT platforms are constraint in terms of processing capabilities*

## 2.2 Edge Computing

With no doubt, the backbone powering the Internet of Things is the cloud [21]. Over the past decades, the cloud is revolutionizing the ICT industry to the point where any person, with even basic technical skills, can access via the internet, vast and scalable computing resources by shifting IT spending to a pay-as-you-use model [64]. For small businesses and startups, this well-established argument is sound. Cloud computing eliminates capital expense of buying hardware and diminishes costs for configuring and running on-site computing infrastructures of any size [65] [66]. Thus, riding on the early popularity of the cloud, IoT developers saw potential for their applications and peripherals (e.g., sensors, actuators) generating massive monitoring streams but residing on hardware featuring limited processing capabilities (e.g., Arduino, Raspberry Pi, Beagle Board), as depicted in Table 2.1. These "things", as depicted in Figure 2.1, communicate directly with the cloud, or through mobile gateways denoted as cloudlets [67], and interact with each other through web services, with developers naturally adopting the cloud as a universal and centralized computation and storage backend for the intensive analytic jobs required by IoT services [68]. Recent studies show that indeed more than 50% of today's IoT services reside in the cloud[1] with IoT developers revealing that reduced capital expenses and simplified resource management are driving cloud adoption [69] [70].

Although, performing computing tasks on the cloud is an efficient way for data

[1] This number increases to 67% for production-ready services

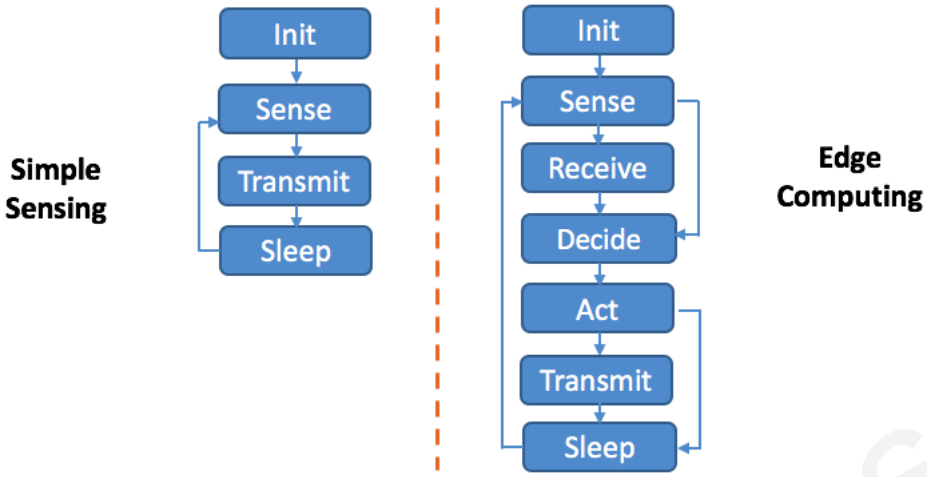
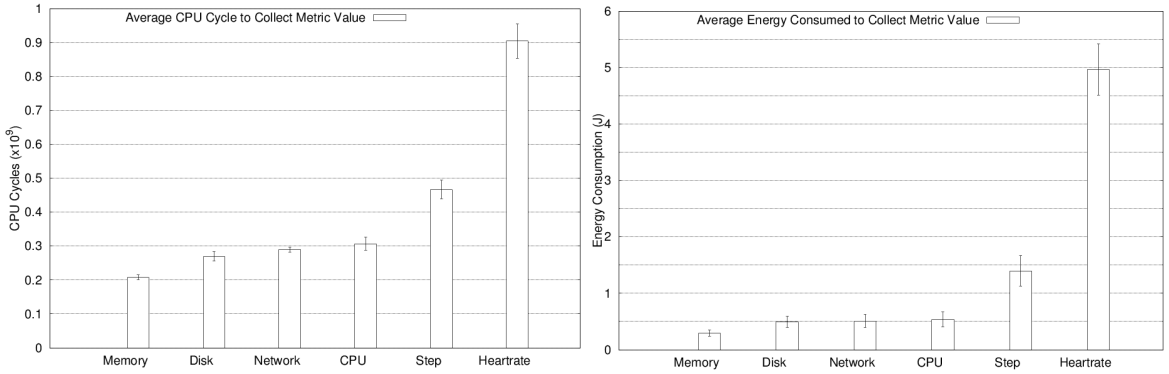



*Figure 2.2: Simple Sensing vs Edge Computing*

processing, since the computing power of the cloud outclasses the capability of mobile and remote IoT devices, compared to the fast developing data processing speed, network bandwidth remains standstill [71]. Thus, with the growing volume of data generated at the logical extremes of IoT networks, data dissemination is becoming a bottleneck constraining the cloud computing paradigm. To illustrate this challenge, consider that a Boeing 787 generates about 5GB of data per second, although the link bandwidth between the airplane and the base station on the ground is not large enough for such data transmission [72]. In turn, Uber, the world's largest on-demand taxi service, receives approximately 3 million car telemetry updates per second and monitors more than 500 million metrics for its daily taxi services [73]. If all the data needs to be sent to the cloud for processing, response time will be large. Not to mention that current network bandwidth and reliability would be challenged for its capability of supporting large numbers of data sources.

Thus, it seems only inevitable that monitoring data must be processed at the edge of the network for shorter response times, more efficient processing and significantly less network pressure. Edge computing refers to the enabling technologies allowing computation to be performed at the logical extremes of the network, such as on downstream data, on behalf of cloud services, and upstream data, on behalf of IoT services [74]. In this context, *edge sources differ from traditional sensing devices in that sensory data are processed in place and converted from raw signals to contextually relevant information*, as depicted in Figure 2.2. Thus, the rationale of edge computing is that computing should happen at the proximity of the data source with the "Edge" constituting any computing and network resources along the path between



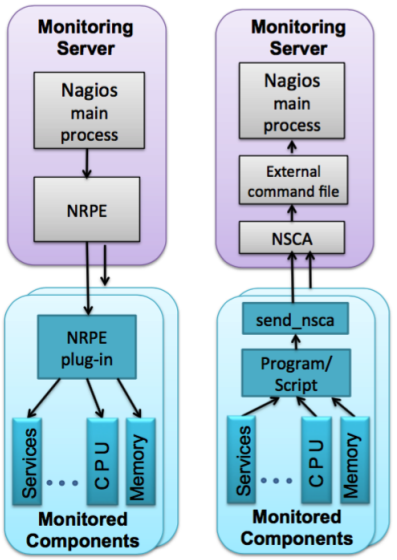

*(a) Compute Cycles* *(b) Energy Consumption*

*Figure 2.3: Average compute cycles and energy consumption comparison for metric collection on an IoT device with ARM processor (1 core 32MHz, 128MB RAM) and a 35mAh battery*

data sources and the cloud [10]. However, computing at the edge does not come without challenges. Sensing the physical world, requires collecting external stimulus and then performing, in place, costly analysis [36]. In particular, the monitoring task is strained from limited processing capability devices while, at the same time, intense processing leads to increased energy consumption, and thus, less battery life [75]. Figure 2.3 depicts a comparison between collecting system-level metrics extracted from OS-written files and metrics which are the process of sensing and processing external stimulus (e.g., step counting, heartrate monitoring) obtained from an Android Wear wearable device. Thus, offloading computation from the cloud to the edge possesses great benefits, however, while IoT hardware capabilities are projected to increase, battery capacity and bandwidth are, simply, not growing at the same rate [76].

## 2.3 Monitoring

Even before the advent and dominance of the cloud as a widely accepted computing paradigm, a number of monitoring solutions were already available for monitoring diverse ICT infrastructural resources such as networks, storage elements, physical servers and computing clusters [77]. To date, monitoring tools such as Ganglia [28], Nagios [27], Zabbix [29], IBM Tivoli [78] and GridICE [79] are popular among system administrators and are under constant development because of their diverse monitoring capabilities, simplicity and (under some deployment models) low-runtime




(a) Active mode (b) Passive mode

*Figure 2.4: Nagios active and passive mode*

footprint. Monitoring tools typically follow the client-server architectural paradigm. This paradigm is often attributed as an *agent-based monitoring architecture* where **agents**[2] are placed on each element intended to be monitored (e.g., physical host, virtual machine) and are responsible for the metric collection process and subsequent metric dissemination to monitoring server(s). In turn, **monitoring servers** are the entities responsible for digesting, processing and storing monitoring metrics to a database backend and for agent coordination. Additional functionality that monitoring servers may offer, include: (i) providing analytic insights by further processing collect metrics (e.g., Monalytics [80], MELA [81], VScope [82]); (ii) determining threshold violations set by system admins or users such as SLA violations (e.g., Cacti [83], LoM2HiS [84]); and (iii) providing various alerting channels for event detection such as SMS or email alerting (e.g., Nagios [27], Zabbix [29]).

**Centralized monitoring solutions** use either one monitoring server to poll all assigned agents for monitoring metrics [27] [83] [85]; or multiple monitoring servers are in use but with the caveat that these tools incorporate centralized components (e.g., directory services, schedulers, data stores) which are potential single-points-of-failure [86] [87] [88]. Among the most notable centralized solutions is Nagios [27]. Nagios is a monitoring tool which supports two deployment alternatives, as depicted in Figure 2.4. For Nagios either a central server is responsible for periodically pulling monitoring updates via a plugin executor (Nagios NRPE) installed within

[2] Agents are also referred to in the literature as monitors, data sources, or collectors

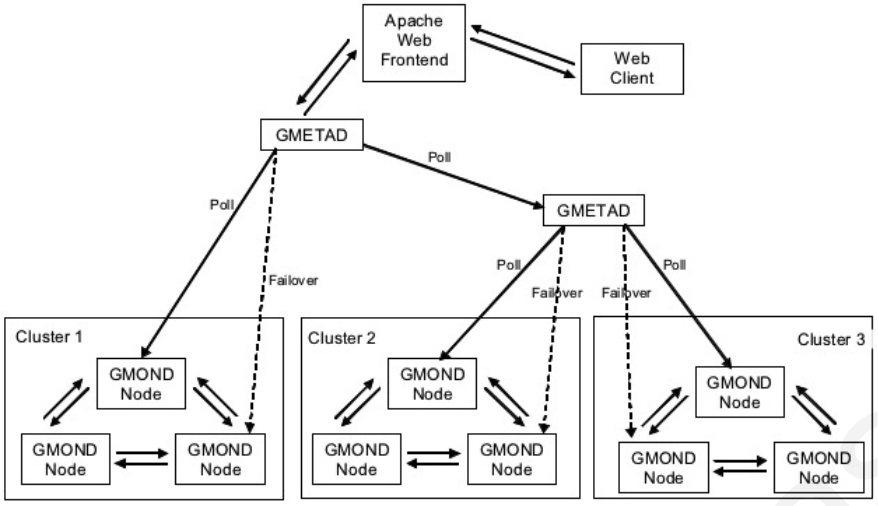



*Figure 2.5: Ganglia monitoring system architecture – Reprinted from [Massie et al., PC, 2004] [28]*

every monitored element (active mode) or a service check executor (Nagios NSCA) provides asynchronous push-based notifications from monitored elements to the central server (passive mode). Typically, centralized solutions (e.g., JAMM [85], PCMons [87]), are used for service status monitoring, although Nagios and Cacti (another open-source solution resembling Nagios) also offer resource utilization monitoring (e.g., CPU, memory usage).

To enhance scalability, robustness and fault-tolerance, an hierarchy of monitoring servers can be utilized to provide **distributed monitoring**. Monitoring tools following this approach include Ganglia [28], Zabbix [29], Lattice [89], GMonE [90] and OpsView [91]. Figure 2.5 depicts an abstract architectural overview of the open-source and popular Ganglia monitoring tool. In this setting, Ganglia servers (named gmetads) are connected together to form a tree-based hierarchy [28]. Each server is in charge of periodically aggregating metrics originating from agents (named gmonds) residing on nodes in the same cluster, via a multicast-based listening protocol, and then distributing aggregated cluster metrics further up the hierarchy. The obvious downside of this approach is that while the per-message processing overhead otherwise imposed to a central monitoring server is reduced, servers comprising the tree have to relay monitoring data for all monitoring tasks. Thus, monitoring servers closer-to-the-root are overloaded by relaying costs in large and distributed deployments. In [22] we show that hierarchical monitoring tools feature intrusive runtime footprints on user-paid resources as the deployment scales to include multiple metrics generated at high frequencies from more than 150 virtual instances.

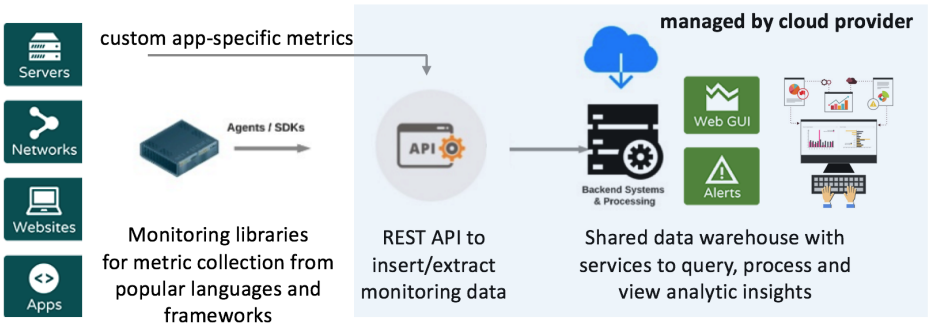



*Figure 2.6: Monitoring-as-a-Service at a high-level*

Cloud monitoring tools such as Amazon CloudWatch [30], Paraleap AzureWatch [31] and RackSpace CloudMonitor [92] provide monitoring capabilities to their virtual infrastructure service consumers. Monitoring information is usually accessible via APIs or intuitive graphical dashboards, although the metric collection methodology of these tools is considered proprietary and is not revealed. Additionally, although cloud service providers also enable custom application-specific metric collection, consumers must alter their applications accordingly to export metrics directly to the central monitoring service. Moreover, in many cases metric collection is limited to a fixed periodicity (e.g., CloudWatch periodicity is 5min; lowered to 1min with a premium fee). However, *monitoring information is only useful if collected in meaningful time intervals*, which vary for different metrics. Most importantly, despite the fact that these tools are easy to use, fully documented and well-integrated with the underlying platform, their biggest disadvantage is that they are commercial and proprietary which limits their operation to specific cloud providers. Thus, these tools lack in terms of *portability* and *interoperability*.

Nonetheless, what makes the aforementioned monitoring tools popular with cloud consumers along with other State-of-the-Art and third-party cloud application monitoring solutions such as New Relic APM [33], Datadog [93] and AppDynamics [94], is that they offer **Monitoring-as-a-Service (MaaS)**. Just as with other paradigms comprising the X-as-a-Service economy [95], MaaS eases the management of the monitoring infrastructure as service providers enable multi-tenant and scalable monitoring over the internet following a pay-as-you-use model [96]. MaaS providers typically offer users monitoring libraries supporting both IaaS and PaaS deployment models for various programming languages (e.g., java, ruby, python) and frameworks (e.g., tomcat, django, sql). These tools expose APIs for resource utilization metric collection, application performance monitoring and custom application-

specific metric collection. Metrics are usually periodically disseminated to shared data warehouses allowing users, depending on their subscription model, to perform queries on real-time and historical data, and produce analytic insights.

However, although recent trends in monitoring present a movement towards Monitoring-as-a-Service, there still exist a number of challenges, especially, in the field of the Internet of Things and Edge Computing, such as:

- The sensing challenge in the metric collection process where edge devices must sense the physical world by powering hardware peripherals (e.g., accelerometer) and perform analysis on the device itself to produce insights which is usually compute and energy intensive;
- The increased movement of monitoring data from (mobile) edge devices (e.g., wearables, traffic sensors) across geo-distributed availability zones to central monitoring and processing endpoints;
- Data restrictions and security risks of disseminating sensitive human, governance and application performance data across availability zones especially when moving from IaaS monitoring to application and client-side monitoring (e.g., health tracking, traffic patterns, customer behavior, sale transactions);
- Providing rapid elasticity in the form of horizontal, vertical or diagonal scaling of either the service or the monitoring infrastructure requires topology insights from sources spanning across the edge of the network in order to distribute load while baring in mind user strategies such are optimization for performance, costs and network latencies.

For these reasons various approaches mentioned in what follows attempt to provide monitoring solutions which propose architectures or features that dealt with the challenges of rapid elasticity, data movement and sensitive data dissemination.

An interesting approach is suggested by Calero et al. [96] who introduce MonPaaS. MonPaaS is a private cloud (on-premise), distributed and agent-less monitoring solution. For MonPaaS, instead of providing one shared monitoring processing tier, a dedicated monitoring server is allocated per application to provide consumers with secure and dedicated monitoring services, administration capabilities for the deployment and a graphical user interface. This approach is much similar to the Xen hypervisor approach where "Dom0" is the management entity providing services

to the virtualization stack with specialized management privileges for the rest of the virtual hosts [97]. A similar approach is also suggested by Koller et al. [98] who introduce a data-oriented pipeline for producing analytic insights where data collection is performed via VM introspection instead of the use of agents to reduce on-VM monitoring overhead. However, as with MonPaaS, while scalability is claimed, with the use of only one monitoring server per application, intra-service monitoring is bounded by the monitoring intensity and number of running instances.

In regards to elasticity, a handful monitoring solutions attempt to tackle the challenge of elasticity support but limit *portability* at different levels of the cloud. Specifically, Carvalho et al. [99] propose the use of passive checks by each physical host to notify the central monitoring server about the virtual instances that are currently instantiated. Katsaros et al. [100] extend Nagios through the implementation of *NEB2REST*, a REST event broker utilized to provide elasticity capabilities through an abstraction layer between monitoring agents and the management layer. A more interesting approach is followed by Clayman et al. [89]. The authors introduce *Lattice*, a cloud monitoring framework, which monitors not only physical hosts but also virtual instances. *Lattice* can be utilized to monitor elastically adaptive environments. In particular, the process of determining the existence of new VMs is performed at the hypervisor level. A *controller* is the responsible entity for retrieving a list of running VMs from the hypervisor, detecting if new VMs have been added or removed. However, while Lattice offers elasticity support it moves the dependency to the hypervisor layer as it is tightly coupled to Xen hypervisor. In turn, Lattice cannot monitor applications distributed across multiple cloud regions due to its limited multicast network communication model. Moreover, Lattice features an excessive runtime footprint. Another approach is Panoptes [101], which utilizes a **pub/sub communication model** between agents and servers to enhance private cloud monitoring performance. Specifically, Panoptes requires a broker (similar to a directory service), which acts as a central contact point for newly instantiated monitoring agents to: (i) contact and request a list of available monitoring servers; (ii) notify all monitoring servers of their existence, and (iii) wait for monitoring servers to start the subscription process, which is a significant overhead for rapidly elastic environments.

On the other hand, to cope with significant network latencies and bandwidth limitations, while supporting rapid elasticity along with providing autonomicity

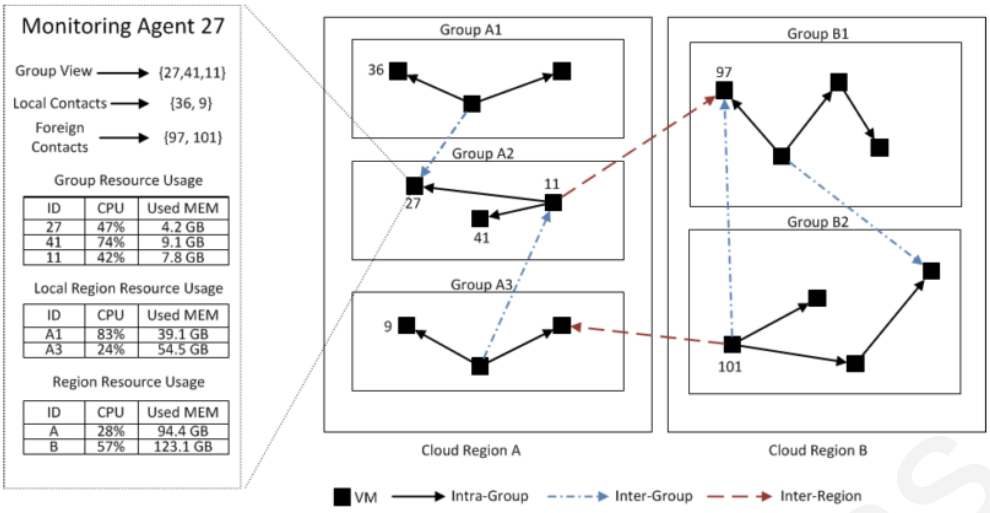



*Figure 2.7: P2P-based monitoring architecture – Adapted from [Ward et al., IEEE DIDC, 2014] [103]*

of the monitoring infrastructure in large and distributed topologies, a number of monitoring solutions suggest moving towards a **P2P monitoring architecture**. As depicted in Figure 2.7, in this setting monitored elements (e.g., VMs, sensors) form a gossiping overlay network to propagate monitoring information in order to ease the coordination of the monitoring infrastructure and query computation. Gossip denotes periodic and pairwise propagation of the current state between two monitored peers. When a new monitored element is instantiated, it must be bootstrapped based on some peer placing policy to join the network. The monitoring state propagation rate from a single peer to all other peers can be described by the following equation:

$$S_{i+1} = T \cdot F \cdot \frac{S_i \cdot X_i}{n} \tag{2.1}$$

where $S$ is the number of susceptible processes (those which have not yet received the state update), $T$ is the updating period, $F$ is the fanout value, $X$ is the number of infected processes (have received the new state), $n$ is the number of processes and $i$ is the current time interval. Therefore, the propagation delay is reduced by decreasing the updating period at which gossiping occurs (thus increasing the frequency) or by increasing the fanout value (thus increasing the number of targeted peers).

One of the first P2P-based monitoring tools was proposed by Kutare et al. [102] who introduce Monalytics. Monalytics is a flexible monitoring framework attempting to move online data analysis in the monitoring layer to reduce delays and computation costs of obtaining real-time analytic insights. What makes Monalytics interesting is that the authors introduce the concept of Distributed Computation Graphs (DCGs). DCGs allow system administrators to configure their virtual in-

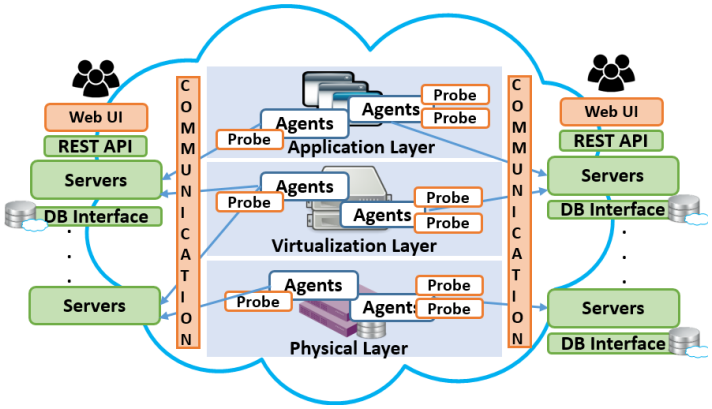



*Figure 2.8: JCatascopia architecture overview – Reprinted from [Trihinas et al., IEEE TCC, 2016] [22]*

stances in any topology that suits their needs (e.g., centralized, star, hierarchical, P2P) with an overlay gossip network formed only by monitoring servers (named brokers) which run user-defined functions (e.g., aggregation, filter functions) to-be-applied on collected monitoring data from the underlying VMs. While an interesting approach, dedicated monitoring infrastructure is required per application and elastic scaling is not supported. Another P2P-based approach is Varanus introduced by Ward et al. [103]. Varanus is a monitoring system which leverages a multi-tier P2P architecture to achieve *in situ* monitoring, thus providing a system capable of not only monitoring the underlying services but also the monitoring system itself. Varanus supports the use of monitoring layers, as depicted in Figure 2.7, to enable the use of different updating periods per layer. This achieves low propagation rates in groups of VMs with a high correlation among them but at the same time lower updating frequencies can be used for gossip among instances of different groups and cloud availability zones (denoted as regions) to not saturate bandwidth. Nonetheless, what makes Varanus interesting is that it enables rapid elasticity and autonomicity by (de-)provisioning monitoring peers based on resource utilization to load balance the computational cost of imposed monitoring queries. However, as shown in [22] and [104] solely considering peer resource utilization for elasticity control is restrictive and is not the only bottleneck. Specifically, in large-scale distributed monitoring topologies important contributing limiting factors include the communication overhead imposed by constant monitoring data dissemination, status and health message exchanging and agent coordination, as well as, the processing and archiving rate of the monitoring servers and database backend respectively.

Finally, JCatascopia [22] is an open-source, multi-layer and portable cloud mon-

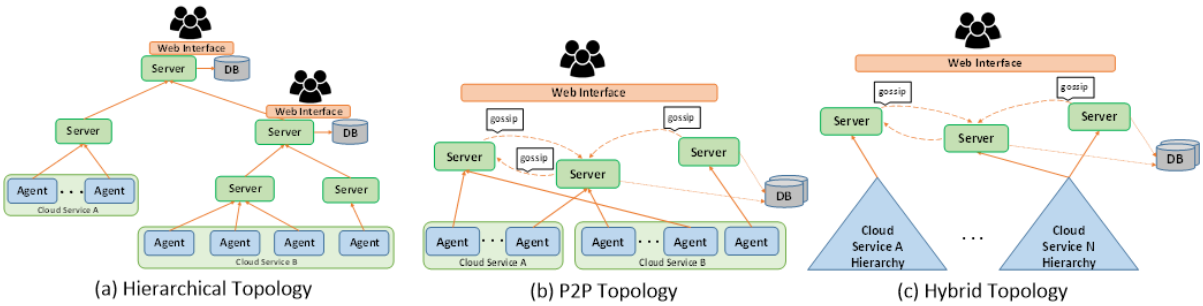



*Figure 2.9: Monitoring topology configurations – Reprinted from [Trihinas et al., IEEE TCC, 2016] [22]*

itoring framework. JCatascopia runs in a non-intrusive and transparent manner to any underlying cloud as neither the metric collection process nor metric distribution and storage are dependent on the underlying platform. As depicted in Figure 2.8, JCatascopia is a modular framework comprised of multiple components where monitoring stakeholders (e.g., cloud consumers, monitoring providers) are free to configure the overlay communication network interconnecting monitoring agents and servers to focus on *scalability*, *locality* or *logic of separation*. Thus, in contrast to Varanus, the overlay monitoring topology is transparent to the underlying monitoring agents, and consequently to user VMs. This means that no reconfiguration is required when the topology adapts or when substituted, even at runtime, with a different configuration. In turn, similar to Monalytics, JCatascopia monitoring servers can be distributed across the network to form a P2P gossiping network, as depicted in Figure 2.9. However, in this arrangement, while all monitoring servers (peers) have the same monitoring responsibilities (e.g. receive, process, store metrics) some peers also act as *seeds* with a seed being responsible for bootstrapping new peers joining the network and monitoring the health of the monitoring servers assigned to it, in par with P2P distributed database communication protocols [105].

In addition to allowing developers configure the monitoring topology, JCatascopia supports rapid elastic deployments by featuring a novel *agent bootstrapping process* based on a variation of the pub/sub communication protocol to dynamically detect when monitoring instances have been (de-)provisioned [34]. This process diminishes the need for re-contextualization when providing elasticity support, by continuously reflecting the current topology and resource configuration. In contrast to Lattice [89] and the proposed Nagios extensions [99] [100], it supports elasticity without requiring any special entities deployed on physical nodes, nor does it re-

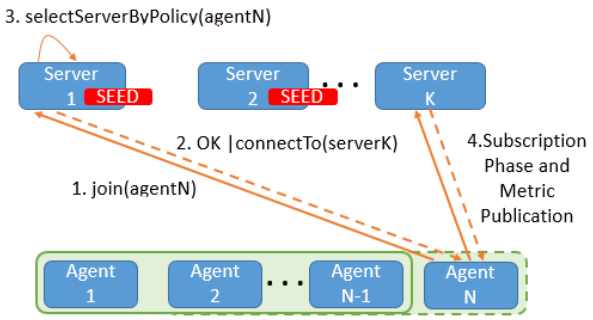



*Figure 2.10: Monitoring agent bootstrapping and placement – Reprinted from [Trihinas et al., IEEE TCC, 2016] [22]*

quest information from the hypervisor regarding the VMs currently running in an application deployment. Specifically, in this setting there are two deviations from the classic pub/sub: (i) monitoring servers bind to a network location as the less dynamic components of the monitoring infrastructure; and (ii) it is the publishers (monitoring agents) which initiate the subscription process. As with any pub/sub implementation, after initialization metrics are published from monitoring agents to the metric stream. Thus, in contrast to other monitoring tools following a classic pub/sub communication model (e.g., Panoptes [101]), the significant overhead in highly adaptive deployments where provisioned monitoring agents must first contact a central pub/sub broker to receive the location of all $N$ monitoring servers, notify them of their existence and then await $M$ monitoring servers to express interest in receiving monitoring stream updates, is reduced to $M$ direct messages[3].

To determine $M$ candidate monitoring servers for connection, JCatascopia provides developers with the ability to either select or implement their own *monitoring agent placement policy*. Specifically, when a new monitoring agent attempts to establish a connection to the monitoring network, the placement policy is used to determine candidate monitoring server(s). Such placement policies may include: (i) assigning monitoring agents to servers based on *fairness*, thus following a round-robin distribution; (ii) based on *locality*, where monitoring agents are assigned to the closest, in terms of latency, monitoring server(s) or in a multi-cloud scenario, to monitoring servers in the same availability zone; and (iii) based on monitoring server utilization, where agents are load-balanced among monitoring servers.

[3] Usually $M << N$ with $M = 1$ if no redundancy mechanism is followed

**Chapter Summary.** This chapter provides a comprehensive overview of the Internet of Things with focus given in highlighting the challenges introduced when augmenting IoT with the power of the cloud to process the wealth of data produced by IoT devices. With the growing volume of data generated at the logical extremes of IoT networks, data dissemination is becoming a bottleneck constraining the cloud computing paradigm with IoT services also exhibiting pricing charges for incoming/outgoing traffic to/from the cloud. To overcome these challenges it seems only inevitable that Edge Computing is adopted, where data is processed in place or at the proximity of the IoT device to achieve shorter response times and significantly reduce network pressure. Towards this, new advancements in monitoring tools can, potentially, bring data computation closer to the actual metric producers. However, Edge Computing introduces new challenges as IoT devices feature limited processing capabilities while intense processing results in increased energy consumption and less battery life. Therefore, while these monitoring tools seem a better fit for the unique settings of IoT networks with monitoring sources scattered across the edge, certain questions still remain unanswered. In particular, how do we deal with the high velocity of data generated from remote data sources in an attempt to offload cloud services from being constantly overwhelmed with IoT data and are struggling to be effective? and, how do we deal with the energy deficit introduced to remote data sources when monitoring the physical world due to intense metric sensing and dissemination?

# Chapter 3

# Problem Statement

This Chapter introduces and formally defines the problem domains researched in the context of the PhD thesis.

## 3.1 Preliminaries

We define a *monitoring stream*[1] $M = \{d_i\}_{i=0}^{n}$ referring to a metric of interest and published by a *monitoring source* (e.g., IoT device) to a *receiving entity* (e.g., base station) as a large stochastic sequence of independent and identically distributed (i.i.d) datapoints, denoted as $d_i$, where $i = 0, 1, ..., n$ and $n \to \infty$. A monitoring source may publish numerous metric streams and for multiple recipients, albeit each stream is limited to a single metric of interest. For instance, an IoT device reporting weather conditions may publish the following metric streams: temperature, humidity and air pollution, with readings reported to local government branches and (open) data repositories [106] [107]. Thus, the number of metric streams published by a monitoring source is not bounded. Monitoring sources and receiving entities may be co-located, either physically or virtually, although in an edge realm the norm is for monitoring sources to be remote and distant from monitoring data recipients.

