Low-Cost Adaptive Monitoring Techniques for the Internet of Things

Demetris Trihinas, George Pallis, Marios D. Dikaiakos

Abstract—Internet-enabled physical devices with "smart" processing capabilities are becoming the tools for understanding the complexity of the global inter-connected world we inhabit. The Internet of Things (IoT) churns tremendous amounts of data flooding from devices scattered across multiple locations to the processing engines of almost all industry sectors. However, as the number of "things" surpasses the population of the technology-enabled world, real-time processing and energy-efficiency are great challenges of the big data era transitioning to IoT. In this article, we introduce a lightweight adaptive monitoring framework suitable for smart IoT devices with limited processing capabilities. Our framework, inexpensively and in place dynamically adjusts the monitoring intensity and the amount of data disseminated through the network based on a low-cost adaptive and probabilistic learning model capable of capturing at runtime the current evolution and variability of the data stream. By accomplishing this, energy consumption and data volume are reduced, allowing IoT devices to preserve battery and ease processing on cloud computing and streaming services. Experiments on real-world data from cloud services, internet security services, wearables and intelligent transportation services, show that our framework achieves a balance between efficiency and accuracy. Specifically, our framework reduces data volume by 74%, energy consumption by at least 71%, while maintaining accuracy always above 89%.

Index Terms—Edge Computing, Internet of Things, Big Data, Monitoring, Cloud Computing

1 Introduction

Recent advances in microelectronics, telecommunications and data mining have led to a growing adoption of *smart* devices, which impact how we live and work [1]. These devices capture and exchange data streams with other network-enabled devices and services, forming what is known as the Internet of Things (IoT). From smart transportation and home appliances, to retail innovations, surveillance, and manufacturing, we are starting 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 [2]. 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 IoT devices while cloud services consuming IoT data are constantly overwhelmed and struggle to be effective [3] [4].

As IoT spreads across almost all industries it triggers a massive influx of big data [5]. According to Gartner [2], 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. Despite attempts to augment IoT devices with the power of the cloud there still exist numerous inhibitors masked under constant data processing and dissemination [6]. Even powerful organisations equipped with high-performance processing engines reach their limits as the volume and velocity of monitoring data keeps increasing [7]. In addition, if IoT devices are battery-powered then intense processing leads to increased energy consumption and thus, less battery life [8]. Table 1 presents such a case where power consumption triples as processing and dissemination load are added to an IoT device. This is even more evident in IoT settings comprised of mobile edge devices equipped with small batteries [4] [9]. For example, the new generation of activity tracking 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 reason battery life on such devices has dropped from 5-7 to 3-5 days [4]. Therefore, it is no wonder why taming data volume and velocity, as well as, energy efficiency, are considered as great challenges to overcome in IoT [10].

If a degree of inaccuracy can be tolerated, the remedy to reduce data volume and network traffic between IoT devices and cloud services, is to apply approximate and adaptive monitoring techniques [11]. *Adaptive sampling* is the process of dynamically adjusting the sampling rate based on a runtime estimation model following the current data evolution, such that when stable phases in the data stream are detected, the sampling rate is reduced to ease processing and energy consumption. In turn, when the evolution of the data fluctuates in time, the sampling rate is increased to immediately capture event violations. Filtering is the process of suppressing metric value dissemination when consecutive values differ less than a range of values. Hence, energy required to transmit values over the network is reduced in favor of exact precision. In turn, *adaptive filtering* is the process of dynamically adjusting the filter to follow the monitoring stream variability without requiring for users to pre-determine fixed filter properties. At the same time, the volume and velocity of data reaching IoT services is regulated to ease processing. Despite advances in the field, current adaptive monitoring techniques are not tailored for the challenges of IoT, since they either: (i) require excessive profiling to configure optimal framework param-

• *Demetris Trihinas, George Pallis and Marios D. Dikaiakos are with the Department of Computer Science, University of Cyprus. e-mail: { trihinas, gpallis, mdd }@cs.ucy.ac.cy*

eters, a task difficult for users; (ii) present large runtime footprints reducing the benefits of introducing adaptiveness in the end; (iii) fail to acknowledge abrupt and transient shifts in the data evolution or assume that once determined there will be no shifts in the data value distribution; or (iv) require coordination from server-side components.

To address these challenges we introduce the Adaptive Monitoring framework[1]. AdaM is a lightweight framework developed for software agents and IoT devices. AdaM, *inexpensively* and *in place*, dynamically adjusts the monitoring intensity and the amount of data disseminated through the network based on a runtime estimation model capturing the current data evolution and variability. By accomplishing this, energy consumption and data volume are reduced, allowing IoT devices to preserve battery and ease processing at data consuming services, while still preserving accuracy. To achieve this, AdaM incorporates two algorithms, one for adaptive sampling and one for adaptive filtering. Both algorithms provide estimations, adjusting the sampling rate and the filter range 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 model at runtime by extending our model that was limited to adaptive parameter weighting [12]. To this end, we introduce trend detection so that our algorithms immediately identify abrupt transient changes in the data evolution and overcome time lagging effects in the estimation process. Most importantly, AdaM runs on the source device without the need of any additional coordination from central management endpoints or excessive profiling to determine framework parameters.

A thorough evaluation and comparison to other IoT adaptive techniques is conducted with real-world data from cloud and internet security services, wearables and intelligent transportation services. To the best of our knowledge, this is the first study extending the evaluation to include a performance and energy consumption comparison. Results, show that AdaM reduces data volume by 74%, energy consumption by at least 71%, while maintaining accuracy above 89%. Also, we show that cloud services consuming IoT data can benefit, in terms of lower monitoring costs and achieve greater scalability when devices use AdaM in their software core.

The rest of this article is organised as follows: Section 2 presents the related work. Section 3 the problem statement. Section 4 introduces our framework. Section 5 presents a thorough evaluation, while Section 6 concludes this article.

2 Related Work

Edge-mining is a term coined to reflect data processing on devices scattered across the logic extremes of a network. Adjusting the monitoring intensity and amount of data transmitted by an IoT device, is a form of edge-mining [3]. To date, a number of cloud monitoring tools claim to be suitable for IoT devices as they run on limited resources [13] [14]. These tools assume metric collection is somewhat trivial with server monitoring (e.g., CPU, memory) merely the task of parsing OS files (e.g., `/proc/*` for Linux). However, this assumption is far from true when the monitoring task must collect external stimulus and perform costly analysis, as in the case of heartbeat monitoring. Thus, tools without self-adaptive capabilities to reduce energy consumption and network traffic are unsuitable for IoT. Although hardware-specific techniques have been developed in the past [15], trends show IoT is moving towards software-defined realms to quickly provision, manage and monitor IoT services with adaptivity envisioned as a manageable asset for monitored services [6]. In what follows, are a number of software-defined techniques for adaptive sampling and filtering.

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

Raspberry Pi 2 Model B	**Power**
Idle state	420mA (2.1W)
Max CPU load	800-1100mA (4W)
Max CPU load + disk I/O	900-1200mA (4.5W)
Max CPU load + disk I/O + metric dissemination over the network	1250-1400mA (6.25W)

TABLE 1: IoT Device (Raspberry Pi) Power Consumption

2.1 Adaptive Sampling

Rastogi et al. [16] propose a Discrete Fourier Transformation (DFT) for differential privacy which perturbs coefficients of a timeseries, reconstructing afterwards a k-approximate version of the original signal from the inverse DFT. However, while at the receiver-side there appears to be adaptivity in the data stream, due to discarding coefficients, in reality all values are still collected. Chowdhury et al. [17] introduce a framework adjusting the monitoring intensity if the current metric evolution violates certain user-defined policies. Andreolini et al. [18] introduce a similar approach, with the difference that the current stream variability is used as a more suitable mechanism to discriminate stable from variable monitoring states. On the other hand, Meng et al. [7] propose a violation-likelihood detection approach for cloud networks based on the probability of misdetecting a violation between two consecutive datapoints. Thus, the sampling rate is increased when metric values approach a user-defined threshold; otherwise, the sampling period is restored to a fixed rate. Nonetheless, for the aforementioned techniques to be applicable a number of parameters and policies must be pre-defined by the user and cannot change at runtime, thus assuming the data stream value distribution will always remain relevant.