Each datapoint $d_i$ is a tuple $(m_{id},\ t_i,\ v_i)$ described, at the minimum, by a unique identifier $m_{id}$ denoting the monitoring source origin, a timestamp $t_i$ and a value $v_i$. Although, a datapoint can potentially have multiple value dimensions (e.g., GPS coordinates are characterized by latitude and longitude), we will limit the scope of

[1] The terms monitoring stream and metric stream are interchangeable terms used throughout the thesis

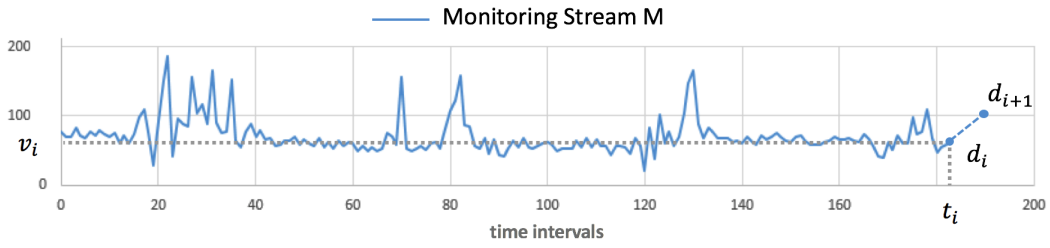



*Figure 3.1: Exemplary metric stream*

the problem to *univariate metric classes*[2], where $v_i \mapsto \mathbb{R}$. In turn, A datapoint may include a set of other attributes (e.g., area of coverage) where $\mathcal{A} = \{attr_1, ..., attr_z\}$ and $|\mathcal{A}| = z$. Although for brevity, and unless otherwise stated, when describing a datapoint we will omit these attributes without loss of generality. Furthermore, datapoint values are numeric, meaning, the type of the data must be quantitative and arithmetic operations can be used to compute the degree of difference between metric values. This also holds in the case of nominal datapoint values (e.g., blood type, current timezone, machine IP address) where a mapping to a numerical plane is both meaningful (e.g., frequency of an IP address) and provided. Finally, the number of generated datapoints depend solely on the task assigned to the monitoring source. Therefore, receiving entities have no control on the input rate, with datapoint dissemination scheduled by monitoring sources based on some *push-based* metric delivery protocol (e.g., pub/sub) [34] [101].

## 3.2 Defining the Terms "Low-Cost", "Approximate" and "Adaptive"

With "approximate" we will refer to techniques basing their decision mechanisms on estimation models capturing and predicting the runtime evolution of a monitoring stream within certain accuracy guarantees. In turn, with "adaptive" we will refer to techniques capable of adapting the properties of a monitoring source based on the predicaments of the utilized estimation model(s). Specifically, we will focus on adaptive techniques capable of dynamically changing the monitoring intensity and the rate at which metrics are disseminated through the network based on the evolution and variability of the load imposed to the referenced monitoring source.

[2]Although the domain of a metric ($v_i \mapsto \mathbb{R}, \mathbb{R}^n, \mathbb{R}^{n \times m}$, etc.) does not alter the problem formulation, certain implementation decisions are tightly coupled to the metric dimensionality (see Section 7.2)

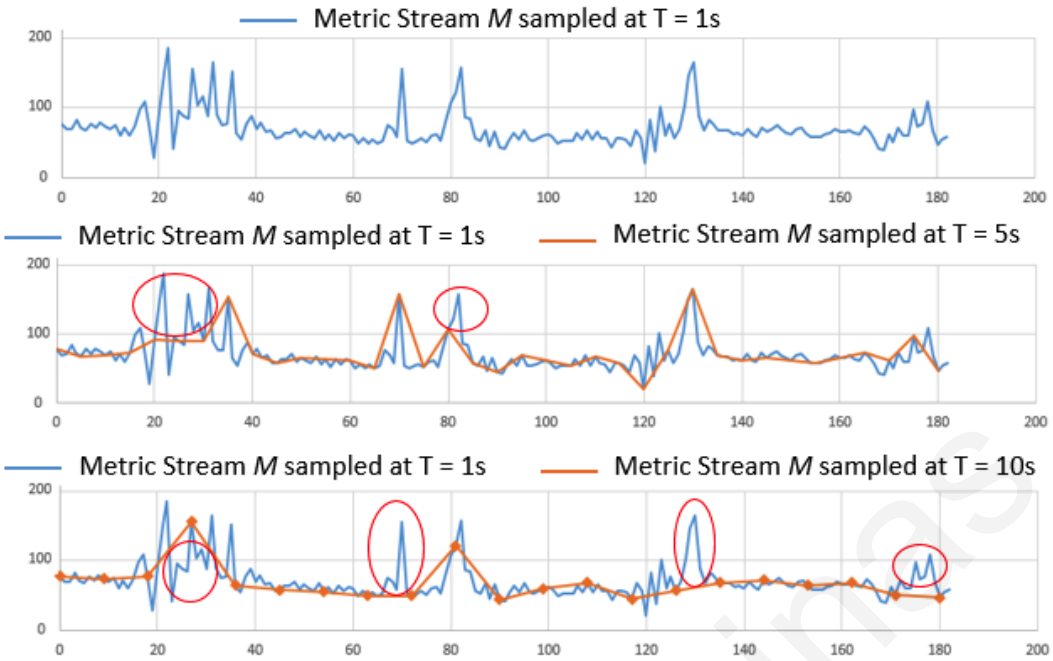



*Figure 3.2: Metric stream obtained from a wearable device counting steps*

Emphasis is given on introducing techniques capable of providing *in-place* and *inexpensive* adaptive monitoring. This means that the "adaptiveness" is handled on the monitoring source itself in order to offload processing and continuous communication between edge devices and services consuming monitoring streams. In turn, the costs, in regards to *resource utilization* and *time*, of applying adaptive monitoring techniques are (much) less than leaving the monitoring process as is.

## 3.3 Adaptive Sampling

For a monitoring stream $M$, *periodic sampling* is the process of triggering the metric collection mechanisms of a monitoring source every $T$ time units. Thus, for a metric stream indexed by $I \subseteq Z^+$, with $T$ denoting a fixed time interval (e.g., 1s, 10s, 1min), the $i^{th}$ datapoint ($i \in I$) is collected at time $t_i = i \cdot T$. Due to its simplicity this process is widely adopted by many monitoring tools (e.g., Ganglia [28], Zabbix [29]). Although a number of monitoring tools allow users to configure the metric collection periodicity, this process is performed before initialization and cannot be altered at runtime [28] [29] [89]. For example, the open-source and popular Ganglia monitoring tool [28], allows users to change the `collect_every` parameter located in the Ganglia configuration file before initialization. However, after initialization to alter metric collection, the monitoring process must stop and start again, if it is to acknowledge

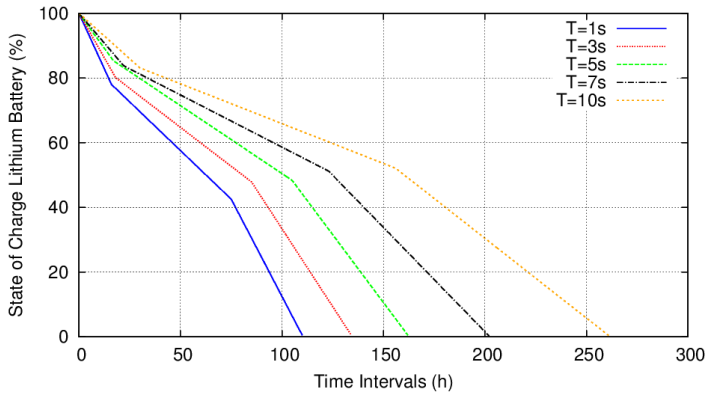



*Figure 3.3: State-of-Charge (SoC) projection for a wearable with a lithium battery*

further changes to the periodicity of a metric stream.

In this thesis we argue that using a fixed periodicity on monitoring sources, especially when battery-powered and with limited processing capabilities, features a number of constraints. For example, consider the monitoring stream extracted from a wearable device introduced in Figure 3.2. If a small $T$ is utilized (e.g. $T = 1s$), a high volume of data is generated and must be disseminated through the network to be processed or stored for further use although there exists phases of low datapoint value fluctuation in the monitoring stream. On the other hand, if a larger periodicity is used (e.g. $T = 5s$, $T = 10s$), then sudden events or significant insights may remain undetected. Most importantly, if the monitoring task must collect external stimulus and then perform costly analysis, such as in the case of human activity monitoring from wearable devices, then significant energy consumption is required to collect a single datapoint of a monitoring stream. This is particularly evident in Figure 3.3 which depicts an estimation of the *state of charge* for a lithium battery[3] found on a wearable device (35mAh capacity), powering the monitoring source publishing the previous metric stream for various sampling periods. Nonetheless, even in the case where data sensing is not an intense process for the monitoring source, if a high collection rate is used then the volume of data generated introduces a new challenge as services consuming monitoring streams from multiple and numerous edge sources can be quickly overwhelmed, thus restricting further service scalability. Therefore, from Figures 3.2 and 3.3 we conclude that it is both resource and energy consuming, for both monitoring sources at the edge and monitoring data recipients, to periodically collect datapoints, especially when consecutive metric values (e.g.,

[3] State-of-Charge (SoC) is a linear algorithmic process used to model the energy capacity remaining in a lithium battery [75]

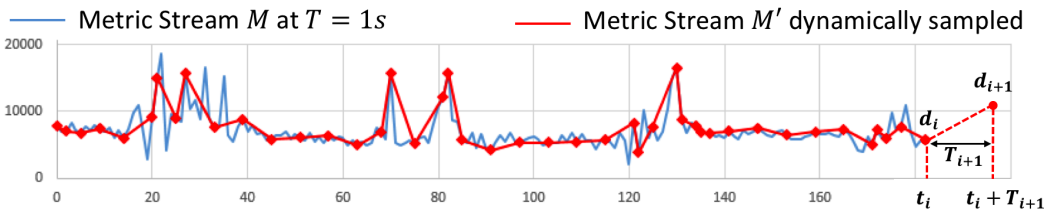



*Figure 3.4: Adaptive sampling vs periodic sampling when applied to a metric stream*

$v_{i-2}$, $v_{i-1}$, $v_i$, ...) do not vary. In general, because sampling depends on the data and its evolution in time, we argue that using a fixed sampling period is not effective, as metrics and insights are only useful if collected in meaningful time intervals.

To accommodate the challenges introduced from periodic sampling, *adaptive sampling* is used. Adaptive sampling is the process of applying approximation techniques to dynamically adapt the sampling period $T_i$ of a monitoring source (e.g., IoT device), based on some estimation model, denoted as $\rho(M)$, capturing runtime information of the metric stream evolution. Hence, when the datapoint values of the monitoring stream are relatively stable, and therefore the monitoring stream can be approximated by the estimation model, the sampling period should be increased to reduce the volume of data generated and the energy consumed by the monitoring source for datapoint collection. In turn, when the values fluctuate and the estimation model cannot approximate the monitoring stream within certain user-defined accuracy guarantees, the sampling period should be decreased or restored to a (default) minimum value in order to preserve accuracy and immediately report any unexpected and sudden events. In any case, how large of an adjustment is required, is dependent to some evaluation metric, denoting the ability, or better the confidence, of the approximate technique to (correctly) estimate and follow the metric stream current evolution within the certain accuracy guarantees given by the user. Hence, if the estimation model $\rho(\cdot)$ is able to approximate the metric stream within the given accuracy guarantees then an increase in the sampling period can be considered, whereas if distribution shifts are introduced in the metric stream value base, and are unexpected, a decrease in the sampling period should be considered in order to obey the given guarantees.

Therefore, assume $d_i$ to be the latest datapoint of $M$, and that $T_i$ accepts discrete integer values in the range $[T_{min}, T_{max}] \subseteq \mathbb{Z}^+$ without loss of generality. Now, suppose $M$ is periodically sampled every $T_{min}$ time units, opposed to $M'$ which is a reconstructed version of the original metric stream via adaptive sampling, as depicted in

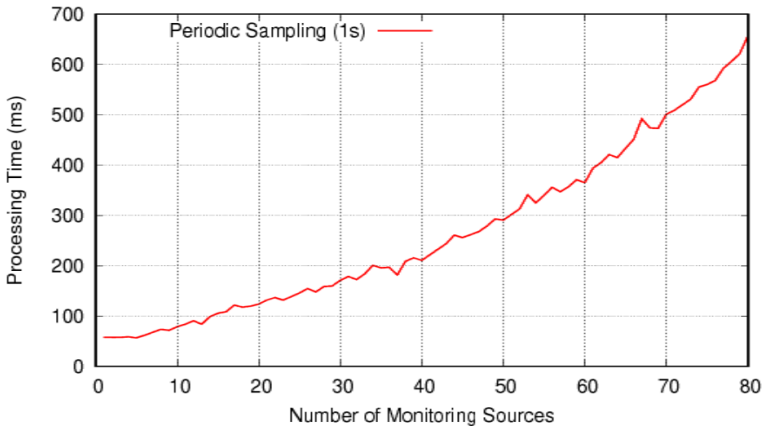



*Figure 3.5: Monitoring service scalability with monitoring sources periodically collecting and disseminating metric updates*

Figure 3.4. Let *err* denote the difference of $M'$ from $M$ based on some evaluation metric which will be used to evaluate the accuracy of the estimation process. Hence, the goal of adaptive sampling is to provide a *sampling function* $f(\cdot)$, capable of finding the maximum $T \in [T_{min}, T_{max}]$ to collect $d_{i+1}$, based on an estimation of the metric stream evolution $\rho(M)$, such that the difference between $M'$ and $M$ is upper bounded by the user-defined maximum tolerable imprecision $\gamma$, for the range $t \in [t_i, t_i + T]$. Thus, the problem is summarized with the following equation:

$$T^* = \arg\max_{T} \{f(d, T, \rho(M), err(M, M')) \mid err < \gamma, T \in [T_{min}, T_{max}]\} \tag{3.1}$$

Intuitively, as $\gamma \to 0$ the metric stream $M' \to M$, but, the sampling period $T \to T_{min}$, defeating the purpose of adaptive sampling. To reduce data volume and preserve energy, an adaptive technique is likely to select, at any given time, a sampling period where $T > T_{min}$, which is only applicable if a degree of imprecision is tolerable. Therefore, *it is desirable to select a sampling function which achieves a balance between efficiency and accuracy*.

## 3.4 Adaptive Dissemination

Monitoring stream dissemination is another fundamental process of monitoring services commonly implemented by *periodically disseminating* collected datapoints, such that for a fixed period of time $D$, the $i^{th}$ datapoint of a metric stream indexed by $I \subseteq Z^+$, is reported to interested receiving entities at $t_i = i \cdot D$. Based on this, all collected datapoints are reported to respected receiving entities and therefore,

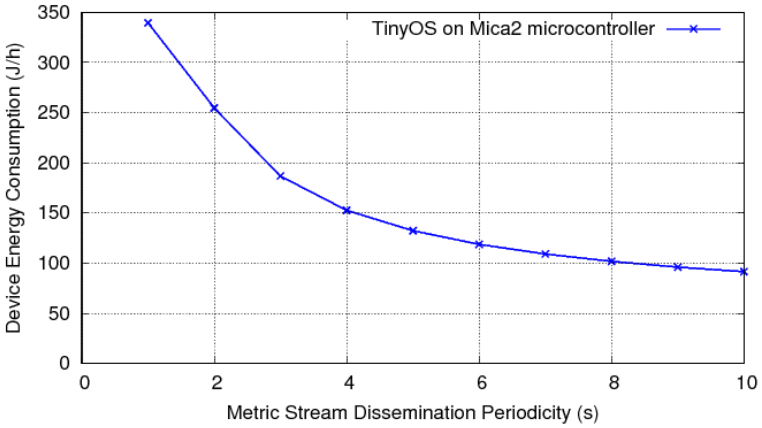



*Figure 3.6: IoT device energy consumption for metric stream dissemination*

under expected circumstances, no data loss will occur for datapoint dissemination. Due to its simplicity, this approach is widely adopted by monitoring systems (e.g., Nagios [27]) although it is also common for monitoring systems to follow certain aggregation policies (e.g., Ganglia [28]) where dissemination is triggered only when a window $W$ of $k$ datapoints $|W_j| = K$, indexed by $J \subseteq Z^+$ and $\{j \mid j = i/K \ \wedge \ j \in J\}$, is collected. However, for mobile and battery-powered sensing devices, metric stream dissemination to (distant) consuming services is another primary energy drain [9].

Let us now assume that one receiving entity exists and that it is interested in receiving a metric stream, such as the one depicted in Figure 3.2. The energy consumed by the network unit of a monitoring source residing on a micro-controller running TinyOS when disseminating the aforementioned metric stream is shown in Figure 3.6. From this figure we observe that the energy consumed is significantly reduced if the dissemination period $D$, is lowered. In particular, the amount of energy to send a message of $x$ bytes is given by $\mu_s + \beta_s x + \epsilon_s$, where $\mu_s$ and $\beta_s$ represent the *per-message* and *per-byte* dissemination costs, whereas $\epsilon_s$ is the energy consumed for *state transitions* and therefore, the energy required for the IoT device network unit to transition from idle to active and then to a transmitting state [108].

Based on the above, reducing either the number or size of messages sent through the network is required to optimize energy-efficiency [36]. Nonetheless, $\beta_s$ is typically at least two orders of magnitude smaller[4] than $\mu_s$. Therefore, assuming, as mentioned in Section 3.1, that a datapoint is described in a minimal form (e.g., a timestamp and a value) or that compression is available by the monitoring tool when an aggregation policy is applied, it is evident that to reduce on device energy

[4] Typical values for a Mica2 microcontroller are $\beta_s$ = .0144 mJ/byte, $\mu_s$ = .645 mJ and $\epsilon_s$ = .331 mJ

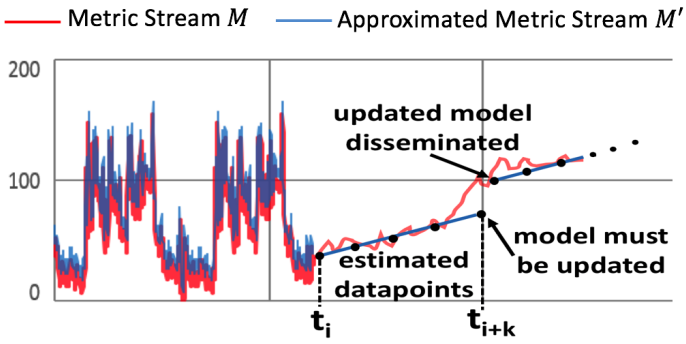



*Figure 3.7: Adaptive dissemination applied on a metric stream*

consumption and ease processing at the receiver-side, the velocity of datapoints disseminated across the network must be reduced. Reducing the velocity of disseminated datapoints also offloads computationally interested recipients consuming monitoring streams from remote monitoring sources as mentioned in the previous section and depicted in Figure 3.5. However, as with periodic sampling, reducing the dissemination rate risks delaying interested receiver notification of sudden events and significant insights. Hence, in this thesis, we argue that periodically disseminating metric streams features energy and resource constraints, especially when consecutive datapoints (or window of datapoints) do not vary. In general, because dissemination depends on the data and its evolution in time, we argue that using a fixed dissemination rate is not effective, as metrics and insights are only useful if disseminated in meaningful time intervals.

To accommodate the challenges introduced from periodic dissemination, *adaptive dissemination* is used. Adaptive dissemination is the process of applying approximation techniques to sensed datapoints in order to reduce the communication overhead imposed directly to a monitoring source (e.g., energy consumed by IoT device) and indirectly to a network of monitoring sources (e.g., IoT network data overwhelm), by suppressing the dissemination of consecutive datapoints with "little" change in their metric values. How much "change" is considered as "little" depends on the approximation technique and the certain accuracy guarantees given by the user. In particular, we will consider that accuracy guarantees are given in the form of confidence guarantees $\delta \in [0, 1]$, denoting the probability with which estimated datapoints are approximated from sensed datapoints.

To reduce to the communication overhead, each monitoring source must maintain a runtime estimation model, denoted as $\rho(M)$, capturing knowledge of the monitor-

ing stream evolution. This model is then disseminated to interested receivers. Let the model be reported at the $i^{th}$ time interval. From this point, the receiver operates under the assumption that sensed datapoints can be approximated within the given confidence by a forecasting function of the estimation model, denoted as $f(\cdot)$. Therefore, subsequent $k$-datapoints reported at $t_k = t_i + k \cdot T \mid k \subseteq \mathbb{Z}^+ \ \wedge \ k > i$, are inferred from the model with $d_{i+k|i} = f(\rho(M), d_i)$. At the same time, the monitoring source withholds datapoint dissemination, interacting with the receiver only when a decision function, denoted as $g(\cdot)$, detects shifts in the metric stream that render the model no longer able to describe the metric stream evolution within the given user-defined accuracy guarantees. At this point the monitoring source must disseminate to the receiver an updated parameterization of the estimation model.

Based on this, let $M'$ depict the reconstructed version of $M$ from $f(\cdot)$ at the receiver side at a given $k$ time interval, *dist* to denote the difference of $M'$ from $M$ based on some distance metric, and $\eta(\delta)$ to denote the accuracy budget given by the user in the form of confidence guarantees. Assuming dissemination for a model update was triggered at the $i^{th}$ time interval then the problem of detecting a shift in the monitoring stream in a subsequent $t_k$, is summarized with the following equation:

$$g(M, M', t) = \begin{cases} \textit{trigger model update}, & \textit{dist} > \eta(\delta) \\ \textit{suppress dissemination}, & \textit{otherwise} \end{cases} \tag{3.2}$$

Hence, the goal of adaptive dissemination is twofold: (i) provide an estimation model alongside a forecasting function capable of approximating the runtime evolution of the monitoring stream; and (ii) provide a decision function capable of detecting when shifts in the monitoring stream render the current model parameterization, shared among monitoring source and receivers, no longer able to describe the monitoring stream within the given accuracy guarantees.

## 3.5 Adaptive Filtering

A specialized case of adaptive dissemination is *filtering*. In particular, filtering is the process of suppressing datapoint dissemination when consecutive datapoint values differ less than a range of values. Hence, energy required by a monitoring source to transmit metrics over the network, as well as, the velocity at which data arrive to services consuming monitoring streams are reduced while adhering to certain user-

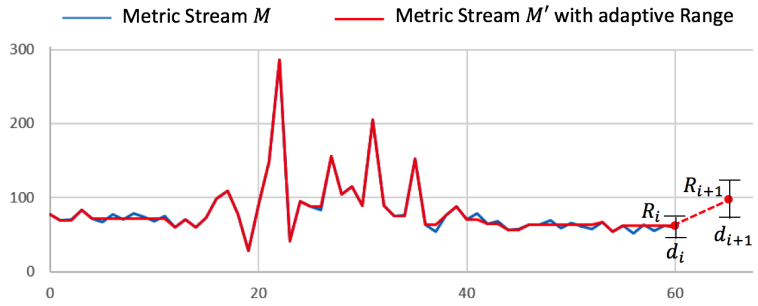



*Figure 3.8: Adaptive filtering $R \in [0, 3]$ compared to fixed range filtering $R = 1$*

defined accuracy guarantees. Thus, a monitoring source with filtering capabilities does not transmit the latest collected datapoint if its value has not "changed" since last reported. As in adaptive dissemination, how much "change" is considered as "little" depends on the decision function used by the filter and the certain accuracy guarantees given by the user or as a configuration parameter to the monitored application. Nonetheless, while filtering is used to suppress metric stream dissemination, forecasting is not used to provide an approximate version of the original metric stream. Hence, the metric stream recipient, assumes that the values of any unreported datapoints remain unchanged.

However, depending on the type of *filter* in use, the number of datapoints filtered may vary. For example, suppose a *fixed filter range* approach is followed. The datapoint $d_i$ with value $v_i$ is filtered, if $v_i \in [v_{i-1} - R,\ v_{i-1} + R]$, where $R \subseteq \mathbb{R}^+$ is a fixed filter range. Although this approach is simple and followed by monitoring tools, it features a number of disadvantages. Specifically, using a fixed filter range, assumes that the user has previous knowledge of the metric evolution and that it will not change in the future. Otherwise, there is no guarantee that any values will be filtered at all [109]. For instance, let us consider the metric stream $M$, presented in Figure 3.8, where the filter range $R$ is enabled once and set to a fixed value. From Figure 3.8, we observe that a phase of high variability is both preceded and followed by a phase of low variability, where metric values $v_i$ oscillate between $[v_{i-1} - R - \epsilon,\ v_{i-1} + R + \epsilon]$ with $\epsilon \to 0$. With a fixed filter in use, no datapoints are filtered. That is because the filter cannot adapt to the current data variability, extending its range to encapsulate *near-by* values, thus satisfying the reduction guarantees and, at the same time, adhering to the accuracy requirements of the user.

To overcome the above issues, an *adaptive filter* technique is used. Adaptive filtering is the process of dynamically adjusting the filter range $R$ based on the

current variability of the metric stream, denoted as $q(M)$. Adaptive filtering must target filtering values without requiring for users to "guess" what filter range should be used, as depicted in Figure 3.8. Thus, suppose $M'$ is a reconstructed version of $M$ with an adaptive filter range $R \in (0, R_{max}]$. Let *err* denote the difference of $M'$ from $M$ based on some error evaluation metric. After collecting $d_i$, the goal of adaptive filtering, is to provide a *filtering function* $f(\cdot)$ capable of finding the maximum $R$ to apply on $d_{i+1}$ based on the variability of the metric stream $q(M)$, such that the difference between $M'$ and $M$ is upper-bounded by the user-defined maximum tolerable imprecision $\gamma$. Hence, the problem is summarized with the following equation:

$$R^* = \arg\max_R \{f(d, R, q(M), err(M, M')) \mid err < \gamma, R \in (0, R_{max}]\} \tag{3.3}$$

Intuitively, as $\gamma \to 0$ the timeseries $M' \to M$, but then the filter range $R \to 0$, defeating even the purpose of filtering with a fixed range $R$. To reduce network traffic an adaptive filtering technique is likely to select, at any given time, a filter range where $R > 0$, which is only applicable if a degree of imprecision is tolerable.

**Chapter Summary.** This chapter presents the problem domains investigated in the scope of the thesis by highlighting the limitations of following a fixed periodicity approach for both metric collection and dissemination which are considered the prime energy drains for mobile edge devices. To tackle these limitations we introduce the concept of low-cost approximate and adaptive monitoring and provide a formal definition for three such techniques. *Adaptive Sampling* dynamically adjusts the intensity of the metric collection process depending on the evolution and variability of the monitoring data stream to reduce the overhead imposed to a monitoring source. *Adaptive Filtering* discards consecutive metric values that do not differ more than a range of values that is adjusted to given accuracy guarantees in order to reduce the volume of data disseminated over the network. In turn, rather than transmitting metric values, *Model-Based Dissemination* favors modeling the data stream so that consuming entities infer metric values from the model, triggering a model update only when shifts in the stream evolution render the model as inaccurate.

# Chapter 4

# Related Work

Other than distributed system and network monitoring [11] [110] [111], "adaptiveness" finds wide applicability in various application domains such as signal processing [10] [112], wireless sensor networks [108] [113], biosignal reporting [26] [114] and differential privacy preserving [40] [41]. Embracing adaptive monitoring to either offload processing via data reduction or dynamically adjust the monitoring intensity to optimize energy efficiency; is a form of edge mining albeit not limited to the Internet of Things. This is acknowledged in the literature overview that follows by including any adaptive monitoring techniques that may fit the requirement settings of IoT networks. In turn, although a number of hardware-specific techniques are still being proposed, especially for signal processing [115] [116] [117], trends show that IoT is moving towards software-defined realms to quickly provision, manage, monitor and orchestrate IoT services with "adaptivity" being envisioned as a manageable asset for monitoring services [8] [20]. Hence, in what follows, a number of software-defined techniques for adaptive monitoring are presented.

## 4.1 Adaptive Sampling Techniques

One of the first software-defined sampling techniques for remote monitoring sources was introduced by Rastogi et al. [40]. In particular, the authors are motivated by the lack of privacy preserving mechanisms for monitoring user sensitive information in the form of timeseries collected by heath and activity tracking services (e.g., wearables monitoring user weight and calorie consumption). The authors propose using a Discrete Fourier Transformation (k-DFT) [118] which first perturbs coefficients of a finite timeseries adding laplacian noise [119] to each of the coefficients to ensure

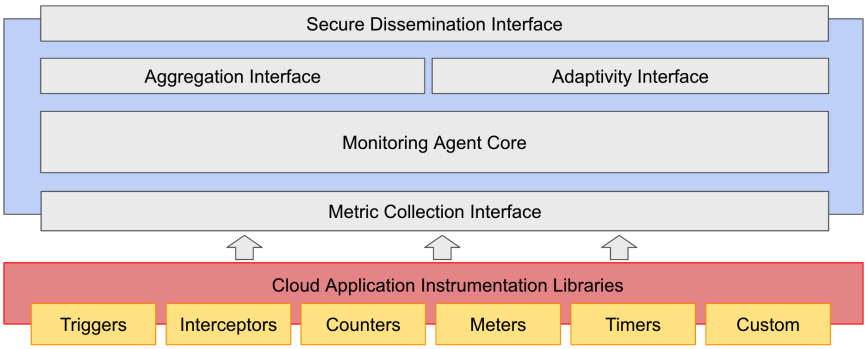



*Figure 4.1: Abstract overview of envisioned software-defined monitoring agents in the Monitoring 4.0 era*

user privacy. To reduce the intensity of this process[1] instead of transmitting the full length of the timeseries comprised of $n$-coefficients the $k$ most significant coefficients are selected ($k < n$). Trusted service providers can afterwards reconstruct the original timeseries ($k = n$) or a $k$-approximated version ($k < n$) from the inverse k-DFT. The obvious downside of this technique is that while at the receiver-side there appears to be adaptivity in the metric stream collection process, due to discarding coefficients, in reality all datapoints are still collected at the remote monitoring source.

To address these challenges Alippi et al. introduce ASA [120]. In particular, ASA performs summarizations on windows of consecutive datapoints in the form of either a cumulative sum (CUSUM) or an exponential weighted moving average (EWMA). Upon initialization ASA requires from users two parameters: $\delta$ and $h$. With $\delta$ the authors denote the difference of the current window summary from the previous summary. If the difference is more (less) than $\delta$, then an decrement (increment) in the sampling period must be considered. However, a change is triggered only if $h$ consecutive violations are detected to reduce, as claimed by the authors, the false positives that induce a continuous change in the sampling period (Algorithm 1). Nonetheless, as with Rastogi et al. [40], $n$-length datapoint windows still must be collected before applying this technique, thus limiting real-time applicability.