Fan et al. [19] introduce FAST, an adaptive framework for differential privacy which computes an estimate of the sampling rate based on the adjustment given by a PID controller fed by the current estimation error, the time intervals between previously collected metric values and a given inaccuracy budget. FAST's adaptive sampling is aggressive producing large sampling periods as the purpose of its development is applying costly user-differential privacy on each interval. Thus, it uses a Kalman filter to generate estimates for non-sampled intervals to optimize accuracy under the differential privacy constraint. However, as the Kalman filter is used to filter signal noise, this approach does not always 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 is still required to increase accuracy. In contrast, Gaura et al. [3] propose L-SIP, an adaptive algorithm specifically tailored for IoT devices. L-SIP encodes the state of an IoT device as a point in time with attributes the metric value and its rate of change. This is performed by using an exponential weighted moving average (EWMA), with the sampling rate increasing if the difference between the observed and estimated value are

Notation	Description
$s_i(t, v)$	The i^{th} sample of a sequence comprising a metric stream with a timestamp t_i and value v_i
$M = \{s_i\}_{i=0}^{n}$	A metric stream of a monitoring source comprised of collected samples with $i = 0, 1, ..., n$ and $n \to \infty$
T_i	Periodicity to collect s_i such that $T_i = t_i - t_{i-1}$ and $T_i \subseteq \mathbb{Z}^+$ restricted to $T_i \in [T_{min}, T_{max}]$
R_i	Filter range to decide if s_i, assuming a window $W = [v_{i-1} - R_i, v_{i-1} + R_i]$, should be filtered ($v_i \in W$)
err	Difference between metric stream M and reconstructed stream M' via an adaptive technique for a range of sample values e.g. $err = \sum \|v_i - v'_i\|, i \geq 0$
$\rho(M)$	Function containing information to characterise evolution of metric stream M (e.g. moving average)
$q(M)$	Function characterising variability of metric stream M (e.g. the coefficient of variation)
γ	Max acceptable imprecision for reconstructed metric stream via an adaptive technique ($\gamma \in [0, 1]$)

TABLE 2: Table of Notations

larger than a user-defined estimation error. L-SIP is an interesting lightweight algorithm suitable for adaptive sampling on IoT devices, but it is slow to react to highly transient and abrupt fluctuations in the evolution of the monitoring data.

2.2 Adaptive Filtering

In [20] we introduce a monitoring tool that autonomously adjusts the metric filter range depending on the percentage of values previously filtered. Du et al. [21] show that network traffic is further reduced if monitoring tools apply filtering by sending only the median from a window of collected data, when the current datapoint has not changed more than a predefined threshold. Although interesting, both assume that no distribution shifts will occur in the data evolution at runtime.

Deligiannakis et al. [22] suggest a traffic reduction strategy for sensing networks. Their strategy suggests buffering large amounts of metric values at each node, and rather than sending the total of the buffer contents, it transmits a base signal of fewer values (wavelet) which is then used to reconstruct the original signal. However, a large portion of the signal must be made available and stored on the device to provide an estimation. On the other hand, Silberstein et al. [23] propose an monitoring tool for sensor networks, where metric dissemination is suppressed if neighbouring sensors present similar values. The proposed mechanism reduces energy consumption inferred by constant metric transmission but with the caveat that sensors must have knowledge of the entire network topology. Finally, Olston et al. [24] propose a server-side approach for filtering continuous data streams. This approach involves specifying a precision requirement with data sources sending updates to a central server 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. However, filter adjustment is only feasible if data generated at different sources follow a certain similar pattern on all nodes.

In summary, all presented solutions limit their scope to either adaptive sampling or filtering. No solution is capable of estimating and adapting the monitoring intensity of an IoT device to follow the actual data evolution in time, especially, when highly abrupt metric fluctuations are observed.

3 Problem Statement

Before introducing the AdaM framework, it is important to understand the background for each challenge and their respective problem definition. To ease readability, Table 2 presents the notation used throughout the article.