Similar to ASA is Payless [110], a network monitoring tool for Software Defined Networks (SDNs) introduced by Chowdhury et al. For Payless, the polling rate of the monitoring server is increased by a small constant, denoted as $\alpha$, if the difference between the current datapoint is larger than a threshold $\Delta_1$. Otherwise, if the

[1] Naive FFT features an $O(n^2)$ complexity while various advance DFT approaches reduce the complexity up to $O(n \log n)$

**Algorithm 1** ASA Adaptive Sampling – Adapted from [Alippi et al., IEEE Trans. on Instrumentation and Measurement, 2010] [120]

**Input:** **W** window of datapoints, $\delta$ difference to detect, $h$ number of consecutive (non-) differences before adapting sampling period

**Output:** $T$ sampling period

```
1: s_cur ← doSummarization(W)  // e.g., CUSUM, EWMA
2: if |s_cur − s_prev| > δ then
3:     h_1 ← h_1 + 1, h_2 ← 0
4:     if h_1 ≥ h then
5:         T ← (1 − δ) · T
6:     end if
7: else
8:     h_2 ← h_2 + 1, h_1 ← 0
9:     if h_2 ≥ h then
10:        T ← (1 + δ) · T
11:    end if
12: end if
13: s_prev ← s_cur
14: return T
```

difference is less than $\Delta_2$ then the polling rate is divided by another constant, denoted as $\beta$. In turn, Andreolini et al. [111] introduce an adaptive sampling technique similar to Payless, but instead of comparing the current datapoint value to user-defined thresholds, the current variability (standard deviation) is used as a more suitable mechanism to discriminate stable from variable states. While ASA, Payless and [111] are interesting approaches, they require a number of parameters and user-defined policies to be set once upon initialization and cannot change at runtime, thus assuming that the metric stream datapoint value distribution will always remain relevant.

A different approach is followed by Meng et al. [11], who propose an adaptive sampling framework for cloud networks. Specifically, the authors suggest that an adaptive sampling framework should follow a violation-likelihood detection approach. Thus, the sampling rate is increased when metric values approach a user-defined threshold denoted as $\theta$, and restored to a fixed interval when a violation is

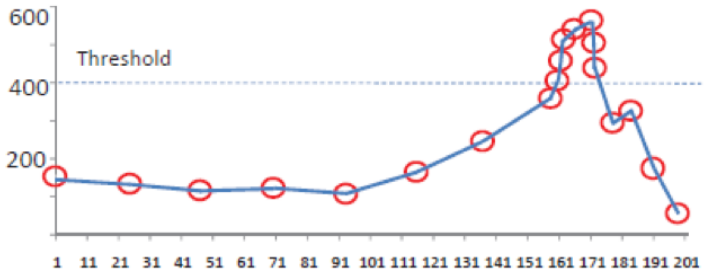



*Figure 4.2: Adaptive Sampling Based on Threshold Violation Likelihood – Reprinted from [Meng et al., IEEE Transactions on Computers, 2013] [11]*

unlikely to occur (e.g., $u_i << \theta$) based on the probability of mis-detecting a violation during the gap between two consecutive datapoint values, as depicted in Figure 4.2. While an interesting perspective, this framework is a server-side solution not applicable to the lightweight monitoring sources comprising IoT networks. In turn, as a threshold-based technique, it fails to detect variances in the metric evolution for values far from the threshold. Thus, it is limited to adaptive sampling tailored solely for anomaly detection.

A more intuitive approach is proposed by Fan et al. [121]. In particular, the authors introduce FAST, an adaptive sampling framework which uses a PID controller [122] as part of a control loop enabling user differential privacy (Figure 4.3). Specifically, FAST computes an estimate of what sampling period should be used to collect the next datapoint based on the adjustment given by the PID controller which is fed by the current estimation error and the time intervals between previously collected datapoints in an attempt to balance a given inaccuracy budget (Algorithm 2). Thus, if the metric stream values are going through rapid changes the estimation error will increase. In response, the FAST PID controller will detect this error and increase the sampling rate accordingly.

However, FAST's adaptive sampling technique is aggressive producing large sampling periods as the motivation behind its development is applying a costly user-differential policy on each sampled interval [41]. Thus, it requires a Kalman filter [123] which generates estimates for non-sampled datapoints to reduce the estimation error. However, as the Kalman filter is used to filter signal noise, this approach does not work well for abrupt transient signals where large portions of the signal are lost due to smoothing. In addition, even for slightly less volatile signals extensive profiling of its parameters, along with the PID controller parameters, are

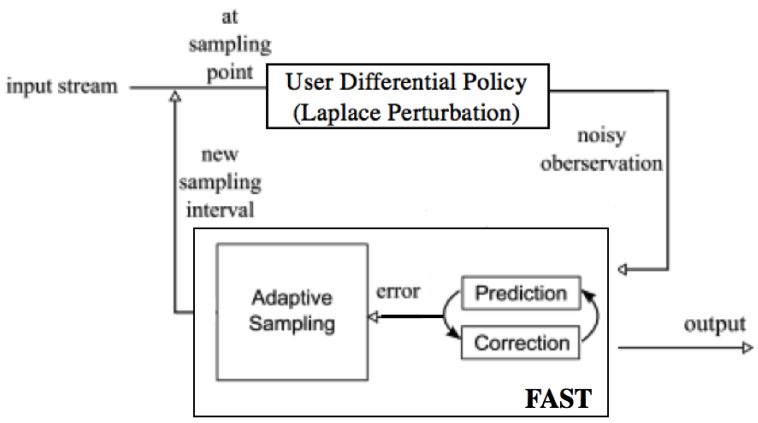



*Figure 4.3: FAST Adaptive Sampling PID Controller – Adapted from [Fan et al., IEEE TKDE, 2014] [41]*

still required to increase accuracy. Most importantly, while energy is preserved from not constantly performing costly sensing (e.g., heartrate monitoring), as non-sampled datapoints are still estimated to increase accuracy, data volume reduction is minimum.

Jain et al. [124] introduce a technique capable of adapting the sampling period for nodes comprising a Wireless Sensor Network (WSN). The period to collect the next datapoint is a two-step process, where first, the sampling period is estimated based on a Kalman filter, much similar to FAST, and selected within a range of possible sampling periods ($T_i \in [T_{low}, T_{high}]$). However, this range of possible sampling periods is provided, by a central "synchronization" service allocating different bandwidth budgets to sensor nodes in the WSN based on the variability of their readings (e.g., sensors with high variability are allocated a larger range of sampling periods). This second step is considered an additional overhead stressing the system when the number of sensors, and consequently their readings, vastly scale. On the other hand, Tata et al. [125] introduce an optimization model for edge monitoring determining which metrics should be monitored and the frequencies at which these metrics are to be monitored. In particular, the authors introduce an iterative approach where the resource constraints of the monitoring source and the volatility of the metric stream are considered as input for a constraint-based linear optimizer (CPLEX) in which the metric frequencies are re-optimized whenever an environmental event or a monitored metric value triggers the need for such re-optimization.

In turn, Gaura et al. [10] propose a number of edge mining techniques and algorithms specifically tailored to IoT networks. With respect to adaptive sampling,

**Algorithm 2** FAST Adaptive Sampling – Adapted from [Fan et al., IEEE TKDE, 2014] [41]

**Input:** $\mathbf{Z}$ sliding vector containing recent datapoint readings, $\xi$ user-defined PID setpoint threshold and $\theta$ magnitude of change

**Output:** $T$ sampling period and $\mathbf{Y}$ latest prediction state vector

1: $\mathbf{E} \leftarrow estError(\mathbf{Z}, \mathbf{Y})$ //control loop error (e.g., mean absolute error)
2: $\Delta_{new} \leftarrow PID(\mathbf{E}, \Delta_{old})$ //proposed adjustment
3: $T \leftarrow max\{T_{min}, \lfloor T + \theta \cdot (1 - e^{\frac{\Delta_{new} - \xi}{\xi}}) \rfloor\}$
4: **for** $k = 1 \ : \ T - 1$ **do**
5: $\quad Y[k] \leftarrow applyKalman(\mathbf{Z}, \mathbf{Y})$
6: **end for**
7: $\Delta_{old} \leftarrow \Delta_{new}$
8: **return** $T$, $\mathbf{Y}$

the authors propose L-SIP, a linear algorithm, similar to ASA (Algorithm 1), but with the difference that it encodes the current state of a monitoring source as a point in time with attributes the datapoint value and its rate of change $\mathbf{y} = (y, \dot{y})^T$. This is performed by using as a state estimation function either an exponential weighted moving average (EWMA) [126] or a Kalman filter, with the sampling rate increasing if the difference between the observed and prediction value are larger than a user-defined estimation error. L-SIP is an interesting lightweight algorithm suitable for adaptive sampling on IoT devices, but exhibits two downsides: (i) it is slow to react to highly transient and abrupt fluctuations in the metric stream; and (ii) it is left to the user to determine, via profiling, which state encoding method suits best her needs.

Finally, Bailis et al. [127] introduce Adaptable Damped Reservoir (ADR) sampling for MacroBase which maintains a non-uniform sample of datapoints from ingested metric streams that is exponentially weighted towards more recent points. Thus, to promptly reflect changes in the underlying metric stream, MacroBase adopts weighted sampling, in which the probability of data retention decays over time. Specifically, the ADR maintains a running count $c_w$ of datapoints inserted into the reservoir of size $k$. When a new datapoint is collected and is considered for insertion, $c_w$ is incremented by one (or an arbitrary weight, if desired). With probability $k/c_w$, the datapoint is placed into the reservoir and a random datapoint is evicted

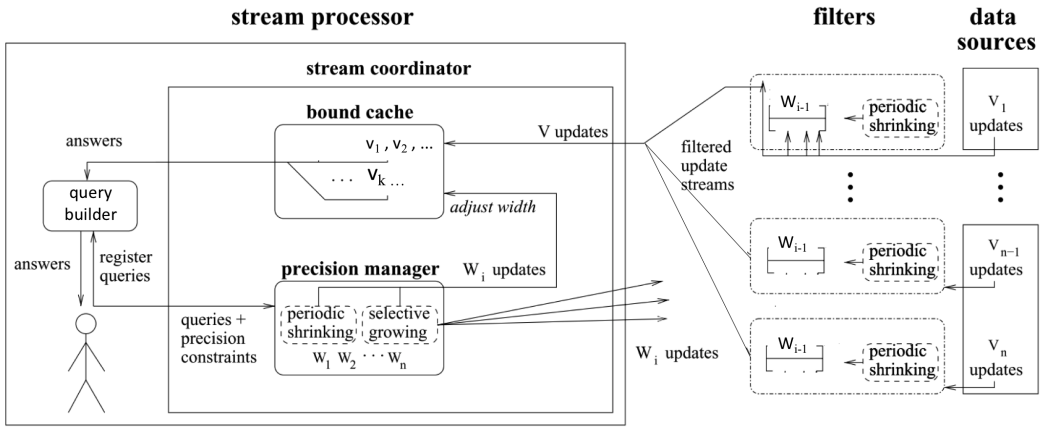



*Figure 4.4: Dynamic Filter Range Window Updating from Server-side Stream Coordinator – Reprinted from [Olston et al., SIGMOD, 2010] [130]*

from the reservoir. When the ADR is decayed via a periodic timer, which means the reservoir decays at a pre-specified rate measured according to real time, its running count is multiplied by a decay factor, $c_w := (1 - a)c_w$. While MacroBase stream consumers observe adaptivity by probabilistically selecting datapoints of significance in arbitrary windows, similar to the k-DFT approach suggested by Rastogi et al., still all datapoints are collected with the periodicity unaltered.

## 4.2 Adaptive Filtering Techniques

Clayman et al. [109] introduce an extension to the Lattice monitoring tool [89] that allows users, through an API interface, to enable fixed filtering for the metrics offered by the respected monitoring tool where the user prior to monitoring instantiation must determine the filter range and metrics that filtering will be applicable. Another notable monitoring tool supporting filtering is Dargos from Povedano-Molina et al [128]. In particular, Dargos users are provided with the tooling to enable low and high filter ranges per monitored metric with the addition of a time-to-live (TTL) parameter so as for the monitoring source to send at least one monitoring state update every TTL intervals to signal that it is still active.

A different approach is followed by Jain et al. who introduce Star [129], an hierarchical monitoring tool with filtering and aggregation capabilities. In regards to filtering, Star assumes a tree-based monitoring hierarchy and requires from users to define upon initialization an error budget $\delta_T$ that denotes the maximum inaccuracy any subtree part of the hierarchy will report to its parent node for each monitored

**Algorithm 3** JCatascopia Adaptive Filtering – Adapted from [Trihinas et al., IEEE/ACM CCGrid, 2014] [34]

**Input:** $v_i$ and $t_i$ current value and timestamp, $N$ window length to adjust filter range, $filtering\_target$ and $step$ the user-defined filter guarantee and step adjustment

**Output:** $R$ new filter range, $filteredVals$ percentage of values actually filtered

1: **if** $v_i \in [v_{i-1} - R, v_{i-1} + R]$ **then**
2: filter($v_i$)
3: $count \leftarrow count + 1$
4: **end if**
5: **if** $t_i \,\%\, N == 0$ **then**
6: $filteredVals \leftarrow count/N$
7: **if** $filteredVals < filtering\_target$ **then**
8: $R \leftarrow R + step$
9: **else**
10: $R \leftarrow R - step$
11: **end if**
12: **end if**
13: **return** $R, filteredVals$



metric. This arrangement reduces network load by filtering small updates that fall within the range of values cached by a subtree's parent. In particular, after a node $A$ with error budget $\delta_T$ reports a range for a given metric $[v_{min}, v_{max}]$ to its parent node (where $v_{max} \leq v_{min} + \delta_T$ ), if node $A$ receives an update from one of its child nodes, node $A$ can skip updating its parent node as long as it can ensure that the true value of the metric for the subtree lies between $v_{min}$ and $v_{max}$.

Olston et al. [130] introduce a server-side framework for filtering continuous data streams from remote monitoring sources. This approach involves users specifying a precision requirement with data sources sending updates to a central server node when new values differ significantly from the previously reported values. If this precision requirement cannot be met, the central server will adjust the filter range at each data source, as depicted in Figure 4.4. However, this adaptive filter adjustment is only feasible if data generated at different sources follow a certain similar pattern on all nodes. In turn, Kim et al. [131] introduce a supervised learning approach for metric filtering in IoT networks where a naive Bayesian classifier is trained to

detect and label "normal" from "eventful" activity hidden in IoT monitoring streams. In this approach "normal" data is used solely for model training and is afterwards filtered so as to offload computationally IoT management services by only providing them with updates regarding monitoring sources in "eventful" states (e.g., device malfunction). In particular, the detection function of the proposed approach is used to predict the data characteristics through the naive Bayesian classifier using stored training examples while in the case of an "event" a k-nearest neighbour process is used to weight the significance of the event.

On the other hand, in Trihinas et al. [34], we introduce an adaptive filtering algorithm enabling monitoring systems such as JCatascopia to autonomously adjust the filter range depending on the percentage of values previously filtered to achieve certain user-defined filtering guarantees. Specifically, as depicted in Algorithm 3, after the collection of a window length of $N$ metric values, the algorithm compares the percentage of values filtered to a $filtering\_target$ defined by the user, adjusting the filter range accordingly. If the target was not achieved then the filter range is increased stepwise ($R \leftarrow R + step$) and in turn if a more aggressive approach was followed, thereby violating the user-defined filtering guarantees, the filter range is decreased ($R \leftarrow R - step$). In turn, Du et al. [132] follow a similar approach to us, but also show that network traffic generated by monitoring sources can be further reduced if monitoring tools suppress metric dissemination by sending only the median from a fixed window of collected values, whenever the current metric value has not changed more than a pre-defined threshold.

Finally, Fbflow is a tool used at Facebook to monitor the network activity of the database fleet [133]. An Fbflow agent resides on each machine listening to the `nflog` socket and parses the headers to extract network traffic information. These parsed headers are then streamed to Scribe and indexed by Hive. Because of the intensiveness of the monitoring process, data is collected at a `millisecond` range, all data streams (250,000+ machines is the user profile database) cannot be digested for analysis, indexing and storage. Thus, Fbflow applies filtering on each machine, where instead of transmitting a new datapoint at $t_j$, it is modeled as a moving process ($p_j$) and the rate of change from the previous reported value at $t_i$ is monitored, $r_{i,j} = p_j/p_i$. Inferring whether $r_{i,j}$ is within an expected range allows the Fbflow agent to decide if filtering should be applied. To decide if the ratio $r_{i,j}$ is expected, Fbflow aggregates and models the ratios within the machine jurisdiction (e.g., rack, cluster,

datacenter) as a normal distribution. The probability of the machine $r_{i,j}$ is then compared with the jurisdiction distribution and a change is deemed unexpected if the probability of $r_{i,j}$ is above or below a certain threshold. The downside is that when low variability is evident in the machine jurisdiction (e.g., due to efficient load balancing) then minor changes to a machine's ratio are "falsely" signaled as unexpected due to not adapting the jurisdiction model sensitivity.

## 4.3 Adaptive Dissemination Techniques

Deligiannakis et al. [113] introduce one of the first adaptive dissemination techniques for remote edge sources. In particular, the authors suggest a traffic reduction framework capable of running on the remote sensing device. Their framework considers buffering large amounts of metric values at each sensing device and, rather than transmitting the total of the buffer contents, it transmits a base signal (wavelet) of fewer values which is then used to reconstruct an approximate version of the original signal. The downside of this approach is that it requires for a large portion of the signal to be made available and stored on the device so as to provide an estimation. In turn, Silberstein et al. [108] introduce CONCH. Specifically, CONCH is an edge-monitoring mechanism for sensing networks, where monitoring query response collection and dissemination is suppressed if responses do not differ across the nodes comprising the network. The proposed mechanism reduces bandwidth and energy consumption inferred from constant monitoring response dissemination by discovering spatio-temporal data correlations among the edge nodes of the network but with the caveat that sensors must have knowledge of the entire network topology or its neighborhood if a partitioning scheme is followed.

Another relevant monitoring framework is Sieve [4], which targets reducing the dimensionality of collected metric streams from large-scale micro-service deployments. This is achieved by collecting, at first, all metric streams and then applying a k-shape clustering [134] so that similar-behaving metrics are clustered together. After clustering, Sieve picks a representative metric from each cluster and only stores and uses the representative to infer metric behavior for metrics of the cluster. This significantly reduces the metric dimensionality as $k$ metric streams are indexed instead ($k << N$). The downside of this approach is that it assumes that the metric stream behavior is always as expected, as there is no provision for anomalous be-

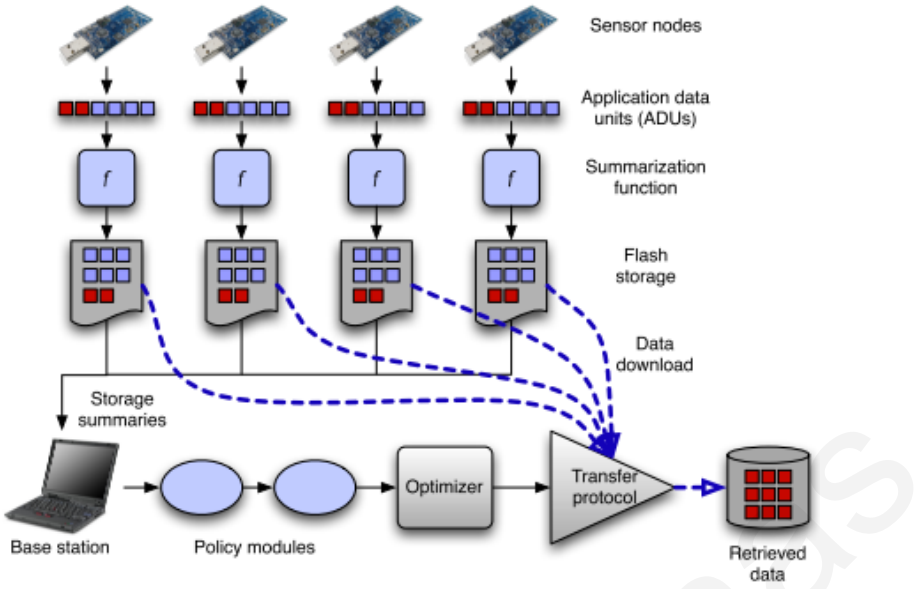



*Figure 4.5: LANCE System Architecture – Reprinted from [Werner et al., ACM SenSys, 2011] [112]*

havior, and that the value-base distribution will not change in time, thus reducing the need for subsequent clustering.

ADWIN [135] is an adaptive shift detection framework for streams. It uses a Naive Bayes predictor to maintain up-to-date estimations of the conditional probabilities describing a metric stream and is able to reduce shift detection delays and the false alarm ratio. To achieve this, it follows a linear approach with two sliding windows to detect shifts based on given confidence intervals. However, while dissemination is triggered when a shift is detected, all metric values collected up to the shift are still disseminated. In turn, Matsubara et al. propose RegimeCast [136], a tensor-based monitoring framework that detects arbitrary length shifts in metric streams. RegimeCast also forecasts possible future events (e.g., a person after sweeping floor most likely will mop it). However, the downside of RegimeCast is that its algorithmic process is computationally intensive and, thus, operates as a server-side framework to coordinate and support IoT devices remotely.

Similar to the adaptive filtering step-wise techniques, previously introduced, Kim et al. [114] propose using a feedback loop to dynamically adjust the metric dissemination rate of mobile biosignal reporting devices. In particular, the authors suggest monitoring the Received Signal Strength Indication (RSSI)[2] modeled as a weighted average summarizing the $n-$latest datapoints. If the RSSI exceeds or

[2] RSSI is a measure of the strength of the power level that a client is receiving from a radio transmitting device [137]

**Algorithm 4** G-SIP Adaptive Dissemination – Adapted from [Gaura et al., IEEE Trans. on Sensors, 2013] [10]

**Input:** $\mathbf{z}$ sliding vector containing recent stream datapoint readings, $t_{new}$ current timestamp, and $\epsilon$ user-defined acceptable imprecision

**Output:** $\mathbf{y}_{new}$ latest state at mon. source and $y_s$ predicted state at receiver-side (sink)

1: $\mathbf{y}_{new} = (y, \dot{y})^T \leftarrow$ estimateLocalState($\mathbf{z}$, $\mathbf{y}_{old}$, $t_{old}$) //based on dEWMA
2: $\mathbf{y}_s \leftarrow$ predictSinkState($\mathbf{y}_{sink}$, $t_{new}$)
3: **if** eventful($\mathbf{y}_{new}$, $y_s$, $\epsilon$) $\|$ $t_{new} - t_{old} > T$ **then**
4: transmit($\mathbf{y}_{new}$, $t_{new}$)
5: $\mathbf{y}_{sink} \leftarrow \mathbf{y}_{new}$
6: **end if**
7: $\mathbf{y}_{old} \leftarrow \mathbf{y}_{new}$
8: $t_{old} \leftarrow t_{new}$
9: **return** $\mathbf{y}_{new}$

drops below certain high/low thresholds then the dissemination rate of the device is adjusted accordingly by a user-defined positive/negative *step* value. However, unlike conventional dissemination control protocols used in signal monitoring, instead of using a fixed threshold (TH), the authors suggest using a Variable Threshold Level ($VTL = TH \pm \beta \cdot \sigma$) adjusted based on the RSSI variability, denoted by the signal standard deviation ($\sigma$), and $\beta$ denoting a user configurable allowance parameter. However, while the aforementioned approaches are interesting, they assume that no distribution shifts will occur in the monitoring stream evolution at runtime.

A more comprehensive approach for IoT and WSN networks is followed by Werner et al. [112]. Specifically, the authors introduce LANCE, a framework for signal collection that reduces bandwidth and energy consumed for monitoring stream dissemination over the network. LANCE sends summaries of windowed data, usually in the form of an average, from nodes part of a sensor network to metric stream receivers referred to as data sinks. Data sinks then decide as to how useful the data are by comparing summaries to user-defined policies (e.g., metric value thresholds). Thus, if the summary violates the defined policy, the high resolution data described by the summary should be retrieved from the remote sensor; otherwise the data is discarded. Since summaries are disseminated across the network and not the entire window length of data values, both energy and bandwidth are preserved when

summaries denote that no significant changes in the metric stream have occurred.

Finally, Gaura et al [10] introduce G-SIP, an adaptive dissemination framework for IoT devices with similar logic to LANCE. In particular, G-SIP disseminates monitoring updates only when the metric stream value distribution produced by the monitoring source changes in a way that cannot be predicted from previous value knowledge by the interested entity receiving monitoring state updates. As a prediction mechanism, G-SIP uses an exponential weighted average to follow the rate at which the metric stream changes in time. Therefore, if the rate of change exceeds a user-defined threshold, thus labelling the change as "eventful", metric updates are sent to the remote receiving entity, otherwise dissemination is suppressed with the latest monitoring state update being filtered (Algorithm 4). While LANCE and G-SIP are interesting approaches targeting adaptive dissemination, the downside of these frameworks is that they are either slow to react to abrupt and volatile monitoring streams and/or static thresholds are used which are fixed once upon initialization.

**Chapter Summary.** After trawling the literature and industry advancements, this chapter presents relevant work in relation to adaptive sampling, adaptive filtering and model-based adaptive dissemination. Despite advancements in the field, the current State-of-the-Art in approximate and adaptive techniques for monitored networks with processing conducted at the edge, still require matureness as in many cases these techniques: (i) require excessive profiling to configure optimal framework parameters, a task difficult for users; (ii) present large runtime footprint in regards to energy consumption or resource utilization reducing the benefits of introducing adaptiveness in the end; (iii) fail to acknowledge abrupt and transient shifts in the evolution of monitoring data or assume that once determined there will be no distribution shifts in the monitoring stream value base; or (iv) require coordination from server-side components.

# Chapter 5

# AdaM: The Adaptive Monitoring Framework

## 5.1 Overview

To address the challenges introduced in this thesis, we have designed and developed the Adaptive Monitoring framework[1]. AdaM is a lightweight framework embeddable in the software core of monitoring sources (e.g., IoT devices) that provides model-based adaptive monitoring. In particular, AdaM dynamically adjusts the rate at which monitoring metrics are collected and disseminated to interested receiving entities based on extracted runtime knowledge capturing the evolution and variability of the metric stream. By accomplishing this, energy consumption and data volume are reduced, allowing monitoring sources, such as IoT and edge devices, to preserve energy and resource utilization, while at the same time ease processing on monitoring data receiving services.

To achieve this, AdaM incorporates low-cost adaptive and probabilistic learning algorithms for **adaptive sampling, filtering and model-based dissemination**, adjusting the metric collection and dissemination rate of the monitoring source based on the confidence of each algorithmic model to correctly estimate what will happen next in the metric stream. In turn, specific consideration is also taken so that the algorithmic model is fine-tuned at runtime by introducing *adaptive parameter weighting*, *trend detection* and *seasonality behavior enrichment* in order for the adaptive monitoring techniques to immediately identify abrupt transient changes in the monitoring stream evolution and overcome any lagging effects in the estimation process.

AdaM has been initially developed in Java to support IoT devices, such as microcontrollers (e.g., Raspberry Pi) and Android (wear) devices, but as it features no

[1] http://linc.ucy.ac.cy/AdaM/

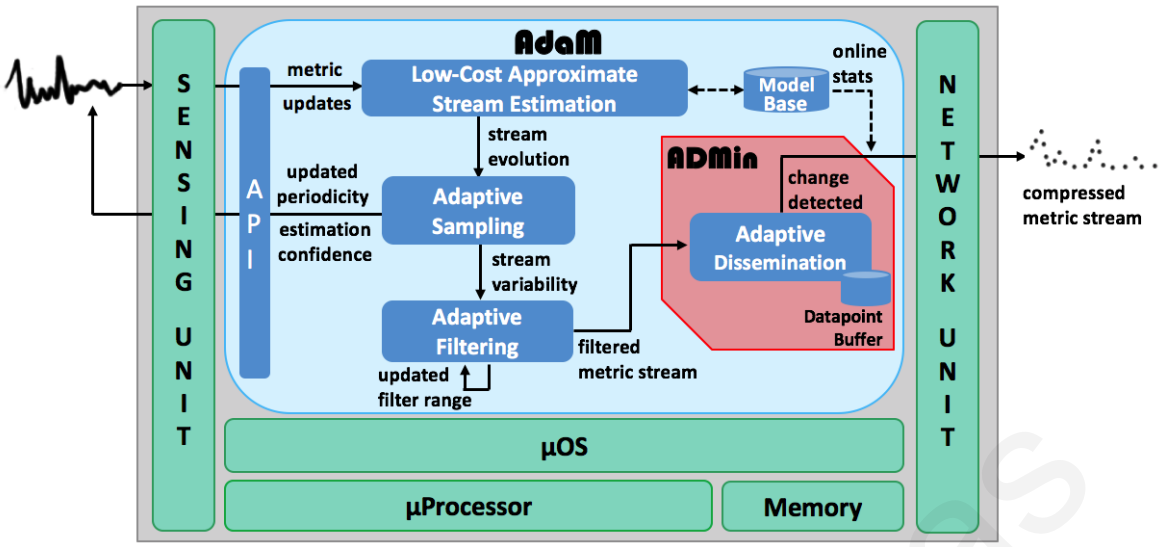



*Figure 5.1: AdaM - the adaptive monitoring framework*

external source code dependencies it can also be ported to other real-world settings and popular programming frameworks. Real-world examples include cloud monitoring (Section 5.7.2) and intelligent transportation services (Section 5.7.2), while AdaM has also been ported to R for offloading graph metric computation [138], and Python for efficient data stream processing in Apache Flink [139]. Figure 5.1 depicts a high-level and abstract overview of AdaM embedded in the software core of an IoT device, where it coordinates metric sensing and dissemination by interacting, as a proxy, between the Sensing and Network Unit of the IoT device.

In particular, when the Sensing Unit of the IoT device collects a new datapoint ($d_i$) it is passed through the AdaM API to the *Low-Cost Approximate Stream Estimation* module. This module updates a local reference estimation model capturing the monitoring stream evolution and variability, while also maintaining in the *Model Base* a number of online statistics (e.g., mean, standard deviation, estimation confidence) that are used by AdaM's adaptive algorithms and may also be of interest to users and metric stream receiving entities. The adaptive techniques offered by AdaM share the estimation model by reusing both extracted knowledge and intermediate results to reduce the overhead imposed to the monitoring source even more. In turn, to assist the estimation process to better follow the current monitoring stream evolution, *adaptive parameter weighting*, *trend detection* and *seasonality behavior enrichment* are used to immediately identify abrupt transient changes in the monitoring stream evolution and also reduce any lagging effects in the estimation process, as IoT data such as human body indicators and environmental data, present such be-

havior [140] [141]. In the particular case where seasonality enrichment is not helpful in the estimation process, AdaM detects this using online statistical testing, thus ignoring its contribution.

After updating the local reference estimation model, the *Adaptive Sampling* module will use the current monitoring stream evolution and variability to return a new estimation of the sampling period ($T_{i+1}$) and a confidence interval for the current estimation. The Sensing Unit may then use $T_{i+1}$ to collect the next datapoint ($d_{i+1}$) and return to an idle state. If adaptive filtering is enabled as an intermediate between the adaptive sampling and dissemination module, then the current measurement is forwarded to the *Adaptive Filtering* to decide if the current datapoint ($d_i$) should be discarded or not. In turn, the filter range ($R_{i+1}$) is adjusted based on the monitoring stream variability and an indicator of the current variability is made available to users via the AdaM API. If the datapoint is not filtered, it will be forwarded to the AdaM *Adaptive Dissemination* module.

If model-based dissemination is enabled, then instead of disseminating metric values, AdaM favors sending estimation model updates from which interested receiving entities can infer the metric values. Therefore, after filtering, the Adaptive Dissemination module is used to determine if there is a distribution shift in the monitoring stream value-base, thus rendering the estimation model as inconsistent or if the datapoint *Local Buffer* has reached maximum capacity. If so, the Network Unit of the IoT device is enabled and a compressed message containing an updated version of the estimation model and the contents of the local storage, is disseminated to interested receivers. Otherwise, monitoring dissemination is suppressed with the Network Unit remaining in an idle state to preserve energy.

In the remainder of this Chapter, we will introduce the Low-Cost Approximate Stream Estimation module which is used to capture runtime knowledge of the monitoring stream evolution and variability, and then, elaborate on how the sampling rate and filter range are dynamically adjusted by the Adaptive Sampling and Adaptive Filtering modules, respectively. In the next Chapter, we will introduce ADMin, a plugin developed to offer Model-Based Adaptive Dissemination by advancing the functionality of the AdaM framework to include seasonality knowledge enrichment, datapoint forecasting and runtime shift detection.

## 5.2 Requirements and Objectives

Obviously, if a degree of inaccuracy cannot be tolerated for a certain monitoring task (e.g., hospital patient vitals) then any framework or technique adapting the properties of a monitoring source, either this is the metric collection or dissemination rate, will fail. Therefore, *absolute guarantees defeat the purpose of approximate and adaptive monitoring*. Nonetheless, even in the case of fixed-rate monitoring, there are no guarantees that the sampling rate is optimal and even with the utilization of a potentially high sampling rate, error, or better inaccuracy, in the form of unreported points of interest can still be masked between two consecutive datapoints. In turn, if a degree of imprecision is tolerable but the estimation model cannot follow the metric stream evolution, then constant model training will be required, especially in the case of model-based adaptive dissemination. Hence, while metric dissemination will be replaced with model updating, the energy drain, from activating the network controller, will not be reduced.

Thus, the following requirements must be taken into consideration when designing a framework for adaptive monitoring:

**R1**: The estimation process must be lightweight and performed in place right on the monitoring source itself.

**R2**: The estimation process must be efficient, meaning it must infer overall less costs than actually collecting and disseminating all datapoints and later discarding them at the receiver-side.

**R3**: While parameters of the framework can be tweaked, no user should be required to enter "magic numbers" for any given parameter.

**R4**: The framework must be practical, achieving good performance for numerous and diverse real-life testbeds.

**R5**: The framework must ensure that the user-defined accuracy guarantees are obeyed at all times.

Hence, our main objective is to provide an estimation model capable of capturing runtime knowledge of the metric stream evolution and variability to produce approximate datapoint values within given confidence guarantees and detect when these guarantees are violated to update the model, used by the underlying adaptive monitoring techniques, in real-time. This will allow monitoring sources (e.g., IoT devices) to preserve energy by reducing the volume of both sampled and dissem-

inated monitoring data while ensuring accuracy guarantees are maintained at all times.

## 5.3 Monitoring Stream Data Model

The following requirements must be met to attach a data stream whose properties will be dynamically adjusted by the AdaM framework:

- Incoming datapoints are tuples that must be in delimiter-separated format (default separator is `','`) and include the following ordered fields: the monitoring source id, current timestamp in unix format, current value, and an optional list of attributes.

```
datapoint:  <srcID,timestamp,val,[attr,]>
```

- If a `srcID` is not defined for a monitoring stream then AdaM will provide a unique identifier. Any incoming datapoints with no `srcID` will be assumed to be of the same origin.

- Datapoint values must be *numeric* (e.g., `int`, `long` or `float`), meaning, the type of the data must be quantitative and arithmetic operations can used to compute the degree of difference between metric values. In turn, AdaM only accepts *univariate* datapoint values, meaning datapoints must be one-dimensional.

## 5.4 Low-Cost Approximate Metric Stream Estimation

To satisfy the aforementioned set of requirements and objectives, we base our approach such that the estimation model used by the introduced adaptive monitoring techniques, is maintained in constant time and space (O(1) complexity), thus satisfying **R1**, which requires low-cost estimation model able to run on monitoring sources (e.g., IoT devices) with limited processing capabilities. In turn, our estimation model incorporates enough knowledge of the metric stream to allow us to provide long-range approximations, thus reducing continuous model updates and satisfying **R2**. Applying an adaptive algorithm is only meaningful if the process can be done inexpensively and online. Therefore, the cost of applying adaptiveness must be less than actually collecting the datapoints and then discarding them. The

**Algorithm 5** AdaM: Approximate and Adaptive Estimation Model

**Input:** current datapoint $d_i$ with timestamp $t_i$ and value $v_i$

**Output:** updated estimation model

1: **if** $t_i > 0$ **then**

*compute current distance*

2: $\delta_i \leftarrow |v_i - v_{i-1}|$ (eq. 5.1)

*compute estimation error, and p- and z- value*

3: $\epsilon_i \leftarrow \delta_i - \hat{\delta}_i$

4: $P_i, Z_i \leftarrow \text{probDistro}(\epsilon_i, \hat{\sigma}_i)$ (eq. 5.5)

*update estimation model for next time interval*

5: $\mu_i \leftarrow \text{updPEWMAwithTrend}(P_i, \delta_i, x_i)$ (eq. 5.7)

6: $x_i \leftarrow \text{updHoltTrend}(\mu_i)$ (eq. 5.6)

*update estimated and observed moving variance*

7: $\sigma_i, \hat{\sigma}_{i+1} \leftarrow \text{updSD}(\mu_i, x_i, \delta_i, \epsilon_i)$ (eq. 5.8)

*compute current estimation confidence*

8: $c_i \leftarrow \text{calcConfidence}(\sigma_i, \hat{\sigma}_i)$ (eq. 5.10)

*computer upper/lower control boundaries*

9: $BH_i, BL_i \leftarrow \mathit{calcBounds}(\delta_i, \sigma_{err})$ (eq. 5.9)

10: **else**

11: $\hat{\delta}_{i+1} = \mu_i \leftarrow v_0,\ \hat{\sigma}_{i+1} \leftarrow 0$ //init values

12: **end if**

13: **return** $\text{estModel}(\hat{\delta}_{i+1}, \hat{\sigma}_{i+1}, c_i, \sigma_{err})$

adaptive monitoring framework supports model parameterization, although as input for each adaptive technique to be used, only requires from the user to provide the certain accuracy guarantees that must be obeyed by the estimation model, thus satisfying **R3**. No domain-specific information for the monitoring data is required, thus providing a generic framework and satisfying **R4**. Dynamically adjusting the properties of a monitoring source is based on the ability of the algorithmic process to (correctly) estimate what will happen next in the metric stream. If an estimation cannot be made within certain confidence intervals obeying the accuracy guarantees given by the user the adaptive technique will rollback to a fixed approach to ensure accuracy over efficiency at all times, thus satisfying **R5**. In light of the above, Algorithm 5 presents our approximate and adaptive estimation model.

At first, we will compute the distance $\delta_i$ between the current two consecutive datapoint values, as follows:

$$\delta_i = |v_i - v_{i-1}| \tag{5.1}$$

The distance $\delta_i$ is used to update the local reference runtime evolution of the metric stream $\rho(M)$. We compute the current metric evolution (steps 2-7) by using a moving average, denoted as $\mu_i$. This provides an estimation of the evolution followed by the metric stream, and is used to estimate the distance of the next two consecutive values, denoted as $\hat{\delta}_{i+1}$. Intuitively, a large distance between the two consecutive values denotes a shift in the metric evolution. Hence, if a large distance is not expected, a decrease in the sampling period should be considered, whereas if the distance is small, then an increase in the sampling period can be considered. Moving averages provide one-step ahead predictions. They are easy to compute and are calculated on the fly with knowledge of only the previous value $\mu_{i-1}$. Equation 5.2 presents an example of a cumulative Simple Moving Average (SMA) where values of a sliding window are aggregated evenly:

$$\mu_i = \frac{\delta_i + (i-1)\mu_{i-1}}{i},\ i \geq 1 \tag{5.2}$$

While a SMA can be used, it weighs all values the same. This is not desired as recent disrupts in the metric evolution should be highly valued in a dynamic metric stream. To address this, an Exponential Weighted Moving Average (EWMA) can be used, where a weighting factor ($0 < \alpha < 1$) is introduced to decrease exponentially the effect of older values [142], as presented in Equation 5.3. While the EWMA is a better suit for our needs it still features one significant drawback; it is volatile to abrupt transient changes. Therefore, any assumption made that the EWMA only changes gradually with respect to the parameterization (exponential weighting), is not always the case [143]. Specifically, the EWMA, as depicted in Figure 5.2, is slow to acknowledge sudden spikes after large stable phases, and, if stable phases follow sudden bursts, spike effects are preserved in the estimation. This results in overestimating subsequent $\delta_i$'s which affect the accuracy of an adaptive technique.

$$\mu_i = \begin{cases} \delta_i, & i = 1 \\ \alpha\mu_{i-1} + (1-\alpha)\delta_i, & i > 1 \end{cases} \tag{5.3}$$

**Probabilistic Weighting.** To dynamically adjust the weighting based on the probability density of the given observation, we adopt a Probabilistic EWMA (PEWMA). The PEWMA acknowledges sufficiently abrupt transient changes, adjusting quickly to long-term shifts in the metric evolution and when incorporated in our algorithmic estimation process (steps 2-7), it requires no parameterization, scaling to numerous datapoints. Equation 5.4 presents the PEWMA where instead of a fixed weighting factor, we introduce a probabilistically adaptable weighting factor $\tilde{a_i} = \alpha(1 - \beta P_i)$. In this equation, the p-value, is the probability of the current $\delta_i$ to follow the modeled distribution of the metric stream evolution. In turn, $\beta$ is a weight placed on $P_i$ and as $\beta \to 0$ the PEWMA converges to a common EWMA[2].

$$\mu_i = \begin{cases} \delta_i, & i = 1 \\ \alpha(1 - \beta P_i)\mu_{i-1} + (1 - \alpha(1 - \beta P_i))\delta_i, & i > 1 \end{cases} \tag{5.4}$$

The logic behind probabilistic reasoning is that the current $\delta_i$ depending on its p-value will contribute respectively to the estimation process. Therefore, we update the weighting by $1 - \beta P_i$ so that sudden "unexpected" spikes are accounted for in the estimation process, however, offer little influence to subsequent estimations, thus restraining the model from overestimating subsequent $\delta_i$'s. In turn, if an "unexpected" value turns out to be a shift in the metric stream evolution, as the probability kernel shifts, subsequent "unexpected" values are awarded with greater p-values, allowing them to contribute more to the estimation process.

Assuming, as mentioned in Section 3.1, a stochastic and i.i.d distribution as the bare minimum for a metric stream, we adopt a Gaussian kernel $N(\mu, \sigma^2)$, which satisfies the aforementioned requirements. Thus, $P_i$ is the probability of $\delta_i$ evaluated under a Gaussian distribution, which is computed by Eq. 5.5. Nonetheless, we note that while a Gaussian distribution is assumed, if prior knowledge of the distribution is made available and given by the user then only step 4 from Algorithm 5 must change in the estimation process.

$$\begin{aligned} P_i &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{z_i^2}{2}) \\ Z_i &= \frac{\delta_i - \hat{\delta}_i}{\hat{\sigma}_i} \end{aligned} \tag{5.5}$$

[2] For simplicity in our model evaluation we will consider $\beta = 1$

In Equation 5.5, $\delta_i - \hat{\delta}_i$ denotes the estimation error $\epsilon_i$ which is the difference between the observed and estimated distance, while $\hat{\sigma}_i$ denotes the (moving) standard deviation.

**Trend Detection.** While adaptive weighting refrains the model from overestimation at bursty time intervals, it does not account for monotonic phases of upward and downward trends which often introduce *time lagging effects* in the estimation process [13]. To fine-tune the estimation process by capturing possible trends, we use Holt's Trend Method to estimate the current monotonic growth/decay in the metric stream evolution [144].

$$x_i = \begin{cases} \delta_i - \delta_{i-1}, & i = 2 \\ \xi\,(\mu_i - \mu_{i-1}) + (1-\xi)\,x_{i-1}, & i > 2 \end{cases} \tag{5.6}$$

Equation 5.6 depicts how the trend, denoted as $x_i$, is updated at each time interval, where $\xi$ is a smoothing weight in the range $[0, 1]$ with values near 1 denoting a preference to favor recent trends. Thus, any lagging effects in the estimation process are reduced by boosting the moving average to the appropriate value base with the PEWMA for $\hat{\delta}_{i+1}$ now also incorporating an additive trend component (eq. 5.7). It may be argued that a statistical test (e.g., Mann-Kendall test) should be used to decide if the additive trend affects positively the estimation. However, with adaptive weighting based on the "expectiveness" of the estimated $\delta_i$ this is not needed as the additive component including the trend, in Equation 5.7, will receive a low weighting $\tilde{a}_i$ and, thus, reduce its contribution to the estimation when a trend is not evident.

$$\mu_i = \tilde{a}_i(\mu_{i-1} + x_{i-1}) + (1 - \tilde{a}_i)\delta_i \tag{5.7}$$

At this point, the metric stream evolution $\rho(M)$, encapsulated by $\hat{\delta}_{i+1}$ and $\hat{\sigma}_{i+1}$, can be efficiently updated with only previous value knowledge and without repeatedly scanning the entire stream ($n \to \infty$), as follows:

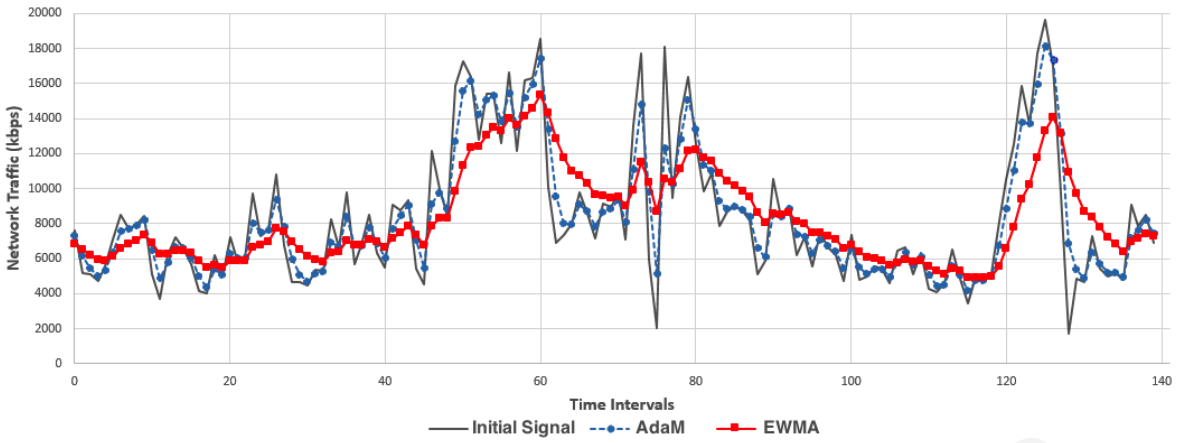



*Figure 5.2: AdaM estimation model compared to an EWMA-based estimation model*

$$\begin{aligned} \tilde{a}_i &\leftarrow \alpha(1-\beta P_i) \\ \theta_1 &\leftarrow \tilde{a}_i \cdot (\theta_1 + x_i) + (1-\tilde{a}_i) \cdot \delta_i \\ \theta_2 &\leftarrow \tilde{a}_i \cdot \theta_2 + (1-\tilde{a}_i) \cdot \delta_i^2 \\ \hat{\delta}_{i+1} &\leftarrow \theta_1 \\ \hat{\sigma}_{i+1} &\leftarrow \sqrt{\theta_2 - \theta_1^2} \end{aligned} \tag{5.8}$$

Figure 5.2 depicts a comparison between AdaM and an estimation model limited to an EWMA [120]. We observe that AdaM is quick to adjust to shifts in the metric evolution, and in contrast to the EWMA, after spikes it does not overestimate subsequent values. Having estimated the standard deviation, when the next sample is collected, the algorithm will update the actual observed moving standard deviation $\sigma_i$ for $\delta_i$ and $\sigma_{err}$ for $\epsilon_i$ (step 7). We note that, $\sigma_{err}$ is not used in adaptive sampling but in adaptive filtering. Also, $\sigma_{err}$ is a useful statistic for users since from $\sigma_{err}$, the current moving estimation $\delta_i$ and a multiplier $K$; high/low estimation control boundaries for detecting outliers and performing change detection are computed as follows:

$$B_{i,high},\ B_{i,low} \leftarrow \delta_i \pm K \cdot \sigma_{err} \tag{5.9}$$

On the other hand, $\sigma_i$ is used to calculate the current confidence, denoted as $c_i$ (step 8). The confidence ($c_i \leq 1$) is a ratio computed from the difference between the estimated and the observed standard deviation (eq. 5.10) and is used as our error evaluation metric which denotes the ability of the algorithmic process to (correctly) estimate what will happen next in the metric stream. This supports our framework to "reward" larger property adjustments when estimations fall within the accuracy

intervals defined by the user and rollback to a fixed approach when satisfactory estimations cannot be provided. Thus, the semantics behind the confidence is that: *the more "confident" the algorithm is, the larger the property adjustment (e.g., periodicity) for the monitoring source can be*. Hence, as $\hat{\sigma}_i \rightarrow \sigma_i$ the confidence $c_i \rightarrow 1$.

$$c_i = 1 - \frac{|\hat{\sigma}_i - \sigma_i|}{\sigma_i} \tag{5.10}$$

Based on the above, let us consider the case of an adversary purposely feeding AdaM with datapoints that do not adhere the adopted metric value distribution model. In this case, the confidence of the estimation model will be low. If the confidence, at any point, cannot provide an estimation satisfying the maximum tolerable imprecision, then any employed adaptive technique (e.g., adaptive sampling) will rollback to a fixed approach (e.g., periodic sampling) to ensure accuracy is met. Hence, accuracy is guaranteed by AdaM while efficiency depends on the incoming data arrival distribution with efficiency maximized if both the data and the estimation model adhere to a common distribution.

## 5.5 Adaptive Sampling

Adaptive sampling is the process of dynamically adjusting the periodicity $T_i$ at which a metric stream is sampled based on the evolution and variability of the metric stream while still adhering to the accuracy guarantees given by the user. We base our approach such that the estimated sampling period $T_{i+1}$ to collect the next datapoint, is dependent to the current *sampling period* $T_i$, increasing if variability of the load decreases, and, in turn, decreasing if variability increases. How large of an adjustment is required, is dependent on the *confidence metric* $c_i$, denoting the confidence of the algorithm to (correctly) estimate and follow the metric stream current evolution. Therefore, when the estimation model is "confident", the adaptive sampling algorithm will output/award larger sampling periods. Hence, in contrast to threshold-based techniques which adjust the sampling rate solely based on the datapoint value $v_i$, our approach considers the metric stream evolution, as well as, the confidence of the estimation. The reason for this lies in the failure of threshold-based techniques [10] [11] to detect variances in the metric stream when values are far from the threshold (e.g. $v_i << \tau$, where $\tau$ a user-defined threshold). Hence, events remain undetected such as in the case of low rate DDoS attacks [145]. Similarly,

**Algorithm 6** Adaptive Sampling

**Input:** imprecision $\gamma \in [0, 1]$ given by user and confidence $c_i$

**Output:** $T_{i+1}$

**Ensure:** $\{T_{i+1} \mid T_{i+1} \in \mathbb{Z}^+ \text{ and } T_{i+1} \in [T_{min}, T_{max}]\}$

*if $T_{i+1}$ can be adjusted (either up or down) based on the determined confidence $c_i$, and user-defined imprecision $\gamma$, then do so, else rollback to default $T_{min}$*

1: **if** $t_i > 0$ **then**
2: **if** $(c_i \geq 1 - \gamma)$ **then**
3: $T_{i+1} \leftarrow T_i + \lambda \cdot (1 + \frac{c_i - \gamma}{c_i})$ (eq. 5.11)
4: **if** $(T_{i+1} > T_{max})$ **then**
5: $T_{i+1} \leftarrow T_{max}$
6: **end if**
7: **else**
8: $T_{i+1} \leftarrow T_{min}$
9: **end if**
10: **else**
11: $T_{i+1} \leftarrow T_{min}$ //init values
12: **end if**
13: **return** $T_{i+1}$

stable phases with high values will fail to receive a sampling period decrement as well, as a violation is still probable.

Thus, after updating the estimation model, Algorithm 6 is used to perform adaptive sampling. Having computed the current confidence (eq. 5.10), we then compare it to the acceptable user-defined imprecision, denoted as $\gamma$ from the problem definition. The imprecision parameter ($\gamma \in [0, 1]$) is used to set the sensitivity while computing a new sampling period $T_{i+1}$ (eq. 5.11).

$$T_{i+1} = \begin{cases} T_i + \lambda \cdot (1 + \frac{c_i - \gamma}{c_i}), & c_i \geq 1 - \gamma \\ T_{min}, & else \end{cases} \tag{5.11}$$

Intuitively, if $\gamma \to 0$ then our algorithm converges to a periodic sampling approach (unless an "exact" estimation is made). In turn, if $\gamma \to 1$ an adjustment will take place on each interval even if a confident estimation cannot be made. Hence,

if the algorithm cannot provide an estimation within a certain confidence interval, then our adaptive sampling algorithm will rollback to the default sampling period $T_{min}$ for the next datapoint estimation $d_{i+1}$. Moreover, in contrast to stepwise techniques [110] [120] which adjust the sampling rate solely based on a step function (e.g., $T_{i+1} \leftarrow T_i \pm T_{step}$), our approach is quick to react to highly volatile metric streams adapting the sampling rate based on its confidence to the appropriate period in the range $[T_{min}, T_{max}]$.

The complexity of our approach is $O(1)$ constant time, since all calculations are based on previous collected values and do not require the entire metric stream to be available. Moreover, the imprecision $\gamma$, is the only parameter which is user-defined in the estimation process. Nonetheless, users are free to change: (i) $\lambda$ which is an optional multiplicity factor (e.g. default $\lambda = 1$) to be used if a more aggressive approach should be followed; and (ii) the weights $\alpha$ and $\xi$, although as shown in our evaluation, $\alpha$ and $\xi$ may take a wide range of values due to the probabilistic weighting process and can be left to default values for a small imprecision penalty.

## 5.6 Adaptive Filtering

Metric filtering is the process of suppressing datapoints when consecutive datapoint values differ less than a range of values, denoted as $R$. Therefore, the current datapoint $d_i$ with value $v_i$ is filtered, if $v_i \in [v_{i-1} - R,\ v_{i-1} + R]$. However, this assumes that the user has previous knowledge of the datapoint value distribution and that it will not change in the future, otherwise, there is no guarantee any values will be filtered at all. To address this downside, we adopt Adaptive Filtering, to dynamically adjust the filter range $R$ based on the metric stream variability while still adhering to the accuracy guarantees defined by the user. Thus, the job of adaptive filtering is to adjust the filter range in an attempt to filter consecutive datapoints but only if the accuracy guarantees given by the user can be obeyed.

In our approach, the filter range $R$ is dependent to the metric stream variability. The reason for this lays in the failure of stepwise techniques [34] [132], which adjust $R$ incrementally based on the number of datapoints previously filtered, in cases such as biosignal monitoring where precision is required in a small range of values [130]. For instance, let us consider glucose monitoring as a suitable example where the filter range is adapted ($R_i \leftarrow R_{i-1} \pm r_{step}$) based on stepwise adjustments, denoted as

**Algorithm 7** AdaM: Adaptive Filtering

**Input:** $\mu_i$, $\sigma_i$ and $\sigma_{err}$ from estimation model

**Output:** $R_{i+1}$

**Ensure:** $\{R_{i+1} \mid R_{i+1} \in [R_{min}, R_{max}],\ 0 \leq R_{min} \leq R_{max}\}$

*if metric stream not dispersed and inaccuracy budget permits it then $R_{i+1}$ can be widen, else it is shortened*

1: **if** $t_i > 0$ **then**
2: $F_i \leftarrow calcFanoFactor(\sigma_i, \mu_i)$ (eq. 5.12)
3: **if** $F_i < 1$ **then**
4: **if** $\sigma_{err} < \gamma$ **then**
5: $R_{i+1} \leftarrow R_i + \lambda \cdot (\frac{\gamma - \sigma_{err}}{\gamma})$ (eq. 5.13)
6: **if** $(R_{i+1} > R_{max})$ **then**
7: $R_{i+1} \leftarrow R_{max}$
8: **end if**
9: **else**
10: $R_{i+1} \leftarrow R_i$
11: **end if**
12: **else**
13: $R_{i+1} \leftarrow R_{min}$
14: **end if**
15: **end if**
16: **return** $R_{i+1}$

$r_{step}$. In this case, ignoring metric variability will result in filtering out critical values, if $r_{step}$ is too large, or, if too small, will require multiple (delayed) adjustments to achieve the desired value-base when filtering must occur.

To show the extent of the variability in relation to the current evolution of the monitoring stream, we utilize the Fano factor [146]. In particular, the Fano factor ($F \geq 0$), like the index of dispersion, is a normalized measure of the dispersion of a probability distribution, which is used to quantify whether a set of datapoints are currently clustered ($F < 1$) or dispersed compared to a statistical model. The Fano factor is calculated over a time window, denoted as W, as the ratio of the variance $\sigma^2$ to the mean $\mu$, as presented in Equation 5.12:

$$F = \frac{\sigma_w^2}{\mu_w} \tag{5.12}$$

Algorithm 7 depicts how AdaM employs adaptive filtering. To provide both the variance $\sigma^2$ and the mean $\mu$, no additional calculations are required, as both $\sigma_i$ and $\mu_i$ are the probabilistically weighted output of the metric stream evolution provided by the estimation model. This also deficits the need of using a window of datapoints as both $\sigma_i$ and $\mu_i$ capture previous value knowledge and apply weighting to adjust the contribution of each datapoint accordingly. Intuitively, when $\sigma_i$ decreases, the Fano factor $F_i$ follows, indicating a decrease in the variability of the metric stream. Having computed $F_i$, we then compare $\sigma_{err}$ to the user-provided maximum tolerable imprecision, denoted as $\gamma$. If $F_i$ indicates the metric stream is not dispersed and $\sigma_{err}$ is less than $\gamma$, then the filter range is widen, in an attempt, to filter *near-by* values while still remaining in the accuracy guarantees defined by the user (eq. 5.13). Otherwise, if $F_i$ indicates the metric stream is currently over-dispersed, the filter range is shortened or restored to a default value in order to report abnormalities in the data.

$$R_{i+1} = \begin{cases} R_i + \lambda \cdot (\frac{\gamma - F_i}{\gamma}), & F_i < 1 \; and \; \sigma_{err} < \gamma \\ R_i, & F_i < 1 \; and \; \sigma_{err} > \gamma \\ R_{min}, & else \end{cases} \tag{5.13}$$

As with adaptive sampling, the adaptive filtering algorithm has a $O(1)$ constant time and space complexity, as $R_{i+1}$ is computed from its previous value, while $\mu_i$, $\sigma_i$ and $\sigma_{err}$ are the output of the runtime estimation model described in the previous section. Additional parameter configurable by users, is $\lambda$, a multiplier used as an aggressiveness indicator.

## 5.7 Experimentation Study

In this section we present a thorough evaluation of AdaM adaptive sampling and filtering modules, based on public data from cloud services, internet security services, wearables and intelligent transportation systems. First, we present an on device evaluation of AdaM compared to other adaptive techniques. Second, we present a scalability evaluation based on two different big data streaming services which benefit from embedding AdaM in their edge nodes to lower data volume and velocity

while preserving accuracy.

### 5.7.1 On Device Accuracy & Efficiency Evaluation

We compare AdaM to three state-of-the-art adaptive techniques frameworks that are suitable for IoT devices: L-SIP [10], i-EWMA [110] and FAST [41], introduced in detail in Chapter 4. Briefly:

- **L-SIP** is a linear technique for adaptive sampling which uses a double exponential moving average to produce estimates of the current data distribution based on the rate sample values change in time.
- **i-EWMA**[3] is a technique encompassing a moving average which increases the sampling period incrementally by one time unit ($T_{i+1} \leftarrow T_i + T_{unit}$), when the estimated error $\epsilon$ is under a user-defined imprecision value $\gamma$, and decreases it ($T_{i+1} \leftarrow T_i - T_{unit}$), when $\epsilon > \gamma$.
- **FAST** is a framework for differential privacy using a PID controller to determine the periodicity accompanied by a Kalman filter to predict values at non sampled points. To configure the Kalman filter $R$ parameter a training phase of 10 intervals was introduced. As differential privacy is not under-evaluation, it is not enabled.

Unless otherwise stated, the user-defined imprecision is set to $\gamma = 0.1$, the aggressiveness to $\lambda = 1$ and the moving average and trend weights to $\alpha = 0.45$ and $\xi = 0.7$.

#### Traces, Testbeds and Evaluation Metrics

Table 5.1 depicts the datasets used for the evaluation. Instead of simple trivial traces (e.g., linear, sinusoidal loads), we have selected **seven publicly available real-world complex traces** to truly reveal the strengths and disadvantages of each algorithm. Figures [5.6a-5.6f] depict these traces. The experiments for the first four traces were run on a **Raspberry Pi** (model B) with 512MB of RAM and an ARM processor (single-core, 700MHz) while emulating the data load of each trace. The Raspberry Pi was selected as a suitable testbed, as it features similar limited processing capabilities of other "smart" devices (e.g., IoT home appliances). The Fitbit Step and Heart

[3] i-EWMA is the EWMA-based version of ASA introduced in the related work

| Trace Name | Origin | Description |
|---|---|---|
| Memory Trace | Cloud Server | A memory trace of 105 samples originating from a physical server running a java sorting benchmark [147]. |
| CPU Trace | Cloud Server | A CPU trace from a physical server of 800 samples from the Carnegie Mellon RainMon project [148]. |
| Disk IO Trace | Cloud Server | A Disk I/O trace from a physical server of 415 samples from the Carnegie Mellon RainMon project [148]. |
| TCP Trace | Internet Security Service | An incoming TCP network traffic trace from an internet security service of 500 samples from the port activity monitor of the Cyber Defense SANS Technology Institute [149]. |
| Step Trace | Wearable | A fitness step trace of 287 samples from a Fitbit Charge device [150]. |
| Heart Trace | Wearable | A fitness heartrate trace of 287 samples from a Fitbit Charge device [150]. |
| Calorie Trace | Wearable | A fitness calorie trace of 287 samples from a Fitbit Charge device [150]. |

*Table 5.1: Description of traces datasets used for performance and accuracy evaluation*

readings were fed, via SensorSimulator, to the **Android Wear** emulator hosting an app computing steps and heartrate measurements. The processing capabilities of the emulator are set to the specifications of a **Fitbit Charge** (single-core ARM 32MHz processor, 128MB Memory). We note that the Fitbit calorie trace was not fed to the Android Wear emulator, as explained shortly, calories are computed via human body indicators and heartrate measurements. Hence, this trace is used as ground truth to evaluate the techniques under comparison. We evaluate each technique towards their estimation accuracy and ability to efficiently use device resources. Also, our Fitbit data harvestor [4] is open-sourced and made available for anyone interested in extracting from Fitbit their own data in raw form. We evaluate each technique towards their estimation accuracy and ability to efficiently use the monitoring source underlying resources, as follows:

**Accuracy**: we evaluate an adaptive technique estimation accuracy by measuring the

[4] `https://github.com/dtrihinas/FitbitDataExtractor`

mean absolute percentage error (MAPE) from the original timeseries ground truth for each trace. Equation 5.14 depicts how the MAPE is calculated, where $A_i$ is the actual value for the $i^{th}$ datapoint and $E_i$ is the estimated value. For each adaptive technique, when a datapoint is not collected, $E_i$ is considered the last reported value.

$$MAPE_n = \frac{1}{n}\sum_{i=1}^{n}|\frac{A_i - E_i}{A_i}| \cdot 100\% \tag{5.14}$$

**Efficiency**: we evaluate efficiency by measuring the processing, network and energy overhead imposed to the device by each technique. In particular, we measure: (i) *CPU cycles* consumed to process the load imposed by each trace; (ii) *network overhead*, where we assume no aggregation technique is available and thus, a datapoint, if not filtered, is disseminated to the receiver-end; and (iii) *energy consumption*, based on the energy model for mobile embedded devices adopted from [75] and presented in Equation 5.15.

$$E = P_{idle} \cdot \tau_{idle} + P_{cpu} \cdot \tau_{cpu} + P_{io} \cdot \tau_{wait} + P_{net} \cdot \tau_{net} \tag{5.15}$$

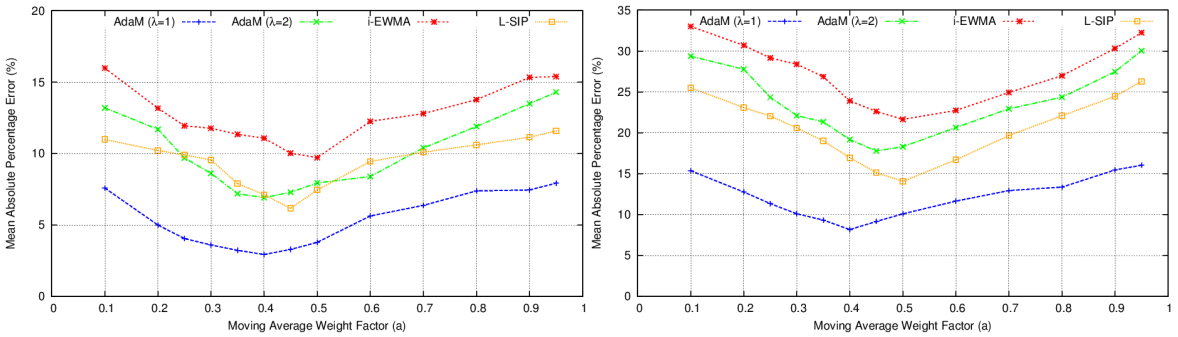
In this model, $P_{idle}$ denotes power in idle state; $P_{cpu}$ processor power (including L1 cache and memory); $\tau_{cpu}$ the CPU time; $P_{i/o}$ power for I/O; $\tau_{wait}$ the I/O time; $\tau_{net}$ the network transmission time; while $P_{net}$ power consumed for transmitting datapoints over the network. We also note that, processing measurements denoting the time spent in compute, i/o and data transmission were acquired with *perf*[5], while power level measurements were acquired by *powertop*[6] after obtaining a power profile (power in idle, active, transmission state) of the underlying device by *wattch* [151].