3.1 Preliminaries

We define a *metric stream* $M{=}\{s_i\}_{i=0}^{n}$ published by a monitoring source (e.g., IoT device) to a receiving entity (e.g., cloud service), as a large stochastic sequence of independent and identically distributed (i.i.d) samples s_i, where $i = 0, 1, ..., n$ and $n \to \infty$. Each sample s_i is a tuple (t_i, v_i) described, at the minimum, by a timestamp t_i and a univariate metric value v_i. A sample may include a set of other attributes (e.g., location), although for brevity, when describing a sample we omit these attributes without loss of generality. In turn, no assumptions are made for the number of generated samples which depend solely on the task assigned to the monitoring source. Therefore, receiving entities have no control on the input rate, with dissemination scheduled by the monitoring source based on a *push-based* metric delivery protocol [20].

3.2 Adaptive Sampling Problem Definition

For a metric stream M, *periodic sampling* is the process of triggering the 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 interval (e.g., 1s, 10s, 1min), the i^{th} sample ($i \in I$) is collected at time $t_i = i \cdot T$. This process is widely adopted by monitoring tools due to its simplicity [14] [25]. In this work, we argue that using a fixed T on battery-powered devices features a number of constraints. Specifically, it is both resource and energy consuming to collect periodically samples, especially when consecutive metric values (e.g. v_i, v_{i-1}, v_{i-2}, ...) do not vary. For example, consider the metric stream introduced in Figure 1. If a small T is utilized (e.g. T=1s), a high volume of data is generated and must be distributed through the network to be processed or stored for further use. If, instead, a large period is used (e.g. T=10s), then sudden events or significant insights may remain undetected. In general, because sampling depends on the data and its evolution in time, we argue that a fixed sampling period is not effective, as metrics and insights are only useful if collected in meaningful time intervals.

To accommodate the above challenges, *adaptive sampling* is used. Adaptive sampling is the process of dynamically adjusting the sampling period T_i, based on some estimation model, denoted as $\rho(M)$, capturing runtime information of the metric stream evolution. Assume s_i to be the latest sample 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 (Figure 1). 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. When the metric stream values are relatively stable, the sampling period should be increased and when the values fluctuate, it should be decreased or restored to a minimum value. 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 s_{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

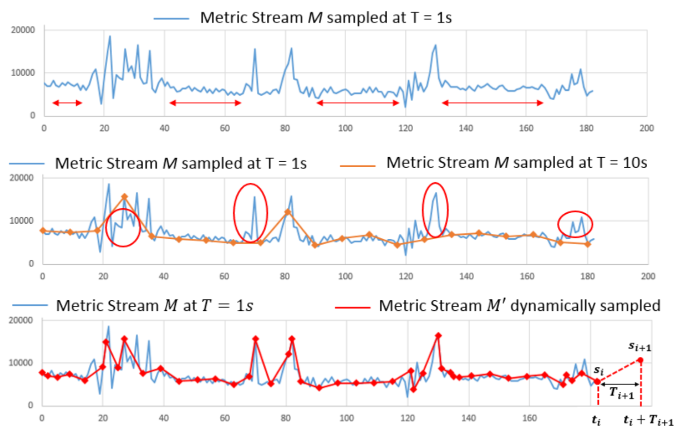


Fig. 1: Periodic and Adaptive Sampling

imprecision γ, for the range $t \in [t_i, t_i + T]$. Thus, the problem is summarized with the following equation:

$$T^* = \arg\max_T \{f(s, T, \rho(M), err(M, M')) \mid err < \gamma, \quad T \in [T_{min}, T_{max}]\} \tag{1}$$

Intuitively, as $\gamma \to 0$ the metric stream $M' \to M$. However, 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.

3.3 Adaptive Filtering Problem Definition

For a metric stream M, *filtering* is the process of suppressing metric value dissemination when consecutive values differ less than a range of values. Hence, energy required by the IoT device to transmit metrics over the network, as well as, the velocity at which data arrive to IoT services, are reduced while adhering to certain user-defined accuracy guarantees. Thus, a monitoring source with filtering capabilities does not transmit the latest collected sample if its value has not "changed" since last reported. How much "change" is required, depends on the type of filter applied, while metric receivers assume the values of any unreported metrics remain unchanged.