**Experiments**

At first, we compare AdaM sampling ($\lambda$=1, $\lambda$=2) to i-EWMA and L-SIP based on the MAPE evaluation metric. These three algorithms use moving averages in their estimation process. Therefore, MAPE is evaluated under different settings for the moving average parameter ($\alpha$) to find the best configuration. We set the minimum sampling period to 1 time interval, equal to the sampling period used to collect the trace ground truth, and set the maximum sampling period to 10 time intervals. We note that FAST is not presented in this test, as it does not use a moving average. Also,

[5] `https://perf.wiki.kernel.org/`

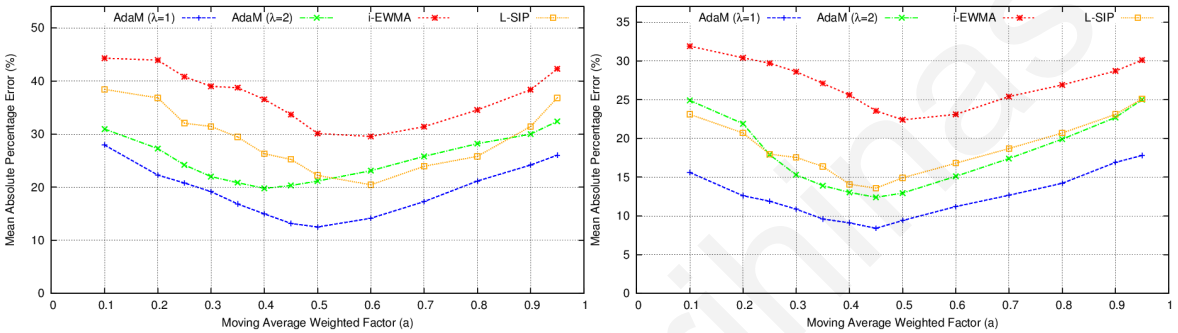
[6] `https://01.org/powertop`




*(a) Memory Trace*



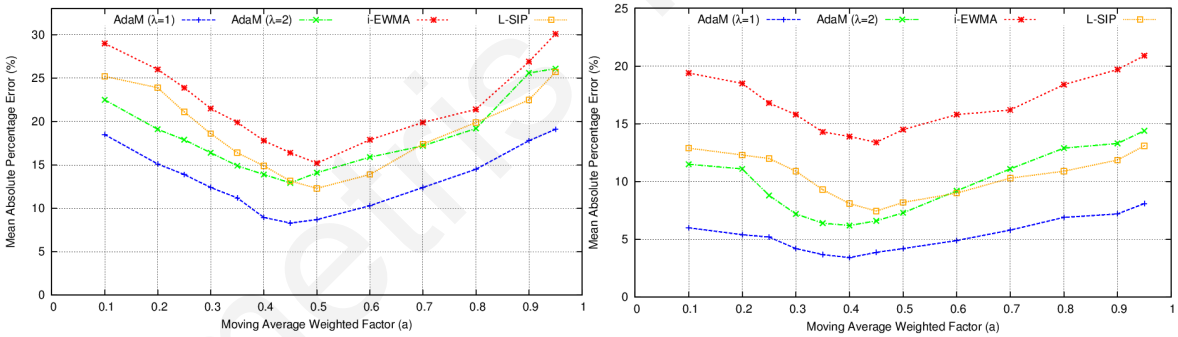

*(b) CPU Trace*




*(c) Disk I/O Trace*




*(d) TCP Trace*




*(e) Step Trace*




*(f) Heart Trace*

*Figure 5.3: MAPE comparison of techniques using moving average estimator*

FAST sampling is aggressive producing only a few sampling points and without applying a Kalman filter for smoothing, its MAPE is very high. Therefore, a test without the filter enabled, is meaningless. To be fair, we present its MAPE in subsequent comparisons while enabling filtering (see Fig. 5.10). Figures [5.3a-5.3f] depict the MAPE metric of each trace for the techniques under comparison.

First, we observe that AdaM ($\lambda$=1) for all traces features the lowest error. In its best setting, AdaM's MAPE is always under 10% except for the Disk trace, where it is

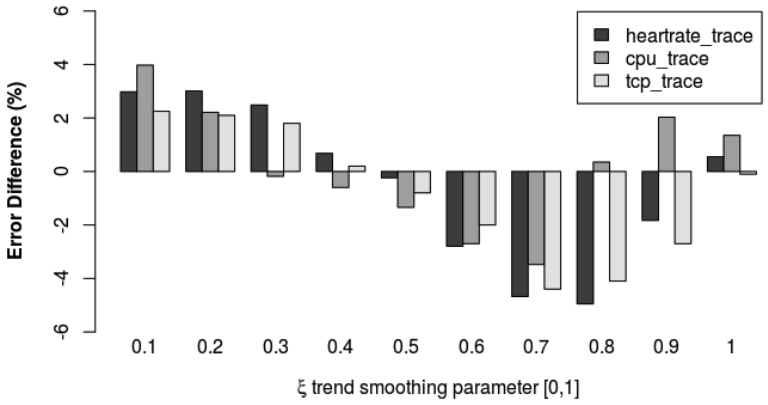



*Figure 5.4: Difference in MAPE when using trend in estimation*

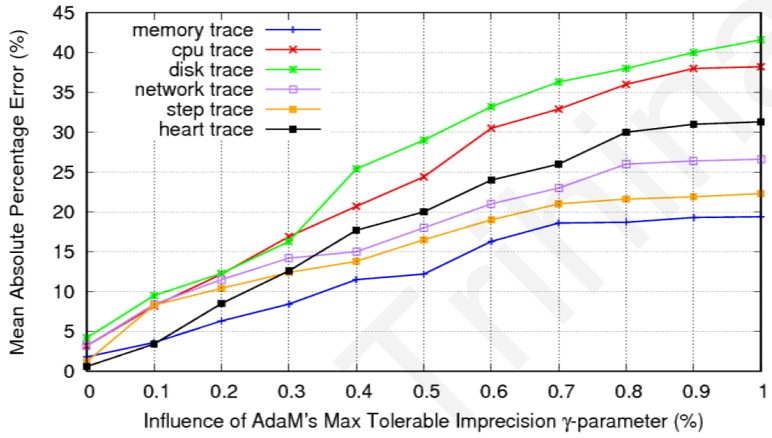
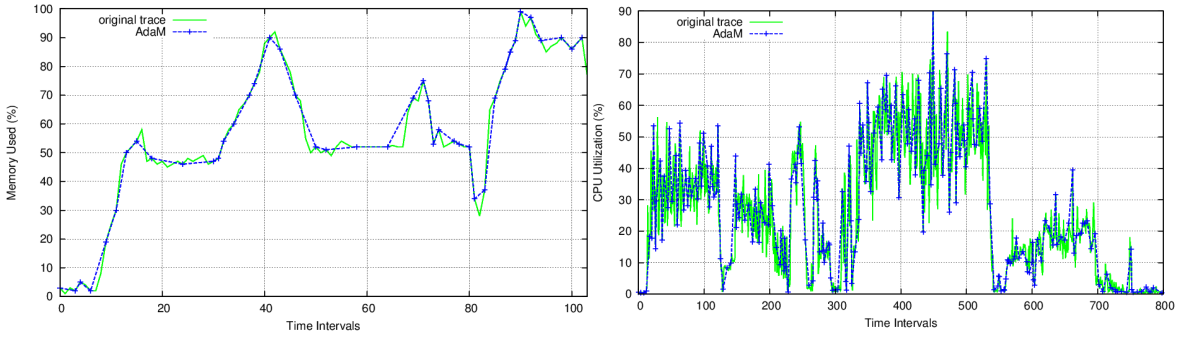



*Figure 5.5: Influence of max tolerable imprecision to estimation*

slightly above at 11%. Even, in a more aggressive setting ($\lambda$=2) AdaM still achieves a low error percentage and is comparable to L-SIP. AdaM sampling achieves a low MAPE due to the the adaptive weighting process which provides the estimation model with the ability to immediately detect abrupt transient changes in each trace. Moreover, due to the adaptive weighting, we observe that AdaM can take a wide range of values for the $\alpha$ parameter ($[0.3 - 0.6]$) with a deviation always under 3% from the best configuration. In turn, with the introduction of the trend component to the estimation model (eq. 5.7), AdaM accuracy is improved by shedding 2-5% of the error. This is depicted in Figure 5.4, where for a wide range of settings for the trend parameter, denoted as $\xi$, the error is reduced compared to not considering trends at all (Fig. 5.4 depicts only the three most challenging traces). Thus, *the analysis conducted shows that profiling to find optimal parameter settings for the estimation process incorporated in AdaM is not always required if slight imprecision is acceptable.*

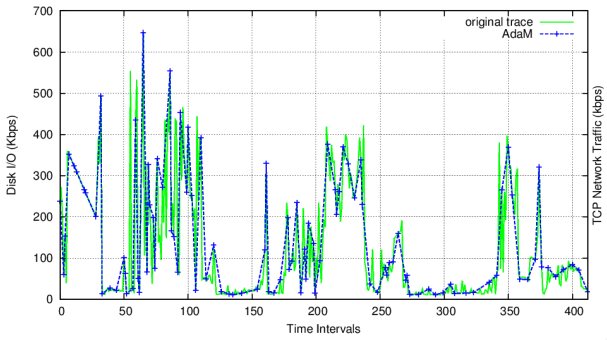
Next, we evaluate the influence of the maximum tolerable impression ($\gamma$), the only parameter that must be set by the user, to the overall estimation error (MAPE).



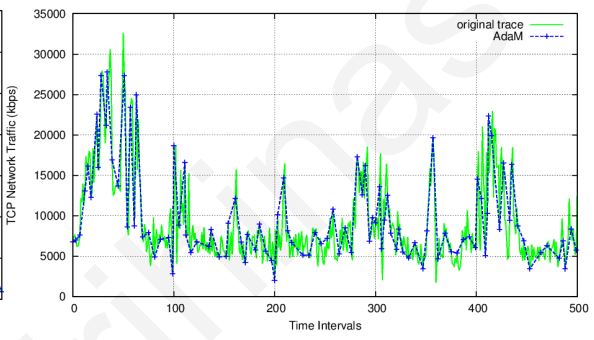

*(a) Memory Trace*



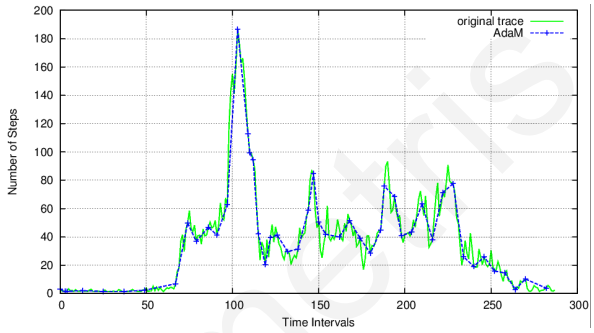

*(b) CPU Trace*



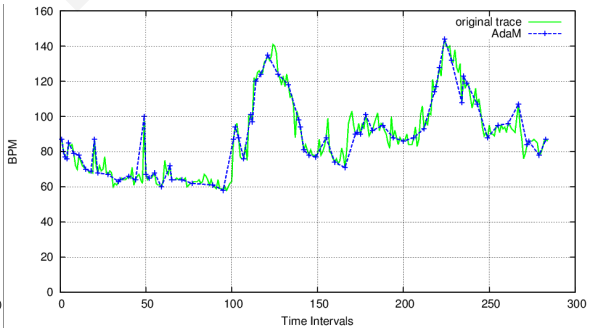

*(c) Disk I/O Trace*




*(d) TCP Trace*




*(e) Step Trace*




*(f) Heart Trace*

*Figure 5.6: Comparison of traces generated via AdaM towards the original traces*

Figure 5.5 depicts the evaluation conducted, where in all imprecision configurations, and for all traces, AdaM's MAPE is well below the acceptable imprecision threshold (MAPE $< \gamma$), thus highlighting the importance of the *confidence metric*, even for $\gamma$-values which indicate a high imprecision tolerance. Furthermore, Figure 5.6 depicts AdaM compared to the original traces, where we observe that *AdaM always follows the data evolution even in highly abrupt and fluctuating phases*.

At this point, we compare all algorithms efficiency based on the overhead im-

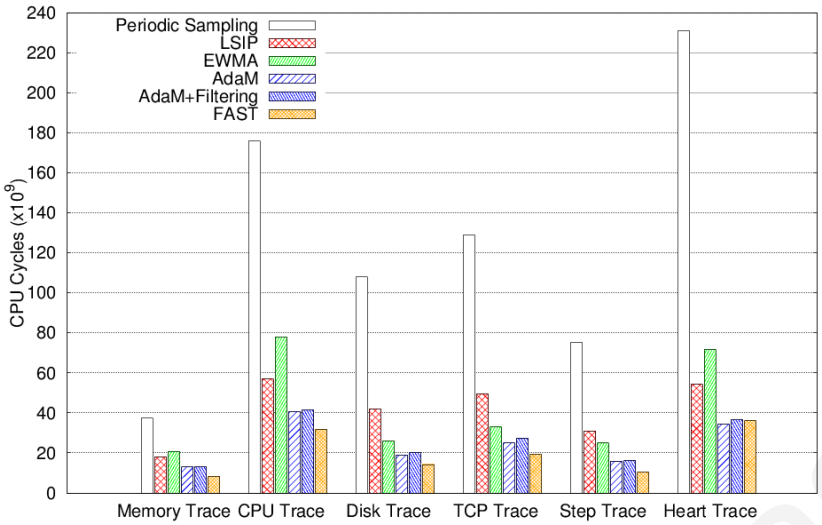



*Figure 5.7: Compute overhead comparison*

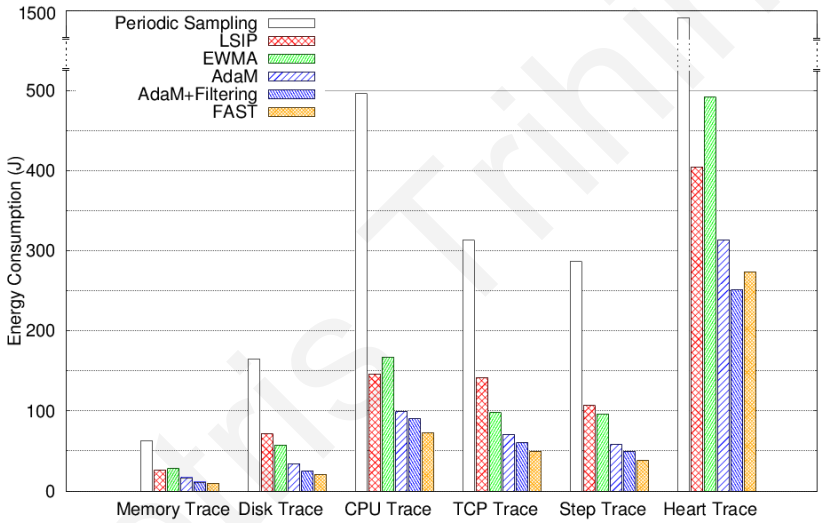



*Figure 5.8: Energy consumption comparison*

posed to the monitoring source. We note that for all subsequent experimentations AdaM incorporates trend detection in its estimation process with $\xi$ left to the default setting. In this test we include FAST, as well as, AdaM with filtering ($R \in [0, 2]$). At first, we observe (Fig. 5.7) that filtering does not impose additional overhead to AdaM as the overhead in all cases is well under 1%, while the gains from reduced network traffic (Fig. 5.9) are significant as an average reduction of 74% is achieved. Nonetheless, with adaptive filtering, AdaM yields a slightly increased error when compared to only enabling adaptive sampling (Fig. 5.10). However, as the user-defined inaccuracy budget ($\gamma$) permits further approximation, accuracy is slightly sacrificed. This improves device efficiency by significantly reducing the network overhead, and consequently energy consumption (Fig. 5.8), with a slightly

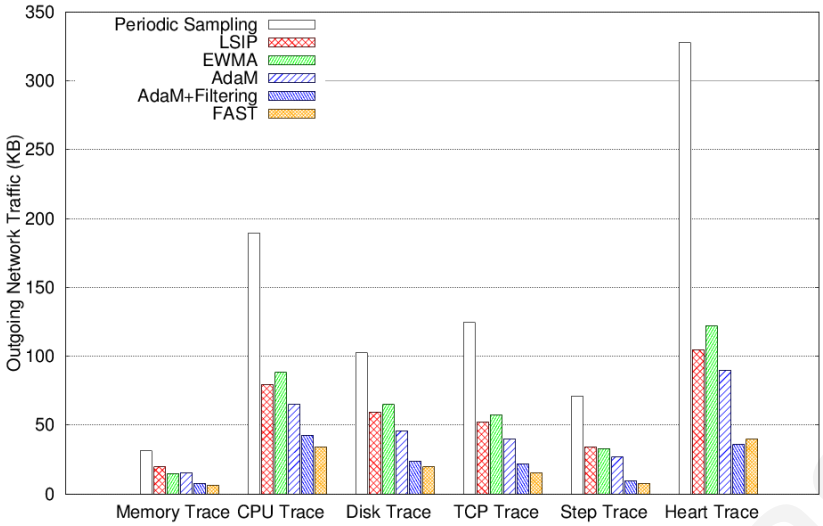



*Figure 5.9: Outgoing network traffic comparison*

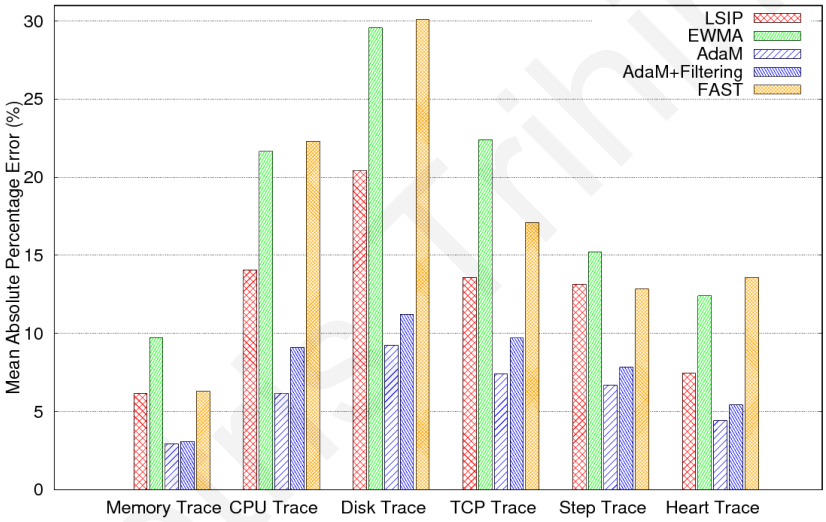



*Figure 5.10: Mean Absolute Percentage Error (MAPE) comparison*

increased MAPE which is never increased more than 3% (disk trace). It should also be mentioned that the selected filtering range ($R \in [0, 2]$) is not the optimal setting ($R^* \in [0, 1.2]$) for the testbed inaccuracy budget ($\gamma$=0.1). In general, when comparing to periodic sampling, *AdaM succeeds in reducing data volume by 74%, energy consumption by at least 71%, while accuracy is, in all cases, greater than 92% and with filtering, greater than 89%*. Most importantly, *in the case of biosignal monitoring where costly signal analysis is performed energy consumption is reduced by more than 86% (Fig. 5.8).*

Moreover, as with initial MAPE comparison, AdaM outperforms i-EWMA and L-SIP. AdaM is able to achieve this due to its low complexity and the introduction of the confidence metric which supports the estimation process to select the appropriate $T$. Nonetheless, its overhead is slightly larger in some traces (e.g., cpu trace) than the

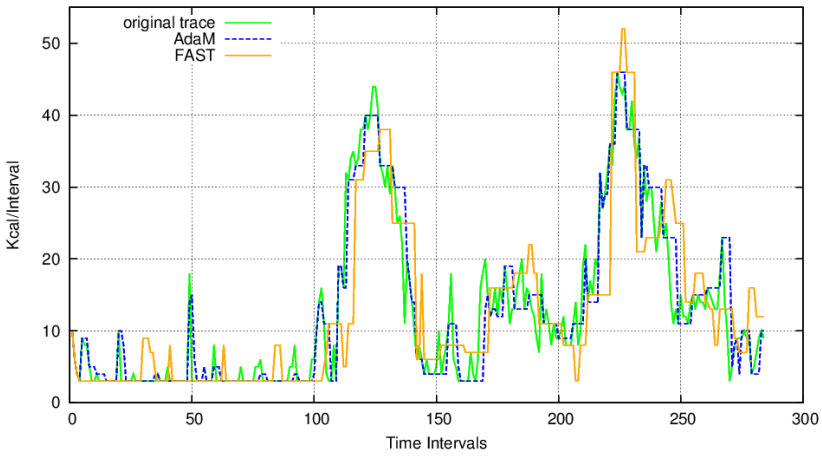



*Figure 5.11: Wearable device calories comparison*

FAST algorithm. FAST's aggressiveness, which computes larger sampling periods, results to slightly lower energy consumption and network traffic. However, this does not come for free. In Figure 5.10, we observe that for FAST to achieve this, significant accuracy is sacrificed, especially for traces featuring limited linearity (e.g., CPU, disk trace) in contrast to *AdaM which features a low-cost approximate and adaptive estimation model capable of achieving a balance between efficiency and accuracy*.

To illustrate the importance of maintaining accuracy, especially in the case of wearable devices, we compute calorie consumption where no further external stimulus is required. Energy expenditure (calories/min) is an algorithmic process based on human body indicators (age, weight, height) and heartrate monitoring[7] [152]. Figure 5.11 depicts the initial calorie trace provided by a Fitbit Charge device and the traces computed by AdaM and FAST. We observe that AdaM provides a better estimation than FAST with AdaM's error growing from 6.42% in heartrate monitoring to 9.07% in calorie counting, while FAST's error increases from 13.61% to 21.83%. To grasp on the gains of sacrificing 9% accuracy when embedding AdaM to a wearable, we perform a battery life expediency test. Specifically, we first calculate the average hourly device consumption in milliamps (mA), denoted as $I_{DC}$, from the temporal energy consumption ($E_i$) and the voltage[8] ($V_i$) driving the device (eq. 5.16). Battery life, denoted as $BL$, is then computed from battery capacity ($B_C$), and device consumption ($I_{DC}$). Battery capacity is set to the capacity of a Fitbit Charge (35mAh) and the multiplicity factor $\beta$ is set to 0.7 which is industry standard practice to account

[7] Fitbit/Garmin also perform additional server-side fine-tuning to produce better calorie estimates based on other parameters (e.g., exercise type, type of food consumed)

[8] In idle state voltage is $1.7V$ but when driving peripherals (e.g. accelerometer) it scales up to $3.3V$

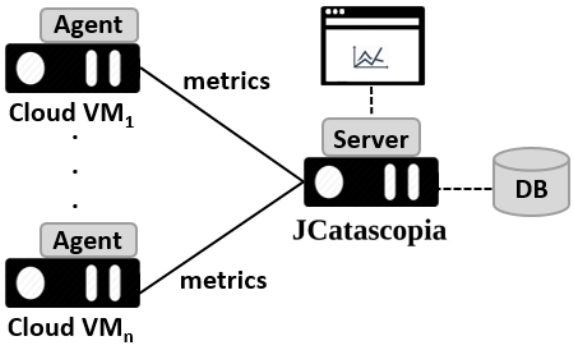



*Figure 5.12: Cloud monitoring topology*

for external factors affecting battery runtime (e.g., temperature).

$$I_{DC} = \frac{1}{n} \cdot \sum_{i}^{n} \frac{E_i \cdot \tau_i}{V_i} \tag{5.16}$$

$$BL = \beta \cdot \frac{B_C}{I_{DC}} \tag{5.17}$$

From our estimations, a wearable device without AdaM has a device consumption of 0.239mA and battery life expediency, on average, of 4.26 days (between 3-5 days, as claimed by the wearable designers). However, *a device with AdaM embedded in its software core is able to reduce consumption by 0.094mA and increase battery life from 4 days to an additional 2.94 days thus expanding battery life from 3-5 days to 6-7 days on a wearable device offering step, calorie and heartrate monitoring*.

### 5.7.2 Big Data Streaming Service Scalability Evaluation

In the next set of experiments, we intend to showcase the benefits of integrating AdaM to edge nodes of two different big data streaming services in regards to the velocity of which data is generated and scalability of the overall system.

#### Cloud Monitoring

In this case we study how cloud monitoring benefits by embedding AdaM in monitored VMs to reduce data velocity and ease monitoring server processing on an Openstack private cloud deployment. To show this we take advantage of the open-source JCatascopia monitoring framework [22] which offers integration endpoints for adaptive algorithms and is capable of running on linux-based IoT devices (e.g., Raspberry Pi). We embed AdaM in the source code of JCatascopia-Agents (metric

collectors), such that they are capable of adapting the sampling rate and the metric filter range. Hence, JCatascopia-Agents upon initialization, randomly select 1 of the 4 server traces (Memory, CPU, Disk I/O and Network) introduced earlier to emulate the behavior of a Raspberry Pi. For each trace we set the minimum sampling period to 500$ms$, in order to generate a high volume of data. We use these traces, and not random collected data, as we have confirmed from our evaluation that AdaM can reconstruct each of the available traces with high accuracy.

Figure 5.12 depicts the topology of a JCatascopia cloud deployment. Initially the deployment is comprised of 1 JCatascopia-Agent and every 5 minutes a new agent is instantiated and added to the deployment until we reach a capacity of 80. Collected metrics are disseminated from Agents to a JCatascopia-Server (4VCPU, 4GB RAM) where they are processed and stored to the monitoring database. We evaluate data velocity by measuring *archiving time*, which is the average time required by the JCatascopia-Server to process and store a received metric. We use this topology to compare AdaM ($\gamma = 0.1, T \in [1, 10], R \in [0, 2]$) against (i) using periodic sampling ($T = 500ms$) for metric collection; and (ii) using periodic sampling ($T = 500ms$) along with filtering on the JCatascopia-Server (we show results for $R = 1\%$ which was the best filtering configuration).

Figure 5.13 depicts the results of our comparison, where we observe that without any adaptive techniques data velocity follows an exponential growth. In turn, when filtering is added to the JCatascopia-Server, thus on the server side, archiving time is comparable to AdaM but only up to 30-35 VMs. After that, archiving time increases exponentially as the per-message dissemination and storage overhead is the actual bottleneck for cloud monitoring services. However, *if data sources utilize AdaM's adaptive capabilities on the edge, data velocity is reduced and a more linear growth is achieved*.

**Intelligent Transportation System (ITS)**

Next, we present a thorough evaluation of an ITS with AdaM embedded in the edge nodes of the overall system. The ITS topology is depicted in Figure 5.14 and is a (simplified) replica of the Dublin Smart City Bus Network [153] comprised of:

- 1000 buses with GPS tracking devices sending periodic updates to the ITS. Each update reports 16 metrics including: location, bus id and area of coverage.

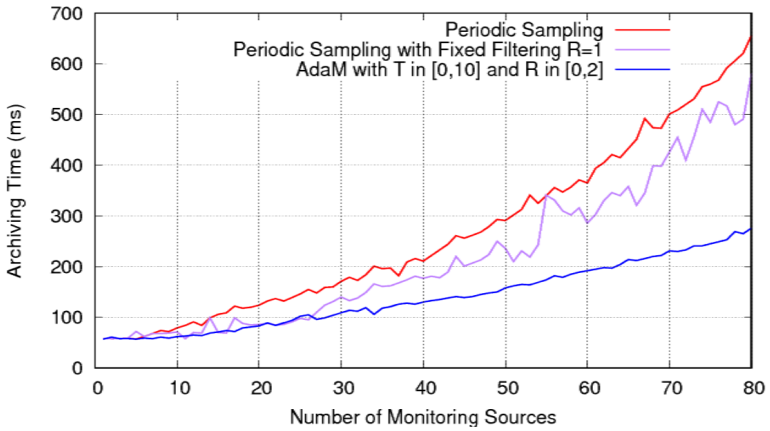



*Figure 5.13: Cloud monitoring scalability evaluation*

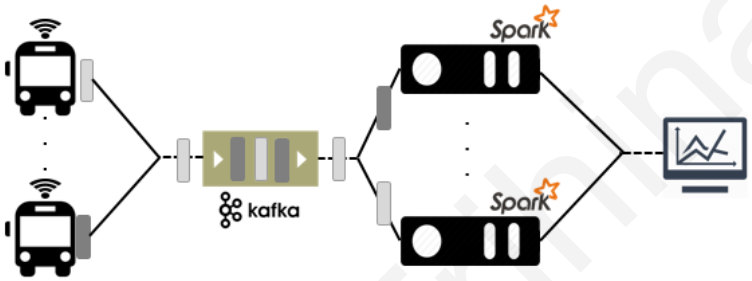



*Figure 5.14: ITS topology*

Most importantly, in each update is an estimate of the current route delay (how many seconds off schedule is the bus);

- An Apache Kafka queueing service for bus updates to be dequeued by the ITS processing engine. The Kafka instance resides on a x-large VM (16VCPU, 16GB RAM). The queueing service is also used by the ITS processing engine to enqueue results for the dashboard;
- The distributed processing engine powered by Apache Spark to process bus updates. The Apache Spark cluster is comprised of 5 large worker nodes (8 VCPU, 8GB RAM, 40GB Disk) with a batch window of 1 second;
- A dashboard used by ITS operators to view the results.

We have created a Bus Emulator[9] to emulate the tracking behavior of Dublin buses. Each bus instance initially receives a busID and from there on, it emulates the behavior of the specific bus by sending updates to the ITS based on publically available and real data collected from 1000 Dublin buses for an entire month (Jan 2014).The ITS processing engine pulls data from the queue service every time interval

[9] `https://github.com/dtrihinas/JobEmulator`

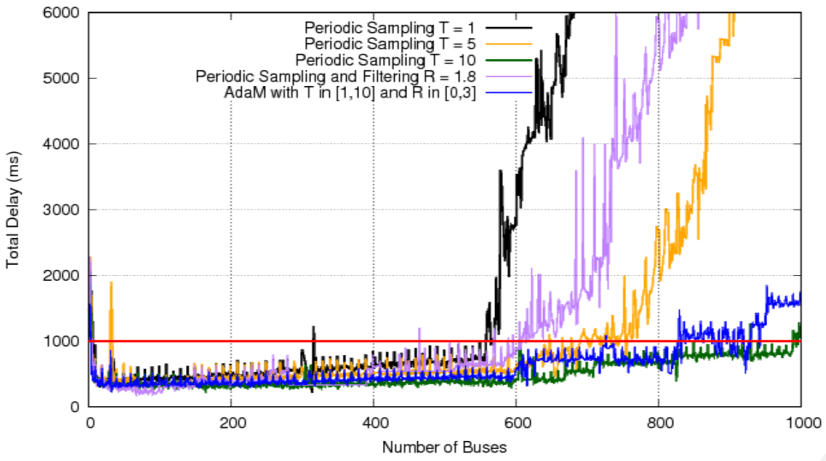



*Figure 5.15: Apache spark streaming total delay*

and processes bus updates. With processing we denote the intensive task of checking per bus if the current route delay is over one standard deviation from its average delay in the current city block based on a weekly sliding window. If the delay is lower than this threshold, the message is discarded. If not, the processing engine increases an aggregator counting the number of buses with delays in the area and signals a warning if more than 10 violations are detected in a 5 min sliding window. Hence, ITS operators always receive timely warnings via the bus network to further investigate and take action or not.

For our experiments, the topology is initially configured to host 50 buses and every 10 minutes 50 more buses are added to the network until we reach a topology with 1000 buses. First, we set the sampling period of each bus to 1 time interval which is the dataset ground truth. In the second and third test the sampling period is set to 5 and 10 intervals respectively. Afterwards, we embed AdaM to each bus emulator ($\gamma = 0.15$, $\lambda = 1$, $T \in [1, 10]$, $R \in [0, 3]$). We also evaluate the ITS service with T=1 along with filtering made available on Apache Spark (results shown for $R = 1.8$ which was the best configuration after testing). We perform this test to show that even if filtering is enabled at the cloud-side, the scheduling time in distributed data engines is a cost that should not be ignored. Thus, we monitor the *total delay* imposed to the Apache Spark cluster, which includes both *processing* and *scheduling time*. Processing time denotes the time required to parse a batch of updates, while scheduling time denotes the time from which a batch is dequeued up to the time it starts being processed. In order for a Spark cluster to be considered as stable, the total delay must be comparable to the batch window. Otherwise, if the delay is

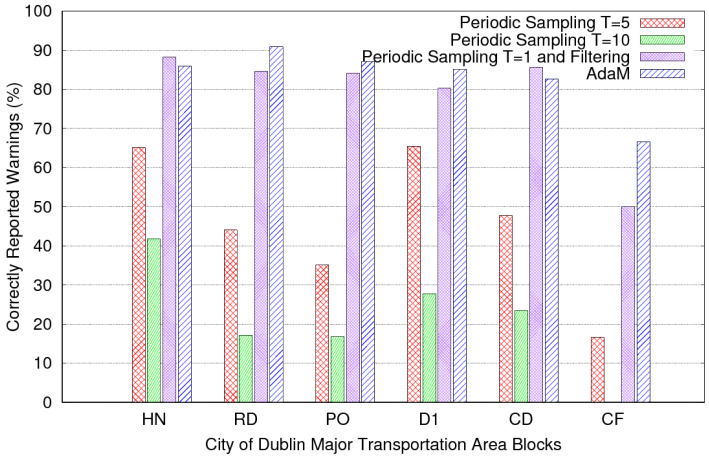



*Figure 5.16: Percentage of correctly reported warnings per area*

continuously increasing, batches are queued and not processed immediately as the system is unable to keep up. Therefore, the system is characterized as unstable [154].

Figure 5.15 depicts the total delay metric as the number of buses and message velocity increase. We observe that for T=1 (ground truth periodicity), the system becomes unstable after 572 buses. As the total delay grows exponentially, the system becomes unstable and we terminate the experiment run. Similarly, for T=1 with filtering enabled and with sampling period set to T=5, the system after 632 and 746 buses respectively is unstable again. Remarkably, for T=10, the maximum number of buses is reached with the system slightly rising above the threshold at 971 buses. On the other hand, with AdaM embedded in each bus emulator we observe that at 817 buses the threshold is violated for the first time, however the maximum number of buses is achieved without the need to terminate the deployment. At this point one may argue that a solution to the high data influx is auto-scaling where the ITS service dynamically scales by provisioning additional resources (e.g., Spark nodes) to cope with the overwhelming load [22]. However, auto-scaling should be used wisely as scaling large data-intensive infrastructures, other than monetizing costs for resources, involves hidden costs such as the time cost for data (re-)balancing, replication, and the additional network overhead [104] [155]. *Hence, if auto-scaling can be avoided or postponed for latter in favor of a smaller cluster achieving the same throughput, then it should.*

After evaluating the total delay, we proceed to an accuracy evaluation by comparing the correct and false reported warnings per Dublin area by each experiment run to the ground truth. From Figures 5.15-5.17 we observe that although a T=10

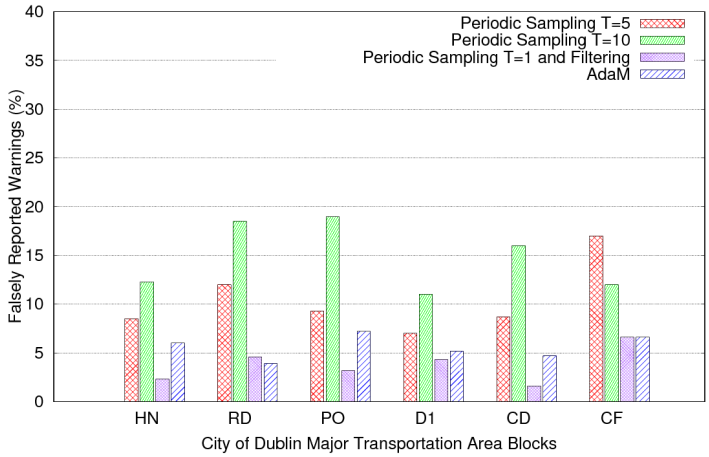



*Figure 5.17: Percentage of falsely reported warnings per area*

sampling period features the lowest streaming delay, the percentage of correctly reported warnings merely spans from 0% to 38%, while the false alarm percentage spans from 11% to 19%. For T=5, the percentage of correctly reported warnings spans from 17% to 63%, while the false alarm percentage spans from 7% to 17%. On the other hand, AdaM achieves a percentage of correctly identified warnings of at least 87% in all Dublin areas except for the CF city block where it correctly detects 8 in a total of only 12 actual warnings. Remarkably, AdaM, with a significantly lower streaming delay, is even comparable to using T=1 with filtering enabled at the cloud-side. In turn, AdaM's percentage of falsely reported warnings is significantly lower than utilizing a fixed periodicity as its estimation mechanism is able to quickly detect and adapt the periodicity to follow abrupt and transient fluctuations in a bus monitoring stream. Hence, *AdaM is able to adapt, at the edge, the monitoring intensity while satisfying the accuracy guarantees set upon initialization*.

In turn, Figure 5.18 depicts the average occurrence of each sampling period per hour of a weekday for buses servicing the Central Dublin area (day buses operate between 6am-11.59pm). We observe that for hours with high traffic such as morning rush hours (e.g., 7-9am), lunch time (e.g., 12-13pm) and afternoon rush hours (e.g., 17-19pm), high sampling rates (T = 1-2s) are preferred, while the rest of the day low sampling rates are preferred. However, one must note the variety of sampling rates used throughout each hour of the day justifying the need for adaptive monitoring. Hence, *AdaM is able to adapt in place and inexpensively the monitoring intensity, thus reducing the volume and velocity of incoming data to distributed big data streaming services while preserving accuracy*.

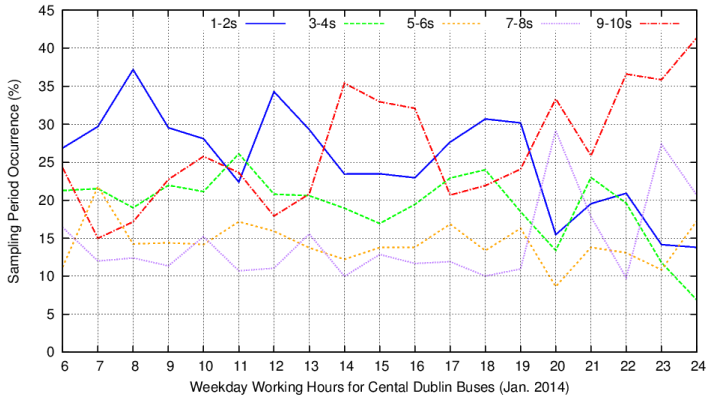



*Figure 5.18: Sampling periods used by buses in the central dublin area per hour of the day*

**Chapter Summary.** This chapter introduces AdaM, a software library offering low-cost approximate and adaptive monitoring for software agents and IoT devices. AdaM uses probabilistic learning by incorporating in its software core a probabilistic exponential weighted moving average (PEWMA) which captures the metric stream runtime evolution and variability. To immediately identify abrupt and transient changes in the metric stream evolution and overcome any lagging effects in the estimation process, AdaM uses adaptive parameter weighting and trend detection to fine-tune the algorithmic learning model at runtime. AdaM dynamically adjusts the metric collection periodicity and filtering range based on the estimation model "confidence" which is an (error) evaluation metric denoting the ability of the model to correctly estimate what will happen next in the metric stream. This supports our framework to "reward" larger property adjustments (e.g., increase periodicity) when estimations fall within the accuracy intervals given by the user or rollback to a fixed approach when satisfactory estimations cannot be made. Thus, if a "confident" estimation cannot be made, AdaM will rollback to a fixed approach (e.g., periodic sampling) to ensure accuracy guarantees are obeyed over efficiency. After performing a thorough experimentation study using real-world data from cloud applications, internet security services, wearables, and intelligent transportation services, results show that AdaM can significantly reduce energy consumption by at least 83% and the volume of generated data by at least 71%, while maintaining accuracy always above 89%. Thus, big data services consuming IoT data can truly benefit in terms of lower monitoring costs, achieve greater scalability and efficiently utilize underlying resources when embedding AdaM in the software core of their monitoring sources.

# Chapter 6

# ADMin: A Plugin for Model-Based Adaptive Dissemination

## 6.1 Overview

In this chapter we introduce ADMin, a plugin developed to extend the functionality of the AdaM framework by offering model-based adaptive dissemination to efficiently adapt, in place, the rate at which monitoring sources (e.g., IoT devices) disseminate metrics to receiving entities based on the evolution and variability of the metric stream. In particular, when the values of a metric stream can be inferred by an estimation model, instead of sending these values, ADMin favors disseminating updates referring to the estimation model. From the estimation model, metric receiving entities can then infer any missing values. In turn, dissemination to metric receivers is withheld by monitoring sources and enabled only when shifts are detected, thus, rendering the current model as no longer able to describe the metric stream evolution within the confidence guarantees given by the user. This significantly reduces the communication overhead imposed directly to a monitoring source (e.g., energy consumed by IoT device) and indirectly to a network of monitoring sources (e.g., IoT network data overwhelm), by suppressing from dissemination consecutive datapoints with "little" change in their metric values that can be inferred by the estimation model.

Figure 6.1 depicts a high-level abstract overview of the ADMin plugin integrated in the AdaM framework, where focus is drawn on the flow describing how model-based adaptive dissemination is achieved. In particular, ADMin adopts the low-cost approximate stream estimation model of the AdaM framework used to capture the

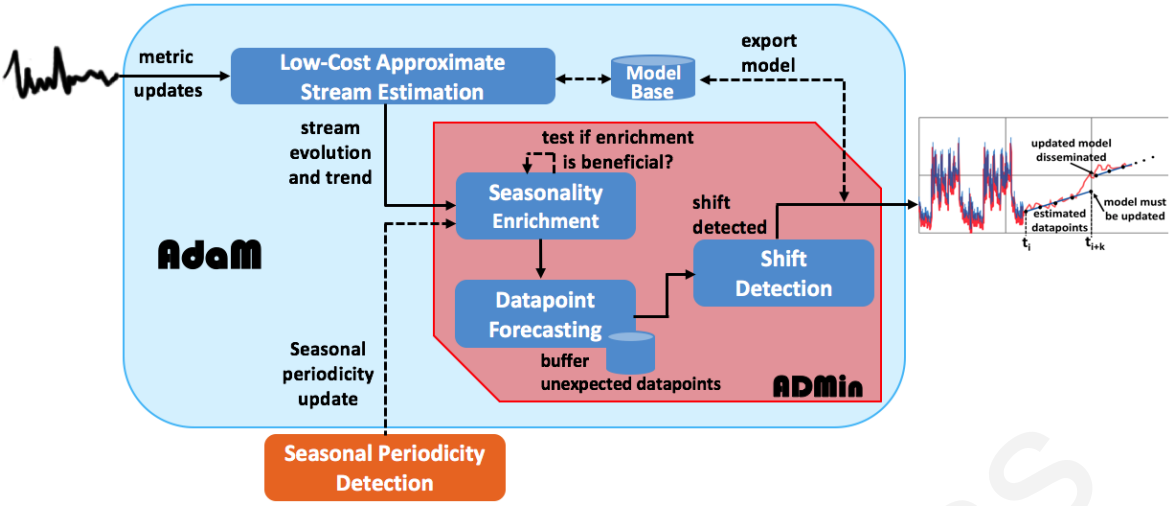



*Figure 6.1: ADMin plugin for AdaM framework*

runtime evolution and trend of the monitoring stream and extends it to also include a seasonal component. The module in charge of this task is the *Seasonality Enrichment* module, which detects if seasonality enrichment provides a more accurate estimation of the metric stream evolution via online statistical testing. If so, it enriches the model with seasonality knowledge; otherwise, the estimation will roll-back to the model's previous estimation state. Although seasonality enrichment is optional for the AdaM framework, as shown in Section 6.4.2, if a metric stream exhibits such behavior, the *estimation error* and *shift detection delay* can be significantly reduced. However, detecting the optimal *seasonal periodicity* is a complex problem by itself [140]. Thus, the user may enable ADMin to integrate and utilize the lightweight tensor-based and parameter-free ComCube framework to determine the near-optimal seasonal periodicity [141].

After updating the estimation model, the *Datapoint Inference* module will then label the current datapoint as "expected" or "unexpected". If the datapoint is "expected", meaning it can be inferred by the forecasting function of the estimation model within the confidence intervals given by the user, it is suppressed; otherwise, the datapoint is locally stored and will be disseminated when dissemination is triggered as such datapoints may be of interest. Next, the *Runtime Shift Detection* module determines if there is a shift in the metric stream evolution rendering the estimation model as inconsistent or if the local storage has reached maximum capacity. If so, a compressed message containing an updated version of the estimation model and the contents of the local storage is passed to the *Network Unit* of the IoT device to

**Algorithm 8** ADMin: Low-Cost Approximate Estimation Model

**Input:** datapoint $d(t_i, v_i)$, confidence $\delta$ given by user and local storage $buf$

**Output:** updated estimation model

**Ensure:** $\mu_i, \sigma_i, x_i, s_i$ are initialized and $k \leftarrow 0$ after dissemination

*compute p– and z–value*

1: $P_i, Z_i \leftarrow \text{probDistro}(v_i, \hat{v}_i, \sigma_i)$ (eq. 5.5)

*update estimation model*

2: $\mu_i, \sigma_i \leftarrow \text{updPEWMAwithTrendandSeason}(P_i, v_i, x_i, s_i)$ (eq. 6.3)

3: $x_i \leftarrow \text{updTrend}(\mu_i)$

4: $s_i \leftarrow \text{updSeasonality}(\mu_i, x_i, L)$

*if datapoint is unexpected then store in buffer*

5: **if** $\text{isDatapointUnexpected}(v_i, P_i, Z_i)$ **then**

6: $\quad buf \leftarrow v_i$

7: **end if**

8: $y_1 \leftarrow \mu_i + kX_i$ (eq. 6.4)

9: $y_2 \leftarrow \mu_i + kX_i + S_i$

*is seasonality enrichment beneficial?*

10: **if** $\text{T–Test}(\delta, \sigma_i, y_1) > \text{T–Test}(\delta, \sigma_i, y_2)$ **then**

11: $\quad \hat{v}_{i+1} \leftarrow y_1$

12: **else**

13: $\quad \hat{v}_{i+1} \leftarrow y_2$

14: **end if**

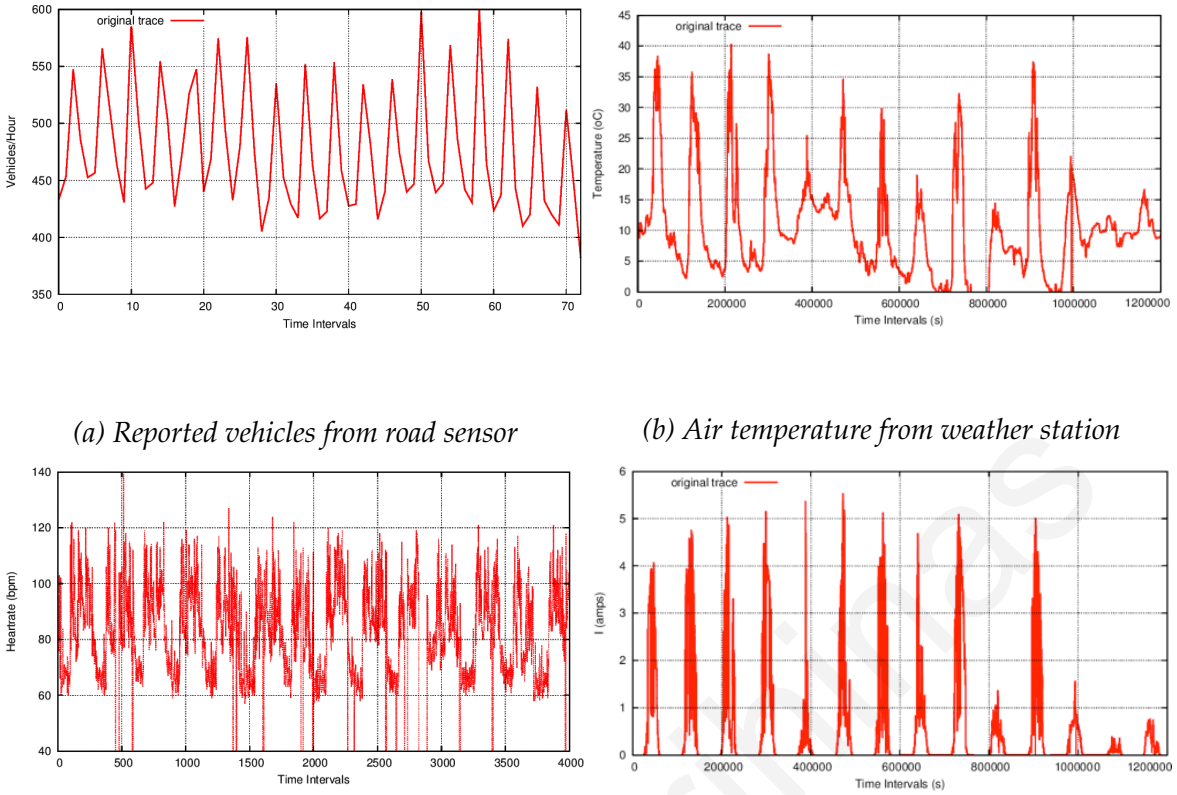
15: $k \leftarrow k + 1$

16: **return** $\text{estModel}(\hat{v}_{i+1}, \mu_i, \sigma_i, x_i, s_i)$

be disseminated to interested receivers. Otherwise, monitoring dissemination is suppressed with the Network Unit remaining in an idle state.

## 6.2 Seasonality Enrichment and Datapoint Inference

Seasonality is defined as the tendency of a series of data, in our case a metric stream, to exhibit behavior that repeats itself every $L$ periods (e.g., hourly, daily) [140]. It is, thus, an indicator of a pattern of regular periodic and predictable fluctuations in a metric stream [156]. IoT data, such as human body indicators and environmental




*(a) Reported vehicles from road sensor*

*(b) Air temperature from weather station*

*(c) Human heartrate from wearable*

*(d) Current production from photovoltaic panel*

*Figure 6.2: Data exhibiting seasonality effects*

data, present seasonality behavior [126]. In particular, and as depicted in Figure 6.2, environmental data such as temperature, humidity and air quality, gradually change from their previous timestamped values and exhibit predictable behavior when following climate cycles. In turn, human activity and (self-)tracking data exhibit seasonality due to repeatable patterns of human social activity and habits (e.g., time we rest, go to work, exercise, etc.). Additional cases which exhibit seasonal behavior include, but are not limited to, smart metering and quality control in manufacturing, home appliances and electricity distribution.

Algorithm 8 depicts how the low-cost approximate estimation model of the AdaM framework can be updated to include a seasonal component and datapoint inference without any alterations required to the other adaptive monitoring techniques comprising the framework. In this context, the seasonality contribution, denoted as $s_i$, is computed by adopting the Holt-Winter's method to update the seasonality contribution of a moving average at runtime by applying exponential smoothing across seasons in addition to smoothing over consecutive datapoints of the moving average

evolution and trend [157]. Equation 6.1 and 6.2 depict how the seasonal contribution $s_i$ is updated at each seasonal cycle with $\omega$ denoting a smoothing weight in the range $[0, 1]$ with values near 1 showing a preference to favor recent seasonal effects and with a zero value converging to the weighted moving average $\mu_i$. In turn, $x_i$ denotes the trend component.

$$s_i = \begin{cases} 0, & i < L \\ \omega\,(v_i - \mu_i - x_i) + (1-\omega)\,s_{i-L}, & i > L \end{cases} \tag{6.1}$$

The reason for introducing two equations for $s_i$ is because the seasonal component may take two forms. Specifically, it can be *additive* (Equations 6.1 and 6.4), indicating that the seasonal component is added with the trend component to bring the (moving) average to the appropriate value-base in the current seasonal cycle, or *multiplicative* (Equations 6.2 and 6.4), in which the seasonal component is proportional to the underlying trend.

$$s_i = \begin{cases} 0, & i < L \\ \omega\,(\frac{v_i}{\mu_i - x_i}) + (1-\omega)\,s_{i-L}, & i > L \end{cases} \tag{6.2}$$

The distinguishing characteristic between the two forms is that in the additive case, the series shows seasonal fluctuations regardless of the overall level of the metric stream, while, in the multiplicative case, the amplitude of the seasonal contribution depends on the overall upward/downward level of the metric stream. When in doubt, the additive model can be used, although knowledge of a multiplicative component can contribute in reducing time lagging effects in the estimation process.

$$\begin{aligned} \text{Level:} \quad & \mu_i = \tilde{a}_i\,(\mu_{i-1} + x_{i-1}) + (1 - \tilde{a}_i)(v_i - s_{i-L}) \\ \text{Trend:} \quad & x_i = \xi(\mu_i - \mu_{i-1}) + (1-\xi)x_{i-1} \\ \text{Seasonality:} \quad & s_i = \omega(v_i - \mu_i - x_i) + (1-\omega)s_{i-L} \end{aligned} \tag{6.3}$$

With the addition of the seasonal component, the estimation model, introduced in Section 5.4, comprised of the PEWMA moving average and trend can now be entirely expressed with Equation 6.3. With the model updated iteratively and with only previous value knowledge, the metric stream can now be approximated by using Equation 6.4 to infer the values for subsequent $k$-datapoints.

$$\begin{aligned} \text{Additive:} \quad & \hat{v}_{i+k|i} = \mu_i + k\,x_i + s_{i-L+k_L^+} \\ \text{Multiplicative:} \quad & \hat{v}_{i+k|i} = (\mu_i + k\,x_i) \cdot s_{i-L+k_L^+} \end{aligned} \tag{6.4}$$

**Algorithm 9** ADMin: Datapoint Expectency

**Input:** datapoint $d(t_i, v_i)$, z-value based on confidence $\delta$ given by user

**Output:** datapoint expectency

1: $z \leftarrow z\_score(\delta)$
2: **if** $(v_i > \mu_i - z \cdot \sigma_i)$ && $(v_i < \mu_i + z \cdot \sigma_i)$ **then**
3: *label* ← `true` //datapoint within confidence intervals
4: **else**
5: *label* ← `false`
6: **end if**
7: **return:** *label*

where $k_L^+ = \lfloor (k-1) \mod L \rfloor + 1$, denotes that the estimate for the seasonal component used in this time interval was last updated $L$ time intervals ago.

To reduce the volume of disseminated data, ADMin will suppress "expected" datapoints, as depicted in Algorithm 9. This means that any datapoints that can be approximated by the estimation model within the confidence intervals given by the user will not be disseminated to interested metric receivers. Rather, metric stream receiving entities can use the model to infer (subsequent) datapoint values that are not disseminated. Intuitively, the more consecutive datapoints that can be approximated by the model the larger the compression of the metric stream will be. Nonetheless, uncertainties in the form of anomalies can be introduced when sensing the physical world. Thus, "unexpected" datapoints are locally stored and disseminated when dissemination is triggered[1].

However, in real-life systems perfect seasonal behavior is rarely observed. Rather, seasonality behavior is exhibited although irregularities in the form of noise, often not of fixed length, are introduced in the seasonal cycle [126]. This is particularly, evident in Figures 6.2b-d, where considering prior knowledge may result in over-estimating subsequent $v_i$'s, if the current metric stream evolution completely differs from the seasonal behavior [140]. For example, let us consider the production of a PV, depicted in Figure 6.3, and as a representative seasonal contribution the previous day hourly average ($S_{i-L}$). This will lead to overestimating the production of PV panel when the current day weather significantly differs from the previous day. To

[1] Anomalies can be a sign of quality degradation for an IoT device

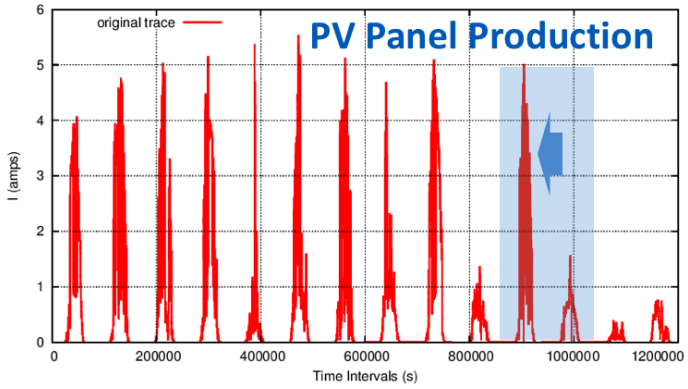



*Figure 6.3: Overestimating seasonal contribution when considering daily seasonal periodicity*

address such irregularities, ADMin embraces the use of two online pairwise T-tests which are conducted after each seasonal cycle to evaluate and select which contribution –before ($y_1$) or after ($y_2$)– seasonality knowledge enrichment, allows us to deliver a more accurate estimation. Thus, in the case where seasonality enrichment is not beneficial such as the highlighted example, the seasonal contribution is not taken into consideration in the outputted forecasts of the estimation model.

Finally, a note on the selection of the *seasonal periodicity*. ADMin, through its API allows users to configure the periodicity of the seasonal cycle prior initialization. In cases, where the seasonal cycle is obvious, such as in the case of environment data (e.g., temperature, humidity) this task may seem trivial. However, in general, detecting the optimal seasonal periodicity is deemed a complex problem by itself and by nature it cannot be solved on a low-power device at runtime [140]. Thus, the user may enable ADMin to utilize ComCube [141], which is a lightweight and parameter-free framework embracing a tensor-based approach to determine the near-optimal seasonal periodicity and update it frequently.

## 6.3 Runtime Shift Detection

Shift detection[2] is the process of monitoring the probabilistic distribution of a series of data in order to detect points in time where the statistical properties of the series have changed. These statistical properties may refer to the mean, variance, correlation or spectral density of a metric stream [158]. Several algorithms are available for shift detection such as BCPD, which adopts a binary segmentation approach and models the problem as a bayesian process [159], and SWAB, which considers shift detection

[2] Also commonly referred to as change point detection

**Algorithm 10** ADMin: Runtime Shift Detection

**Input:** *estModel*, confidence $\delta$, local storage $buf$ and *actTime* denoting if actual shift time is returned

**Output:** shift detected notification and *msg* for dissemination

**Ensure:** length($buf$) < max($buf$) otherwise trigger dissemination

1: **if** *dissemination triggered at* $t_{i-1}$ **then**
2: $h_i \leftarrow$ updShiftThres($\delta$, $\sigma_i$) (eq. 6.8)
3: **end if**
4: $c_i \leftarrow$ updLikelihood($v_i$, $\hat{v}_i$, $\mu_i$, $\sigma_i$) (eq. 6.7)
5: $C_{i,low}$, $C_{i,high} \leftarrow$ updCusum($c_i$) (eq. 6.5)
6: $G_{i,low}$, $G_{i,high} \leftarrow$ updDecision($c_i$) (eq. 6.6)
7: **if** $G_{i,\{low,\ high\}} > h_i$ **then**
8: **if** *actTime* == *true* **then**
9: $t_s \leftarrow$ getActShiftTime($C_{i,low}$, $C_{i,high}$) (eq. 6.6)
10: **return** msg($t_s$, $buf$, *estModel*)
11: **else**
12: **return** msg($buf$, *estModel*)
13: **end if**
14: **end if**
15: **return**

a clustering problem between sliding windows of a timeseries [160]. In turn, PELT is an algorithm used to detect the optimal shifts in the evolution of a data series that introduces a pruning function to reduce the complexity [161]. Nonetheless, these algorithms are not suited for IoT settings as they cannot be used on streaming data or are too complex and cannot react to changes in real time.

Having taken the above into consideration, we adopt the Cumulative Sum test (CUSUM) which is a probabilistic likelihood shift detection algorithm that is lightweight and can be used in an online and incremental fashion [162]. Algorithm 10 introduces an abstract overview of our adaptive shift detection approach. The CUSUM, denoted as $C_i$, embraces a hypothesis test for detecting shifts in i.i.d timeseries, such as metric stream ($M$) depicted in Figure 6.4. In particular, there are two hypothesis $\theta'$ and $\theta''$ with probabilities $P(M, \theta')$ and $P(M, \theta'')$, where the

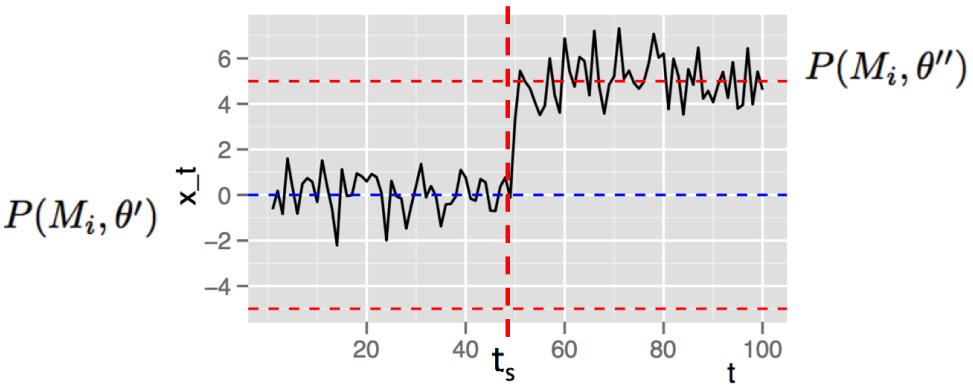



*Figure 6.4: CUSUM shift detection overview*

first corresponds to the statistical distribution of the metric stream prior to a shift ($i < t_s$) and the second to the distribution after a shift ($i > t_s$) with $t_s$ denoting the time interval the shift occurs. The CUSUM is computed with sequential probability testing on the instantaneous log-likelihood ratio given for a metric stream at the $i$-th time interval, as follows:

$$\begin{aligned} c_i &= \ln \frac{P(M_i, \theta'')}{P(M_i, \theta')} \\ C_{i,\{low,\ high\}} &= C_{i-1,\{low,\ high\}} + c_i \end{aligned} \tag{6.5}$$

where *low* and *high* denote the separation of the CUSUM to identify both positive and negative shifts respectively.

The typical behavior of the log-likelihood ratio includes a negative drift before a shift and a positive drift after the shift. Thus, the relevant information for detecting a shift in the evolution of the metric stream lays in the difference between the value of the log-likelihood ratio and the current minimum value. A decision function, denoted as $G_i$, is used to determine a shift in the metric stream when its outcome surpasses a threshold (also referred to as a decision interval) denoted as $h$ and measured in standard deviation units. The time interval at which a shift actually occurs, is computed from the CUSUM as follows:

$$\begin{aligned} G_{i,\{low,\ high\}} &= \{G_{i-1,\{low,\ high\}} + c_i\}^+ \\ t_s &= \arg\min_{j \le s \le i} (C_{s-1}) \end{aligned} \tag{6.6}$$

In Equation 6.6, $G^+ = sup(G, 0)$, $t_i$ is the time ADMin detects the shift and $t_j$ is the time the last shift prior $t_s$ occurs. Now, let us consider the particular case of a metric stream constituted of i.i.d datapoints with the metric stream supposed to undergo

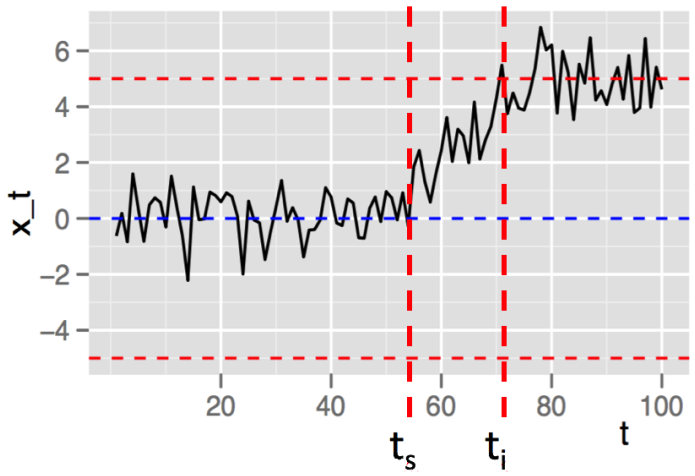



*Figure 6.5: CUSUM delayed shift detection due to metric stream gradual trend*

possible shifts in its evolution modelled by a moving average as in the case of the estimation model previously introduced. Thus, $\theta'$ and $\theta''$ can be rewritten as $\mu'$ and $\mu''$ respectively, with $\mu'$ representing the current evolution, while $\mu''$ the output of the estimation model with $\mu'' = \mu' + \epsilon$, and $\epsilon$ denoting the estimated magnitude of change of the metric stream evolution. As the metric stream evolution is used to provide an estimation for $\hat{v}_i$, the magnitude of change is equal to $\epsilon = \hat{v}_i - v_i$. In turn, let $P(M, \mu')$ and $P(M, \mu'')$ be modeled and computed from Equation 5.5 when adopting a Gaussian kernel as in the case of our estimation model. With some calculations [163], $c_i$ in Equation 6.5 is rewritten, as follows, to perform the decision-making process with only previous value knowledge:

$$c_{i,\{low,\ high\}} = \pm \frac{|\epsilon|}{\sigma_i^2} \left( v_i - \mu' \mp \frac{|\epsilon|}{2} \right) \tag{6.7}$$

However, the CUSUM test features two drawbacks. First, determining the actual $t_s$ requires linear time. In regards to this drawback, if exact knowledge of $t_s$ is not required, then $t_i$, the time a shift is detected from ADMin, can be used as an approximate answer. Nonetheless, in cases of metric streams with gradual trends, as depicted in Figure 6.5, $t_i$ may greatly differ from $t_s$ [164]. However, in the case where trend and seasonality behavior knowledge is added to the estimation process, the metric stream is approximated by ADMin with greater accuracy by quickly adapting to unexpected, abrupt and volatile changes of the metric stream. Thus, with the estimated magnitude of change approximating the actual change in the evolution of the metric stream ($\hat{\epsilon} \rightarrow \epsilon$), the decision function is able to reduce shift detection delays. Second, when the monitoring stream is relatively stable, and thus

the stream variance is deemed as low ($\sigma_i \rightarrow 0$), the CUSUM is prone to falsely signaling that a shift is detected, and thus, model updates are triggered although not needed. Hence, we follow an adaptive approach where $h$ is updated after a model update is triggered, based on the number of standard deviations respecting the given user-defined confidence ($\delta$) and an optional positive value ($h_{min}$) is used to restrict the sensitivity of the CUSUM so as to not oscillate between low values when the monitoring stream is relatively stable.

$$h_i = \max\{h_{min},\ h(\delta_i, \sigma_i)\} \tag{6.8}$$

## 6.4 Experimentation Study

In this section we present a thorough evaluation of ADMin by comparing its performance and accuracy to other state-of-the-art adaptive frameworks with the experimentation based on real-life testbeds and traces.

We compare ADMin to three frameworks: G-SIP [10], LANCE [112] and ADWIN [135], introduced in detail in Chapter 4. Briefly,

- **G-SIP** is a framework which uses an EWMA as its estimation model, to follow the rate at which the metric stream evolution changes in time, triggering datapoint dissemination only if this rate exceeds a threshold policy. We adopt a policy which determines if the estimation falls in the confidence interval given as input by the user.

- **LANCE** is, a similar to G-SIP, framework which also uses an EWMA as its estimation model but for summarizing datapoint values in a given window ($N = 32$ gave the best results). The receiver downloads the datapoints only if the summarization exceeds a threshold policy. We will adopt a similar policy to G-SIP.

- **ADWIN**, on the other hand, is a framework which follows a linear approach with two sliding windows used to detect shifts in the metric stream ($N = 20$ gave the best results). ADWIN uses a Naive Bayes predictor as its estimation model.

We compare all frameworks to ADMin under different configurations and conduct each experiment with a tight confidence parameter of $\delta = 0.9$. For the frameworks

| Trace Name | Origin | Data Points | Optimal Shifts | Description |
|---|---|---|---|---|
| PV Current | PV Panel | 1209598 | 194 | A Photovoltaic (PV) current production trace collected from a PV panel every 1 second for a period of 2 weeks in Jan 2015 |
| Temperature | Meteo Station | 1209598 | 572 | A Temperature trace collected from a remote weather station monitoring the temperature every 1 second for a period of 2 weeks in Jan 2015 |
| Heartrate | Wearable | 40908 | 202 | A Heartrate trace collected from a Fitbit Charge wearable device monitoring beats per minute (bpm) of the person wearing the device for a month (Jun 2016) |

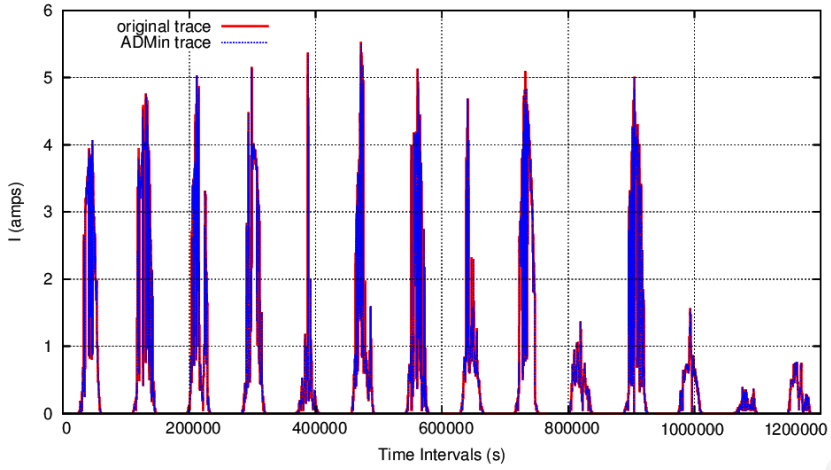
*Table 6.1: Decription of traces used for performance and accuracy evaluation*

using a moving average, we set the smoothing weight to $\alpha = 0.45$, which is the best configuration for both G-SIP and LANCE. We will also leave the trend and seasonality smoothing weight for G-SIP and ADMin to a default value of $\xi = 0.9$ and $\omega = 0.45$ respectively.

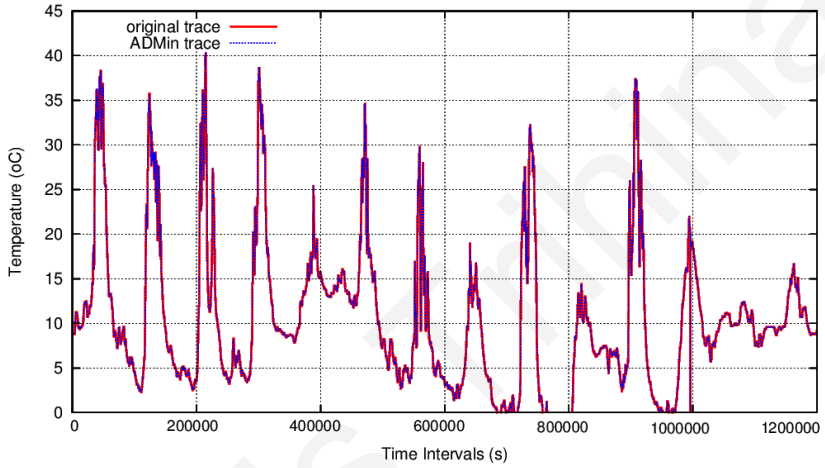
### 6.4.1 Traces, Testbeds and Evaluation Metrics

Table 6.1 presents an overview of the real-life traces used to evaluate the frameworks under comparison. In this table we also present the optimal number of shifts in the metric stream that comprise the ground truth for our evaluation, as obtained from PELT, an optimal offline shift detection algorithm [161]. In turn, Figures 6.6a-6.6c visually depict these traces, where, at a glance, one can observe that each of these traces exhibit irregular seasonality behavior.

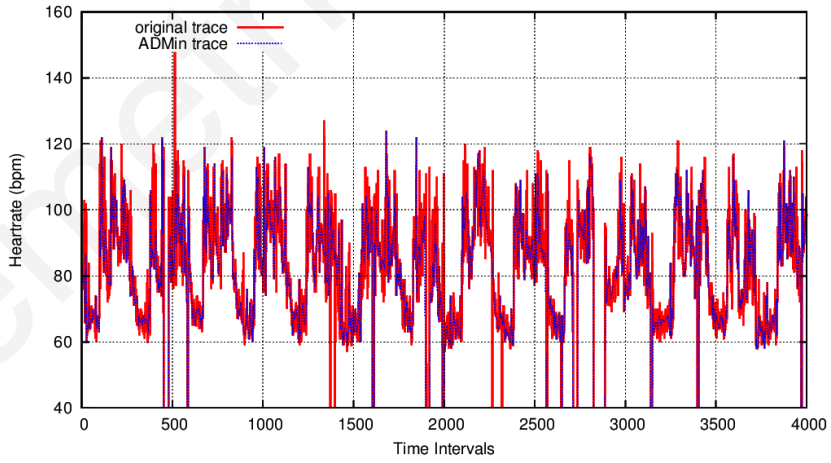
The experiments for the PV and Temperature traces are run on a Raspberry Pi (model B) with 512MB of RAM and an ARM processor (single-core, 700MHz) while emulating the data load of each trace. The Raspberry Pi was selected as a




*(a) PV Panel Production Trace*




*(b) Weather Station Temperature Trace*




*(c) Heartrate Trace from Wearable Device*

*Figure 6.6: Depiction of traces used for performance and accuracy evaluation*

suitable testbed, as it features similar limited processing capabilities of other "smart" devices (e.g., home monitors). The Heartrate raw readings were fed to the Android Wear Emulator hosting an app computing heartrate BPMs. We set the processing capabilities of the emulator to the specifications of a Fitbit Charge wearable device

(single-core ARM 32MHz processor, 128MB Memory). We evaluate each framework towards towards their estimation accuracy and ability to efficiently use resources.
**Accuracy**: we evaluate an adaptive technique by measuring *shift detection accuracy* in relation to both the number of correctly detected shifts (true positives) and the number of false alarms (false positives) when compared to the ground truth; and *shift detection delay*, which is the difference in time to when a framework detects a shift from the actual time of occurrence.
**Efficiency**: we evaluate efficiency by, first, measuring the overall *data volume reduction* and *accuracy* at the receiver-side; and, secondly, by measuring the overhead in relation to *total energy consumption* on the testbed device based on the energy consumption model introduced in Section 5.7.1.

### 6.4.2 Experiments

At first, let us denote the different configurations for our framework. **ADMin** denotes our framework without any seasonality enrichment while **ADMin_S1** and **ADMin_S2** feature seasonality configurations. Specifically,

- **ADMin_S1** follows a Holt-Winters additive approach and uses a static seasonal period configured once upon initialization and representing seasonal knowledge corresponding to the previous day hourly average (e.g., for the current day at 11.05am the average between 11-11.59am of the previous day is considered).
- **ADMin_S2**, on the other hand, uses the output of the ComCube framework [141], which provides a near-optimal approximation of the seasonal periodicity ($L$ in eq. 6.3).

In our first evaluation we compare each framework ability to detect the actual shifts in the evolution of the metric stream. From Figures 6.7a-6.7c, we observe that ADMin achieves, in all configurations, high accuracy approaching the ground truth. ADWIN has a slightly lower accuracy although it is comparable to ADMin (less than 10% difference). However, G-SIP and LANCE feature shift accuracy that significantly varies between traces and is never higher than 73% and 65% respectively. Most importantly, we note the high number of false alarms observed from LANCE and G-SIP due to their restrictive shift detection process. On the other hand, *ADMin is*

| Framework | PV Current (Time Intervals) | Temperature (Time Intervals) | Heartrate (Time Intervals) |
|---|---|---|---|
| ADWIN | 9.34 ± 3.47 | 9.94 ± 3.84 | 10.39 ± 3.96 |
| G-SIP | 10.02 ± 3.96 | 11.76 ± 4.16 | 14.17 ± 4.93 |
| LANCE | 10.78 ± 4.12 | 12.63 ± 3.92 | 15.97 ± 4.12 |
| ADMin | 6.04 ± 2.19 | 7.12 ± 1.97 | 8.03 ± 2.78 |
| ADMin_S1 | 3.13 ± 2.03 | 5.11 ± 2.10 | 6.22 ± 2.83 |
| ADMin_S2 | **2.62 ± 1.94** | **3.23 ± 2.26** | **4.73 ± 2.43** |

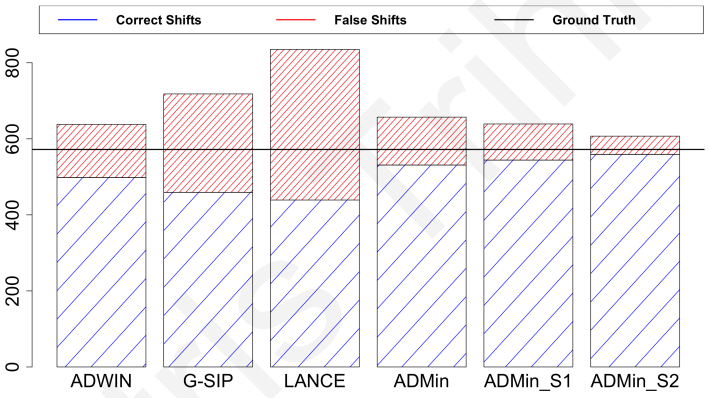
*Table 6.2: Metric Stream Evolution Shift Detection Delay*

*able to achieve a low false alarm ratio and when incorporating seasonality knowledge this ratio is drastically reduced*. Specifically, *with seasonal knowledge ADMin false alarm ratio is under 10% and at least 47% less than the other frameworks*.

Table 6.2 depicts the average time required to detect a shift in the evolution of the metric stream. We observe that *ADMin outperforms the other frameworks by reducing shift detection time by at least 29% due to its runtime shift detection which incorporates enough knowledge of the stream estimation process to quickly identify actual from false shifts in the confidence intervals given by the user*. Moreover, ADMin_S1 and ADMin_S2 are able to reduce shift detection time even more. For traces with irregular seasonality behavior such as the temperature and heartrate trace where more than one seasonal cycles may exist (e.g., daily, weekday, weekend patterns) ADMin clearly outperforms the other frameworks. Specifically, ADMin_S2 achieves a reduction in shift detection time of at least 67%. This also justifies the use of low-cost streaming frameworks such as ComCube, which support ADMin by fine-tuning the seasonal periodicity. Hence, *ADMin is able to reduce the time required to detect shifts in the metric stream evolution by at least 29% and when incorporating seasonality knowledge to the estimation model reduction is at least 67% when compared to other frameworks*.

In the next set of experiments we compare the ability of each framework to reduce on device energy consumption (Figure 6.8), as well as, the volume of IoT data and estimation error at the receiver-side (Figures 6.9a-6.9b). We note that, ADWIN is not present in the receiver-side comparison as it does not apply a data reduction scheme to the metric stream. Moreover, a reference baseline approach is added


*(a) PV Current Trace*




*(b) Weather Station Temperature Trace*


*(c) Heartrate Trace*

*Figure 6.7: Shift Detection Accuracy Comparison*

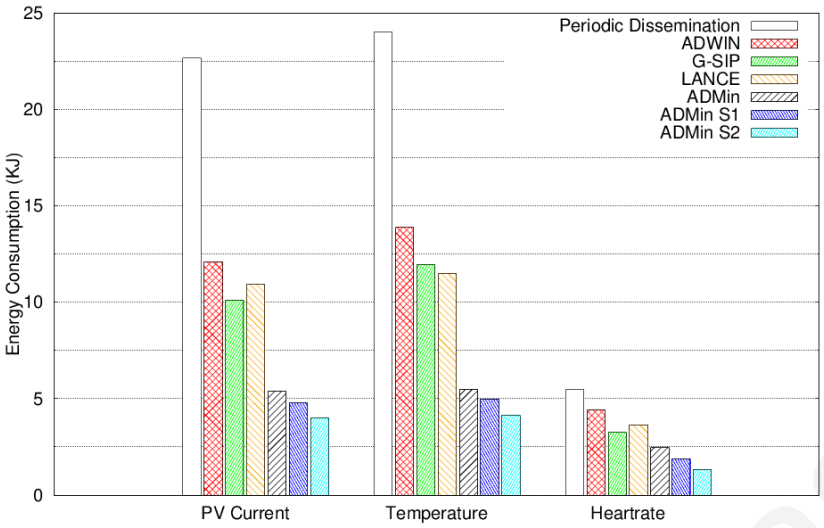
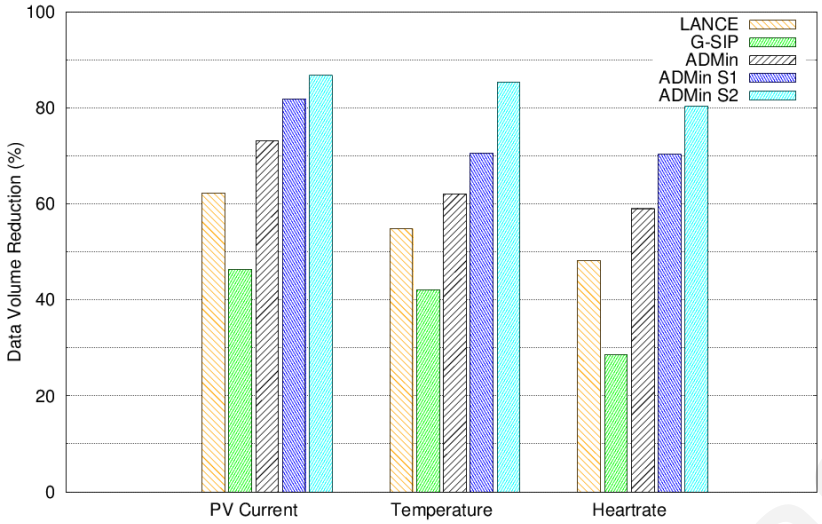


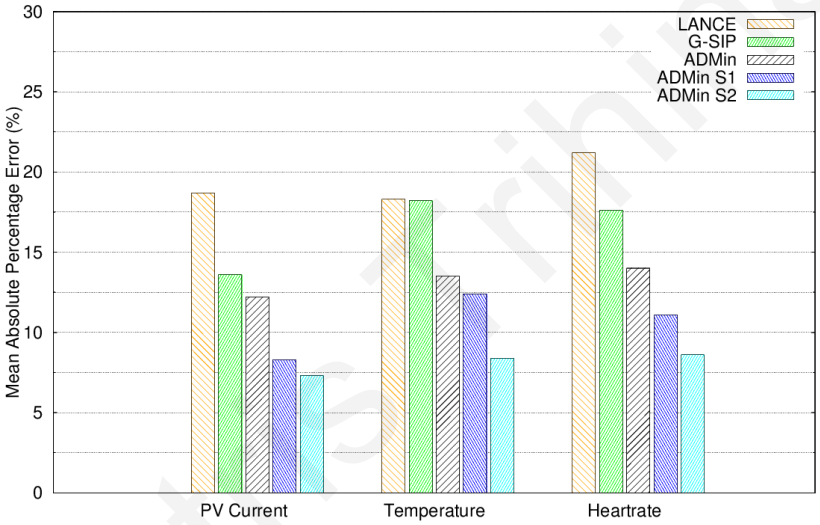

*Figure 6.8: On Device Energy Consumption Comparison*

to the comparison where dissemination is withhold by applying a 10 time interval aggregation scheme.

From Figure 6.8 we observe that ADWIN features the largest energy footprint although we have previously shown that it is able to detect shifts in the metric stream with accuracy comparable to ADMin. This is due to the complexity of the ADWIN algorithm, linear in space and time, along with the absence of a data reduction scheme resulting in significant energy spent for datapoint dissemination and the algorithm itself. On the other hand, LANCE, G-SIP and ADMin have similar complexity requirements. However, as previously shown, LANCE and G-SIP feature a high false alarm ratio which enables the network controller of the IoT device at least x2 times more than ADMin. In turn, from Figure 6.9a one can observe that the data reduction model of LANCE and G-SIP are not as efficient as ADMin. In the case of LANCE, downloading the entire window length of data when the summary is labeled as meaningful drastically affects performance. For G-SIP, not enough knowledge is preserved in the estimation model (only comprised of an EWMA) after datapoint dissemination, often triggering dissemination in subsequent intervals for model updating. Nonetheless, *ADMin is able to reduce data volume by at least 71% which accounts for a reduction in energy consumption of at least 83%*. Most importantly, from Figure 6.9b one can observe that *in regards to accuracy ADMin outperforms the other frameworks by always maintaining accuracy at the receiver to at least 86%, increasing to at least 91% when seasonality behavior is acknowledged by the estimation model*.




*(a) Data Reduction Comparison*




*(b) Receiver-Side Mean Absolute Percentage Error*

**Chapter Summary.** This chapter introduces the ADMin plugin developed to extend the functionality of the AdaM framework to support model-based adaptive dissemination to lower the communication overhead in IoT networks, especially, when monitoring data features seasonal patterns and trends. To achieve this, ADMin adopts the low-cost approximate stream estimation model of the AdaM framework used to capture the runtime evolution and trend of the monitoring stream and extends it to also include a seasonal component to reduce the estimation error and model shift detection delays. In the case where seasonality enrichment is not beneficial to the estimation process, ADMin via online statistical testing will determine this and remove the contribution of the seasonal component. Runtime shift detection is applied to determine if the sensed datapoints can be inferred by the disseminated estimation model within the confidence intervals given by the user and if not, then a

dissemination is triggered with an updated version of the estimation model parameterization. After performing a thorough experimentation study using real-world data from photovoltaics, weather stations and wearable devices, results show that ADMin reduces energy consumption by at least 76%, data volume by 60%, while maintaining accuracy always above 86% when compared to a baseline approach. When incorporating seasonality knowledge, energy consumption is reduced by at least 83%, data volume by 71% while accuracy is always above 91%. Most importantly, the false alarm rate and shift detection delays are reduced by 47% and 61% respectively, when compared to other IoT frameworks.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

With the prevalence of the Internet of Things we are starting to see intelligence, in the form of internet-enabled "smart" devices, aggressively deployed at the edge to produce real-time analytic insights for almost all industry sectors. However, to produce such an unprecedented wealth of insights, intense processing and constant data dissemination over the network are still required. This results in increased energy consumption for monitoring sources while cloud and streaming services consuming IoT data are constantly overwhelmed and struggling to be effective. Despite attempts of augmenting IoT devices with the power of the cloud there still exist numerous inhibitors masked under constant data movement such as bandwidth limitations and network latencies. Thus, processing monitoring data on the "edge" is a valid option for IoT, albeit intense processing results in increased energy-consumption and for battery-powered devices this mean less battery life.

In this thesis, we tackle real-time data processing and energy-efficiency on the edge of monitoring networks by developing low-cost approximate and adaptive monitoring techniques as the remedy to these challenges. Hence, our main idea is that if a degree of inaccuracy can be tolerated, approximate monitoring techniques such as adaptive sampling, filtering and model-based dissemination, can significantly reduce the energy consumption of monitoring sources and the amount of data transmitted, over the network, to streaming services by dynamically adapting, in place and inexpensively, the metric collection and dissemination rate based on the metric stream evolution and variability.

As presented extensively in the thesis, IoT devices can greatly benefit from "adaptivity" as monitoring data collection and dissemination are the two prime energy drains in embedded and mobile sensing devices. To achieve this, we have designed and developed AdaM, a lightweight framework embeddable in the software core of monitoring sources (e.g., IoT devices) to provide model-based adaptive monitoring by incorporating the low-cost approximate monitoring techniques introduced in the scope of this thesis. AdaM is able to dynamically adapt the monitoring intensity and the amount of data disseminated through the network by incorporating a low-cost estimation model with the ability to capture the runtime evolution and variability of a metric stream within certain accuracy guarantees. This allows the algorithms we have developed, to adjust the metric collection and dissemination rate of the monitoring source based on the confidence of the estimation model to correctly estimate what will happen next in the metric stream. As we show, while designing AdaM, specific consideration was taken to fine-tune the algorithmic model at runtime by introducing adaptive parameter weighting, trend detection and seasonality behavior enrichment so that our algorithms immediately identify abrupt transient changes in the metric stream evolution and overcome any lagging effects in the estimation process. By accomplishing this, energy consumption and data volume are reduced, allowing monitoring sources to preserve energy and ease processing at data receiving endpoints, while still preserving accuracy.

After performing a thorough experimentation study with testbeds using real-world data from cloud applications, wearables, photovoltaics and intelligent transportation services, our results show that AdaM succeeds in being able to dynamically adapt both the collection and dissemination rate of a monitoring source and, thus, is a viable solution that is lightweight, practical and achieves a balance between efficiency and accuracy for numerous, diverse and real data streams.

Highlighting our framework and the conducted evaluation:

- For the evaluated testbeds, AdaM can significantly reduce energy consumption by at least 83% and the volume of generated data by at least 71%, while maintaining accuracy always above 89%. These results are achieved without additional monitoring costs, due to: (i) AdaM being placed on the monitoring source which eliminates the need for additional communication; (ii) the entire

estimation process which features an O(1) time and space complexity; and (iii) the ability of the estimation process to capture and adapt to the evolution of the metric stream even in highly abrupt and transient phases.

- With the integration of the ADMin plugin to AdaM, IoT networks streaming monitoring data and featuring seasonal patterns and trends, can greatly benefit in terms of lower communication overhead and on-device energy consumption, by employing model-based dissemination. In contrast to other evaluated frameworks, with the employment of an estimation model capturing the runtime evolution, trends, and seasonal behavior; ADMin can reduce data volume by at least 71% while maintaining accuracy always above 91%. Most importantly, ADMin reduces by 61% the time needed to detect shifts in the metric stream behavior. Hence, by utilizing ADMin, monitoring services can significantly reduce the estimation error by timely adapting to highly abrupt and transient shifts in the data stream.

- With a wide range of real-world datasets originating from different application domains, we show that in contrast to the other under-comparison techniques, the estimation process of the AdaM framework ensures that the user-defined accuracy guarantees are obeyed at all times. In particular, AdaM is capable of capturing the metric stream evolution based on the confidence of the estimation model to correctly estimate what will happen next in the metric stream and is able to follow and adapt to the evolution of the metric stream even in highly abrupt and transient phases. This supports our framework to "reward" larger property adjustments (e.g., periodicity) when estimations fall within the accuracy intervals defined by the user and rollback to a fixed approach when satisfactory estimations cannot be provided. Thus, if a "confident" estimation cannot be made, AdaM will rollback to a fixed approach (e.g., periodic sampling) to ensure, at all times, the accuracy guarantees are obeyed over efficiency. Hence, in the case of either an adversary attack or a data stream which does not present any correlations among consecutive datapoints, AdaM will be at least as effective as a fixed approach to ensure accuracy.

- While a number of parameters can be configured, especially for the AdaM estimation model, the analysis conducted shows that with probabilistic weighting

significant profiling to find the optimal settings is not required and if slight imprecision (< 3%) is acceptable a wide range of values can be used. Therefore, novice users need only to set the maximum acceptable imprecision (error) and can define the min/max range for sampling, filtering and dissemination rate.

- AdaM as a lightweight framework developed in java and with no external dependencies, it can be deployed to numerous monitoring sources and streaming settings. To date, AdaM has can be deployed on Raspberry Pi micro-controllers, Linux-based environments, Android Wear devices and cloud monitoring tools with adaptive interfaces (e.g., JCatascopia [22]). In turn, AdaM has also been ported to, R, for offloading temporal graph metric computation [138], and Python, for efficient data stream processing in Apache Flink [139].

For the discussion to be thorough, we note the limitations of the AdaM framework which can be considered as the operational boundaries of our framework as well:

- A monitoring stream attached to AdaM must produce one-dimensional univariate datapoints. Multivariate datapoints such as GPS coordinates are not supported. In turn, datapoint values must be numeric. Nominal values can be supported but only if a mapping to numerical values is valid and provided.

- AdaM prioritizes accuracy over efficiency. This means AdaM will always attempt to satisfy the user-given accuracy guarantees following a more cautious approach, which will rollback to a fixed approach, when the evolution of the metric stream cannot be captured by the AdaM estimation model. In any case, AdaM will be at least as effective as a fixed approach.

- AdaM makes no assumption of the resource capabilities of the underlying hardware the monitoring source resides on, so that it adjusts the estimation process depending on the resources made available. This means that the estimation process of the AdaM framework does not make any distinction between a low-power sensing device and a powerful computing server.

## 7.2 Future Work

The results of this thesis show that if a degree of inaccuracy can be tolerated, low-cost approximate and adaptive monitoring techniques can significantly reduce the energy

consumption of monitoring sources and the amount of data transmitted, over the network, to streaming services by dynamically adapting, in place and inexpensively, the metric collection and dissemination rate based on the metric stream evolution and variability. At the same time, new research directions arise from the outcomes of this work and can be further exploited to increase wide-spread adoption of approximate monitoring and applicability of the AdaM framework.

**AdaM for micro-services**

The micro-service architectural paradigm arises from the decomposition of an application into smaller pieces (services) organized around discrete business capabilities [165]. The boundaries between these units are usually comprised of functional APIs that expose the core capabilities of each service. Large systems are then composed of many (micro-) services, whereby communication is a central ingredient [166]. Micro-services are ideal for supporting a wide range of domains spanning web, mobile and IoT with software teams stating that faster innovation via micro-services is ideal for rapid development, testing, handling failures and (elastic) scaling. In light of this, the EU co-funded Unicorn project [167], aims to empower the startup eco-system by delivering a unified framework that simplifies the design, deployment and management of secure and elastic-by design cloud applications that follow the micro-service architectural paradigm. To reduce monitoring costs, which are billable and noticeable in distributed topologies, data movement across cloud sites, and the intrusiveness on containerized deployments, we plan to integrate the AdaM framework in the open-source micro-service Spring Cloud ecosystem, developed by Netflix [168]. Hence, the Unicorn project will apply low-cost approximate and adaptive techniques with AdaM for micro-service monitoring to reduce the monitoring footprint and the velocity of data disseminated over the network.

**Adapting the "temporality" of temporal networks**

Temporal networks[1] are networks in which the interactions among a set of elementary units change over time and can be modelled in terms of temporal graphs, which are time-ordered sequences of graphs over a set of nodes [169]. In such graphs, the concepts of node adjacency and reachability crucially depend on the exact temporal

[1] Also referred to as time-varying networks or dynamic networks

ordering of the links [170]. With the physical world becoming an information system itself, temporal graphs are becoming very popular as they feature the analytical benefits of static graph analysis while at the same time retain all temporal information and interactions that may be available [171]. Examples of such temporal networks include email/sms exchange networks, face-to-face interaction networks, vehicular networks and online social networks (OSNs) [172]. While temporal graphs feature multiple benefits, computing complex graph metrics such as betweenness centrality, components and maximum clique size, in short time intervals, is a challenging and resource-intensive task even for organizations equipped with large-scale distributed (graph) processing engines [173]. At the same time, monitoring sources distributed at the edge of the monitored network and comprising the nodes of the temporal graph must, timely, disseminate and receive processed data.

AdaM could be used to adapt the periodicity at which graph metrics are computed if a graph monitoring stream at each distinct time interval outputs graph instances. Still, adaptive sampling dynamically adjusts the monitoring intensity based on the runtime evolution and variability of the metric stream, thus based on the monitoring stream temporal effects. Thus, if the temporal evolution and variability of the metric representing the graph stream introduces "stable" phases then the periodicity of the metric computation is reduced to preserve resources. Otherwise, the periodicity is increased to capture sudden events and insights. However, there exist cases of metrics (e.g., graph diameter) where the temporal evolution of the metric may introduce "stable" phases although the structure of the graph drastically differs from instances of the graph in previous time intervals. Thus, while reducing the periodicity based on temporal effects of a graph metric may preserve resources by computationally offloading graph processing engines, it hinders the challenge of missing structural changes which might capture and reveal significant insights (e.g., capacity management). Therefore, an adaptive implementation developed to adapt the "temporality" of graphs to reduce the compute and network overhead of both graph processing engines and monitored sources comprising the network while still preserving given accuracy guarantees, must consider and correlate both temporal and structural volatility effects. Thus, our main idea is to extend the algorithmic learning model of the AdaM framework to acknowledge, in time, graph topology structure knowledge to allow the estimation process of the adaptive sampling implementation to adapt within given confidence intervals the periodicity of

the computation of graph metrics.

**Making sense of the plethora of monitoring data from the dashboard**

Monitoring data visualization of streaming telemetry is increasingly prevalent in modern data platforms and applications. Data volumes continue to rise, fueled in large part by an increasing number of automated sources, including sensors, processes, and devices. For example, Facebook reports that their production infrastructure generates over 12 million datapoints per second [133] while Uber receives more than 3 million telemetry updates per second [73]. As a result, the past several years have seen an explosion in the development of platforms for managing, storing, and querying large-scale metric streams of timestamped data. However, many existing systems simply plot raw monitoring streams as they arrive, often obscuring as data volume and the plethora of metrics increase, large-scale trends, seasonal patterns and runtime shifts in the stream evolution.

While the AdaM framework is able to significantly reduce the volume of monitoring data based on the evolution and variability of the metric stream and it is able to accurately detect, in time, runtime shifts in the estimation model, visualizing trends, seasonal behavior and evolution shifts, is a complex task by itself. To address this, our main idea is two-fold: (i) to reduce the plethora of metric streams reported per application, clustering can be used to group metrics featuring spatio-temporal correlations and, thus, reduce the dimensionality of the metric space by filtering out metrics that carry redundant information; and (ii) along with AdaM approximate and probabilistic learning scheme which can be used for metric stream smoothing, various mechanisms can be applied to ease visualization and pinpoint trends and shifts in the evolution of a metric stream (e.g., bandit-based hyperparameter tuning [174], kurtosis-based "roughness" measure [175], and MDP outlier-aware detection [127]).

**Supporting multivariate metric streams**

The algorithmic implementation of the estimation process part of the AdaM framework, currently supports one-dimensional metric streams with the monitoring source properties (e.g., periodicity) dynamically adjusted based on the evolution of a metric stream characterized by univariate one-dimensional datapoint values. However,

there exist cases of metrics that cannot be referenced as one-dimensional datapoints, such as GPS coordinates which are characterized by latitude and longitude. To support metric classes of higher dimensionality, where $v_i \mapsto \mathbb{R}^k$ and $k$ denotes the metric dimensionality, the estimation process of the AdaM framework must be extended to support multivariate metric streams. To this end, the PEWMA depicting the metric stream evolution can be extended to a probabilistic Multivariate Moving Average outputting a $K-$dimensional vector, thus, linearly scaling the computation towards the dimensionality of the metric stream [176]. However, as the estimation process confidence metric depends on the variance of the metric stream evolution, a covariance matrix is required for K-dimensional metrics which is a $K \times K$ matrix. This, significantly increases the computation effort of the estimation process and can negatively affect the overall benefits in introducing adaptiveness to the monitoring process. To address the challenges introduced in supporting multivariate metric classes, we envision adopting dimensionality reduction to computationally ease the estimation process. However, this process is not straightforward, with meaningful projections to lower-dimensional spaces only possible if correlations among the dimensions exist, which introduces another challenge, selecting a low-cost exploratory process to detect correlations in multivariate metric streams [177].

# Bibliography


[1] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Gener. Comput. Syst.*, vol. 21, no. 1, pp. 163–188, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.future.2004.07.002

[2] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward, "Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 510–522, July 2011.

[3] M. Chui, M. Loffler, and R. Roberts, "McKinsey: The Internet of Things," Mar 2010.

[4] T. Jorg, R. Antonio, A. Istemi Ekin, B. Pramod, C. Ruichuan, V. Bimal, J. Lei, and F. Christof, "Sieve: Actionable insights from monitored metrics in distributed systems," in *Proceedings of Middleware Conference (Middleware)*, 2017.

[5] K. Schwab, *The Fourth Industrial Revolution*, Jan 2016.

[6] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-Cost Adaptive Monitoring Techniques for the Internet of Things," *IEEE Transactions on Services Computing*, 2017.

[7] V. Cerf and M. Senges, "Taking the internet to the next physical level," *Computer*, vol. 49, no. 2, pp. 80–86, Feb 2016.

[8] IEEE Computer Society, "IEEE Computer Society Predicts the Future of Tech for 2017 and Next Five Years," Dec 2016.

[9] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, "Practical data prediction for real-world wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2231–2244, Aug 2015.

[10] E. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, "Edge Mining the Internet of Things," *Sensors Journal, IEEE*, vol. 13, no. 10, pp. 3816–3825, Oct 2013.

[11] S. Meng and L. Liu, "Enhanced Monitoring-as-a-Service for Effective Cloud Management," *Computers, IEEE Transactions on*, vol. 62, no. 9, pp. 1705–1720, Sept 2013.

[12] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 4, pp. 2233–2243, Nov 2014.

[13] T.-D. Cao, T.-V. Pham, Q.-H. Vu, H.-L. Truong, D.-H. Le, and S. Dustdar, "A Marketplace for Realtime Human Sensing Data," *ACM Trans. Internet Technol.*, vol. 16, no. 3, 2016.

[14] Gartner, "8.4 Billion Connected "Things" Will Be in Use in 2017," http://www.gartner.com/newsroom/id/3598917, 2017.

[15] L. Mearian, "Self-driving cars could create 1GB of data a second," https://www.computerworld.com/article/2484219/emerging-technology/self-driving-cars-could-create-1gb-of-data-a-second.html, 2013.

[16] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. Philadelphia, PA: USENIX Association, Jun. 2014.

[17] A. Jacobs, "The pathologies of big data," *Commun. ACM*, vol. 52, no. 8, pp. 36–44, Aug. 2009.

[18] Rich Data and the Increasing Value of the Internet of Things, http://goo.gl/sfk1hW.

[19] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[20] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[21] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, Jul. 2015. [Online]. Available: https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/zhang

[22] D. Trihinas, G. Pallis, and M. Dikaiakos, "Monitoring Elastically Adaptive Multi-Cloud Services," *IEEE Transactions on Cloud Computing*, vol. 4, 2016.

[23] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15. New York, NY, USA: ACM, 2015, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757397

[24] Colin Wood, "Is Edge Computing Key to the Internet of Things?" http://www.govtech.com/transportation/Is-Edge-Computing-Key-to-the-Internet-of-Things.html, 2015.

[25] B. Moulton, L. Hanlen, J. Chen, G. Croucher, L. Mahendran, and A. Varis, "Body-area-network transmission power control using variable adaptive feedback periodicity," in *2010 Australian Communications Theory Workshop (AusCTW)*, Feb 2010, pp. 139–144.

[26] F. Gu, J. Niu, S. K. Das, and Z. He, "RunnerPal: A Runner Monitoring and Advisory System Based on Smart Devices," *IEEE Transactions on Services Computing*, 2016.

[27] Nagios, http://www.nagios.org/.

[28] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation And Experience," *Parallel Computing*, vol. 30, 2003.

[29] Zabbix, http://www.zabbix.com/.

[30] Amazon CloudWatch, https://aws.amazon.com/cloudwatch/.

[31] Paraleap AzureWatch, https://www.paraleap.com/AzureWatch.

[32] Openstack Ceilometer, https://wiki.openstack.org/wiki/Telemetry.

[33] New Relic APM, https://newrelic.com/application-monitoring.

[34] D. Trihinas, G. Pallis, and M. Dikaiakos, "JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud," in *Cluster, Cloud and Grid Computing (CCGrid), 14th IEEE/ACM International Symposium on*, May 2014, pp. 226–235.

[35] ——, "ADMin: adaptive monitoring dissemination for the internet of things," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications (INFOCOM 2017)*, Atlanta, USA, May 2017.

[36] S. Khalifa, M. Hassan, A. Seneviratne, and S. Das, "Energy-harvesting wearables for activity-aware services," *Internet Computing, IEEE*, vol. 19, no. 5, pp. 8–16, Sept 2015.

[37] Design Considerations for Wrist-Wearable Heart Rate Monitors, https://www.arrow.com/en/research-and-events/articles/design-considerations-for-wrist-wearable-heart-rate-monitors, 2015.

[38] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[39] C. Perera, C. Liu, and S. Jayawardena, "The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey," *Emerging Topics in Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[40] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 735–746.

[41] L. Fan and L. Xiong, "An adaptive approach to real-time aggregate monitoring with differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2094–2106, Sept 2014.

[42] F. McSherry and R. Mahajan, "Differentially-private network trace analysis," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 123–134, Aug. 2010.

[43] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, p. 1101, 2012.

[44] S.-H. Yang, *Internet of Things*. London: Springer London, 2014, pp. 247–261.

[45] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal*, Jun 1999.

[46] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999. [Online]. Available: http://doi.acm.org/10.1145/329124.329126

[47] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, 2010.

[48] M. Ratner and D. Ratner, *Nanotechnology: A Gentle Introduction to the Next Big Idea*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2002.

[49] A New Perspective On The Internet Of Things, Rakesh Sharma, Forbes, http://www.forbes.com/sites/rakeshsharma/2014/02/18/a-new-perspective-on-the-internet-of-things/.

[50] G. Chatzimilioudis, C. Costa, D. Zeinalipour-Yazti, W.-C. Lee, and E. Pitoura, "Distributed in-memory processing of all k nearest neighbor queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, pp. 925–938, 2016.

[51] I. Mitliagkas, C. Caramanis, and P. Jain, "Memory limited, streaming pca," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, December 2013, p. 2886–2894.

[52] S. Mayer, D. Guinard, and V. Trifa, "Searching in a web-based infrastructure for smart things," in *Internet of Things (IOT), 2012 3rd International Conference on the*, Oct 2012, pp. 119–126.

[53] V. Megalooikonomou, G. Li, and Q. Wang, "A dimensionality reduction technique for efficient similarity analysis of time series databases," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 2004, pp. 160–161.

[54] Envisioning the Future of Cities Now – Cisco, http://www.cisco.com/c/dam/en_us/solutions/industries/docs/smart-cities-expo-barcelona.pdf, 2015.

[55] "Dublin Bus GPS Dataset," http://dublinked.com/datastore/datasets/dataset-304.php, Jan 2014.

[56] Smart Cities are Built on the Internet of Things, http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/smart_cities_are_built_on_iot_lopez_research.pdf, 2016.

[57] G. Neff and D. Nafus, *The Self-Tracking*, 2016.

[58] S. Dent, "Engadget: Fitbit data led doctors to shock a patient's heart," Apr 2016. [Online]. Available: http://www.engadget.com/2016/04/11/fitbit-data-led-doctors-to-shock-a-patient-s-heart/

[59] J. J. Darrow, "Harvard Blog: Capsule Endoscopy Instead of Colonoscopy? The FDA Approves the PillCam COLON," Mar 2014. [Online]. Available: http://blogs.harvard.edu/billofhealth/2014/03/04/capsule-endoscopy-instead-of-colonoscopy-the-fda-approves-the-pillcam-colon/

[60] I. Benabdallah, A. Oun, and A. Cherif, "Grid connected pv plant based on smart grid control and monitoring," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 6, pp. 299–306, 2017.

[61] H. G. Lee and N. Chang, "Powering the iot: Storage-less and converter-less energy harvesting," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 124–129.

[62] B. Research, "Towards Smart Farming - Agriculture Embracing the Internet of Things," 2014.

[63] K. Pretz, "Connected Cattle: Wearables are Changing the Dairy Industry," *IEEE Magazine: the Institute*, May 2016.

[64] N. Loulloudes, C. Sofokleous, D. Trihinas, M. D. Dikaiakos, and G. Pallis, "Enabling interoperable cloud application management through an open source ecosystem," *IEEE Internet Computing*, vol. 19, no. 3, pp. 54–59, May 2015.

[65] D. Trihinas, Z. Georgiou, G. Pallis, and M. D. Dikaiakos, "Improving rule-based elasticity control by adapting the sensitivity of the auto-scaling decision timeframe," in *Third International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD 2017), in conjunction with the ALGO 2017 Conference*, ser. ALGO 2017 Conference, Vienna, Austria, September 2017, conference.

[66] L. Willcocks, W. Venters, and E. A. Whitley, *Cloud in Context: Managing New Waves of Power*. London: Palgrave Macmillan UK, 2014, pp. 1–19.

[67] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[68] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. S. Rosing, J. Wawrzynek, D. Wessel, J. Rabaey, K. Pister, A. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, "The swarm at the edge of the cloud," *IEEE Design Test*, vol. 31, no. 3, pp. 8–20, June 2014.

[69] Embarcadero, "State of IoT 2015," Apr 2015. [Online]. Available: https://www.embarcadero.com/united-states/developer-survey-reveals-business-shift-toward-internet-of-things-solutions-in-2015

[70] I. Skerrett, "Eclipse Foundation IoT Developer Survey 2016," Mar 2016. [Online]. Available: http://www.slideshare.net/IanSkerrett/iot-developer-survey-2016

[71] A. Bakshi, L. Chen, K. Srinivasan, C. Koksal, and A. Eryilmaz, "EMIT: An Efficient MAC Paradigm for the Internet of Things," in *2016 IEEE Conference on Computer Communications (INFOCOM)*, 2016.

[72] M. Finnegan, "Boeing 787s to create half a terabyte of data per flight, says Virgin Atlantic," https://www.computerworlduk.com/data/boeing-787s-create-half-terabyte-of-data-per-flight-says-virgin-atlantic-3433595/, 2013.

[73] Uber Engineering, "Observability at Uber Engineering: Past, Present, Future," https://www.youtube.com/watch?v=2JAnmzVwgP8, 2017.

[74] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[75] B. Xiao, Y. Shi, and L. He, "A universal state-of-charge algorithm for batteries," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 687–692.

[76] H. Bauer, M. Patel, and J. Veira, "The internet of things: Sizing up the opportunity," *McKinsey Report*, Dec 2014.

[77] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918 – 2933, 2014.

[78] IBM Tivoli, https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Tivoli%20Monitoring, 2016.

[79] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, and M. C. Vistoli, "Gridice: a monitoring service for grid systems," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 559–571, Apr. 2005.

[80] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: online monitoring and analytics for managing large scale data centers," in *Proceedings of the 7th international conference on Autonomic computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 141–150.

[81] D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar, "Mela: elasticity analytics for cloud services," *International Journal of Big Data Intelligence*, vol. 2, no. 1, pp. 45–62, 2015.

[82] C. Wang, I. A. Rayan, G. Eisenhauer, K. Schwan, V. Talwar, M. Wolf, and C. Huneycutt, "Vscope: middleware for troubleshooting time-sensitive data center applications," in *Proceedings of the 13th International Middleware Conference*. NY, USA: Springer-Verlag New York, Inc., 2012, pp. 121–141.

[83] Cacti Monitoring, http://www.cacti.net/, 2016.

[84] V. C. Emeakaroha, M. A. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. D. Rose, "Towards autonomic detection of {SLA} violations in cloud infrastructures," *Future Generation Computer Systems*, vol. 28, no. 7, 2012, special section: Quality of Service in Grid and Cloud Computing.

[85] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson, "A monitoring sensor management system for grid environments," in *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 97–. [Online]. Available: http://dl.acm.org/citation.cfm?id=822085.823304

[86] A. Cooke, A. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, J. Leake, R. Middleton, A. Wilson, X. Zhu, N. Podhorszki, B. Coghlan, S. Kenny, D. Callaghan, and J. Ryan, "The relational grid monitoring architecture: Mediating information about the grid," *Journal of Grid Computing*, vol. 2, no. 4, pp. 323–339, 2004. [Online]. Available: http://dx.doi.org/10.1007/s10723-005-0151-6

[87] S. De Chaves, R. Uriarte, and C. Westphall, "Toward an architecture for monitoring private clouds," *Communications Magazine, IEEE*, vol. 49, no. 12, pp. 130–137, December 2011.

[88] D. L. Quoc, L. Yazdanov, and C. Fetzer, "Dolen: User-side multi-cloud application monitoring," in *IEEE Future Internet of Things and Cloud*, 2014.

[89] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with Lattice," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010.

[90] J. Montes, A. Sanchez, B. Memishi, M. S. Perez, and G. Antoniu, "Gmone: A complete approach to cloud monitoring," *Future Generation Computer Systems*, 2013.

[91] OpsView Monitoring, https://www.opsview.com/, 2016.

[92] RackSpace Cloud Monitor, http://www.rackspace.com/cloud/monitoring/.

[93] Datadog Real-Time Performance Monitoring, https://www.datadoghq.com.

[94] AppDynamics, https://www.appdynamics.com/.

[95] M. Corcoran, "The as-a-service economy is untapped," *Forbes*, Nov 2015.

[96] J. Alcaraz Calero and J. Gutierrez Aguado, "Monpaas: An adaptive monitoring platform as a service for cloud computing infrastructures and services," *Services Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[97] Xen Hypervisor, https://www.xenproject.org/.

[98] R. Koller, C. Isci, S. Suneja, and E. De Lara, "Unified monitoring and analytics in the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[99] M. B. de Carvalho and L. Z. Granville, "Incorporating virtualization awareness in service monitoring systems." in *Integrated Network Management*, N. Agoulmine, C. Bartolini, T. Pfeifer, and D. O'Sullivan, Eds. IEEE, 2011, pp. 297–304.

[100] G. Katsaros, R. Kubert, and G. Gallizo, "Building a service-oriented monitoring framework with rest and nagios," in *2011 IEEE International Conference on Services Computing (SCC)*, 2011, pp. 426–431.

[101] R. B. Uriarte and C. B. Westphall, "Panoptes: A monitoring architecture and framework for supporting autonomic clouds," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–5.

[102] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: online monitoring and analytics for managing large scale data centers," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2012, pp. 141–150.

[103] J. S. Ward and A. Barker, "Self managing monitoring for highly elastic large scale cloud deployments," in *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing*, ser. DIDC '14. New York, NY, USA: ACM, 2014.

[104] G. Copil, D. Trihinas, H.-L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. Dikaiakos, *Service-Oriented Computing: 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014. Proceedings*. Berlin, Heidelberg: Springer, 2014, ch. ADVISE – A Framework for Evaluating Cloud Service Elasticity Behavior, pp. 275–290.

[105] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[106] CityPulse Smart City Project, http://iot.ee.surrey.ac.uk:8080/datasets.html.

[107] Open Data Institute - BigData Italy City of Trento, http://theodi.fbk.eu/openbigdata/.

[108] A. Silberstein, R. Braynard, and J. Yang, "Constraint chaining: On energy-efficient continuous monitoring in sensor networks," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142492

[109] S. Clayman, R. Clegg, L. Mamatas, G. Pavlou, and A. Galis, "Monitoring, aggregation and filtering for efficient management of virtual networks," in *Proceedings of the 7th Int. Conference on Network and Services Management*, 2011, pp. 234–240.

[110] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[111] M. Andreolini, M. Colajanni, M. Pietri, and S. Tosi, "Adaptive, scalable and reliable monitoring of big data on clouds," *Journal of Parallel and Distributed Computing*, vol. 79–80, pp. 67–79, 2015, special Issue on Scalable Systems for Big Data Management and Analytics.

[112] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh, "Lance: optimizing high-resolution signal collection in wireless sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 169–182.

[113] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing historical information in sensor networks," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 527–538.

[114] S. Kim, S. Kim, and D. S. Eom, "Rssi/lqi-based transmission power control for body area networks in healthcare environment," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 3, pp. 561–571, May 2013.

[115] H. Kim, C. V. Hoof, and R. F. Yazicioglu, "A mixed signal ECG processing platform with an adaptive sampling ADC for portable monitoring applications," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2011, pp. 2196–2199.

[116] J. M. da Silva, C. Oliveira, B. Mendes, R. Dias, and T. Marques, "Design for dependability and autonomy of a wearable cardiac and coronary monitor," in *Digital System Design (DSD), 2015 Euromicro Conference on*. IEEE, 2015, pp. 567–570.

[117] Y. Chen and Y. Tsividis, "Design considerations for variable-rate digital signal processing," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 2479–2482.

[118] A. Graps, "An introduction to wavelets," *IEEE Computational Science and Engineering*, vol. 2, no. 2, pp. 50–61, Summer 1995.

[119] T. Eltoft, T. Kim, and T.-W. Lee, "On the multivariate laplace distribution," *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 300–303, May 2006.

[120] C. Alippi, G. Anastasi, M. D. Francesco, and M. Roveri, "An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 335–344, Feb 2010.

[121] L. Fan and L. Xiong, "Real-time aggregate monitoring with differential privacy," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12. New York, NY, USA: ACM, 2012, pp. 2169–2173.

[122] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, July 2005.

[123] R. E. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, 1960.

[124] A. Jain and E. Y. Chang, "Adaptive sampling for sensor networks," in *Proceeedings of the 1st International Workshop on Data Management for Sensor Networks: In Conjunction with VLDB 2004*, ser. DMSN '04. New York, NY, USA: ACM, 2004, pp. 10–16.

[125] S. Tata, M. Mohamed, and A. Megahed, "An optimization approach for adaptive monitoring in iot environments," in *2017 IEEE International Conference on Services Computing (SCC)*, June 2017, pp. 378–385.

[126] R. J. Hyndman and G. Athanasopoulos, "Forecasting: principles and practice," *Online textbook*, 2013.

[127] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri, "Macrobase: Prioritizing attention in fast data," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 541–556.

[128] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, "Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041 – 2056, 2013.

[129] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang, "Star: Self-tuning aggregation for scalable monitoring," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 962–973.

[130] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 563–574.

[131] D.-Y. Kim, Y.-S. Jeong, and S. Kim, "Data-filtering system to avoid total data distortion in iot networking," *Symmetry*, vol. 9, no. 1, 2017.

[132] M. Du and F. Li, "Atom: Efficient tracking, monitoring, and orchestration of cloud resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2172–2189, Aug 2017.

[133] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[134] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," *SIGMOD Rec.*, vol. 45, no. 1, pp. 69–76, Jun. 2016.

[135] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *In SIAM International Conference on Data Mining*, 2010.

[136] Y. Matsubara and Y. Sakurai, "Regime shifts in streams: Real-time forecasting of co-evolving time sequences," in *Proceedings of the 22th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

[137] S. González-Valenzuela, M. Chen, and V. C. Leung, "Mobility support for health monitoring at home using wearable sensors," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 4, pp. 539–549, 2011.

[138] Luis F. Chiroque, "AdaM sampling implemented in R," https://github.com/luisfo/adaptivegraphmonitoring, 2016.

[139] J. Traub, S. Breß, T. Rabl, A. Katsifodimos, and V. Markl, "Optimized on-demand data streaming from sensor nodes," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 586–597.

[140] A. G. Barnett and A. J. Dobson, *Introduction to Seasonality*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 49–74.

[141] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Non-linear mining of competing local activities," in *25th Int'l Conference on World Wide Web (WWW '16)*, 2016, pp. 737–747.

[142] S. W. Roberts, "Control Chart Tests Based on Geometric Moving Averages," *Technometrics*, vol. 1, no. 3, 1959.

[143] K. M. Carter and W. W. Streilein, "Probabilistic reasoning for streaming anomaly detection," in *Statistical Signal Processing Workshop (SSP), 2012 IEEE*. IEEE, 2012, pp. 377–380.

[144] C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5 – 10, 2004.

[145] Y. Xiang, K. Li, and W. Zhou, "Low-rate ddos attacks detection and traceback by using new information metrics," *IEEE Transactions on Information Forensics and Security*, vol. 6, 2011.

[146] U. Fano, "Ionization yield of radiations. ii. the fluctuations of the number of ions," *Phys. Rev.*, vol. 72, pp. 26–29, Jul 1947.

[147] Java Microbenchmark Kit, https://goo.gl/zRTQDv.

[148] I. Shafer, K. Ren, V. N. Boddeti, Y. Abe, G. R. Ganger, and C. Faloutsos, "RainMon: An Integrated Approach to Mining Bursty Timeseries Monitoring Data," in *Proceedings of the 18th ACM SIGKDD International Conference*, ser. KDD '12. NY, USA: ACM, 2012, pp. 1158–1166.

[149] SANS Technology Institute, https://isc.sans.edu/port.html.

[150] Fitbit Data Extractor, https://github.com/dtrihinas/FitbitDataExtractor.

[151] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Computer Architecture, 2000. Proceedings of the 27th Int. Symposium on*, June 2000, pp. 83–94.

[152] H. Hiilloskorpi, M. Pasanen, and M. Fogelholm, "Use of heart rate to predict energy expenditure from low to high activity levels," *Int'l. journal of sports medicine*, vol. 24, 2003.

[153] Jesse Berst, "Dublin uses real-time data to reduce congestion," http://goo.gl/kGIvJJ, 2013.

[154] Sandy Ryza, "Apache Spark Streaming Performance Tuning," http://goo.gl/jPi728, Mar 2015.

[155] A. D. Sarma, F. N. Afrati, S. Salihoglu, and J. D. Ullman, "Upper and Lower Bounds on the Cost of a Map-reduce Computation," *Proceedings of the VLDB Endowment*, vol. 6, no. 4, pp. 277–288, 2013.

[156] C. Hu, Y. Hu, W. Xu, P. Shi, and S. Fu, *Understanding Popularity Evolution Patterns of Hot Topics Based on Time Series Features*. Cham: Springer International Publishing, 2014, pp. 58–68.

[157] S. Gelper, R. Fried, and C. Croux, "Robust forecasting with exponential and holt–winters smoothing," *Journal of forecasting*, vol. 29, no. 3, pp. 285–300, 2010.

[158] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[159] D. Barry and J. A. Hartigan, "A bayesian analysis for change point problems," *Journal of the American Statistical Association*, vol. 88, no. 421, pp. 309–319, 1993.

[160] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 289–296.

[161] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, vol. 107, no. 500, 2012.

[162] Y. Luo, Z. Li, and Z. Wang, "Adaptive cusum control chart with variable sampling intervals," *Computational Statistics & Data Analysis*, vol. 53, no. 7, pp. 2693 – 2701, 2009.

[163] P. Granjon, "The cusum algorithm-a small review," 2013.

[164] W. Jiang, L. Shu, and D. W. Apley, "Adaptive cusum procedures with ewma-based shift estimators," *IIE Transactions*, vol. 40, no. 10, pp. 992–1003, 2008.

[165] J. Thones, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan 2015.

[166] Martin Fowler, "Microservices - a definition of this new architectural term," 2014.

[167] Unicorn EU co-funded project, http://unicorn-project.eu/.

[168] Spring cloud netflix micro-services, https://cloud.spring.io/spring-cloud-netflix/.

[169] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal Networks*. Springer, 2013, pp. 15–40.

[170] A. Galati, V. Vukadinovic, M. Olivares, and S. Mangold, "Analyzing temporal metrics of public transportation for designing scalable delay-tolerant networks," in *Proceedings of the 8th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, ser. PM2HW2N '13. New York, NY, USA: ACM, 2013, pp. 37–44.

[171] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[172] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, "Temporal distance metrics for social network analysis," in *Proceedings of the 2Nd ACM Workshop on Online Social Networks*, ser. WOSN '09. New York, NY, USA: ACM, 2009, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/1592665.1592674

[173] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 721–732, May 2014. [Online]. Available: http://dx.doi.org/10.14778/2732939.2732945

[174] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Efficient hyperparameter optimization and infinitely many armed bandits," *CoRR*, vol. abs/1603.06560, 2016. [Online]. Available: http://arxiv.org/abs/1603.06560

[175] R. Kang, C. Wang, P. Wang, Y. Ding, and J. Wang, "Pattern Matching with Adaptive Granularity Over Streaming Time Series," *ArXiv e-prints*, Oct. 2017.

[176] N. Chen, Z. Li, and Y. Ou, "Multivariate exponentially weighted moving-average chart for monitoring poisson observations," *Journal of Quality Technology*, vol. 47, no. 3, p. 252, 2015.

[177] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.