Depending on the type of *filter*, the number of samples filtered may vary. For example, suppose a *fixed filter range* approach is followed. The sample s_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 data evolution and that it will not change in the future. Otherwise, there is no guarantee that any values will be filtered at all. For instance, let us consider the metric stream M, presented in Figure 2, where the filter range R is enabled once and set to a fixed value. From Figure 2, we observe that a phase of high variability is both preceeded 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, no samples 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 set by the user.

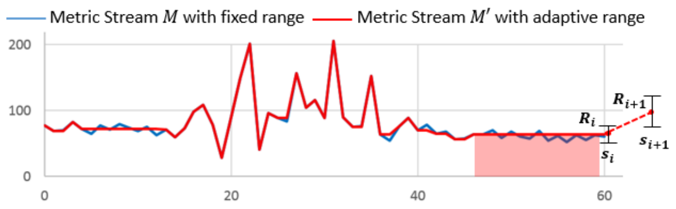


Fig. 2: Fixed Range $R = 1$ and Adaptive Filtering $R \in [0, 3]$

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 2. 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 s_i, the goal of adaptive filtering is to provide a *filtering function* $f(\cdot)$ capable of finding the maximum R to apply on s_{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 γ. Hence, the problem is summarized with the following equation:

$$R^* = \arg\max_R \{f(s, R, q(M), err(M', M)) \mid err < \gamma, \quad R \in (0, R_{max}]\} \tag{2}$$

Similar to adaptive sampling, as $\gamma \to 0$ the metric stream $M' \to M$. However, the filter $R \to 0$ which defeats even the purpose of filtering with a fixed range R. Therefore, to reduce network traffic an adaptive filtering technique, at any given time, is likely to select a filter range where $R > 0$, which is only applicable if a degree of imprecision is tolerable.

4 The AdaM Framework

The AdaM Framework provides model-based adaptive monitoring, by dynamically adjusting the monitoring intensity and the volume of data disseminated to cloud services, based on extracted runtime knowledge capturing the metric stream evolution and variability. To achieve this, AdaM incorporates low-cost adaptive and probabilistic learning algorithms for adaptive sampling and filtering. AdaM is developed in java as a lightweight framework embeddable in the software core of IoT devices (e.g., raspberry Pi, android devices). It can also be ported to other popular programming frameworks (e.g., python, R) as it has no external source code dependencies.

Figure 3 depicts a high-level 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*. When the *Sensing Unit* collects a new metric sample (s_i), it is passed through the API to the *Low-Cost Approximate Stream Estimation* module. This module updates a local reference estimation model capturing the metric stream evolution and variability, while also maintaining in the *Model Base* online statistics (e.g., mean, standard deviation) used by AdaM modules and may also be of interest to users and metric stream receiving entities. In turn, to assist the estimation process to better follow the current monitoring stream evolution, trend detection is also used to reduce any time lagging effects in the estimation

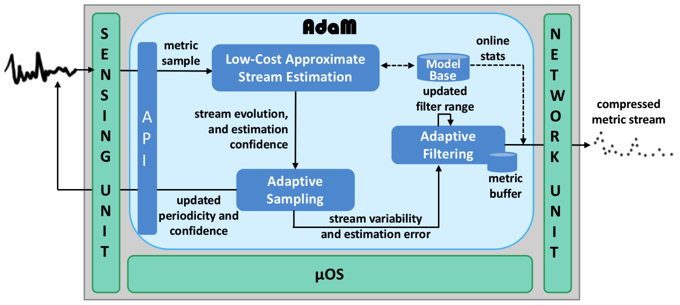


Fig. 3: AdaM Framework embedded in IoT Device

process, as IoT data such as human body indicators and environmental data, present such behavior [5].

After updating the estimation model, the *Adaptive Sampling* module will use the current updated metric stream evolution and trend 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 sample (s_{i+1}) and return to an idle state. If adaptive filtering is enabled, the current sample is forwarded to *Adaptive Filtering* to decide if the sample 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 sample is not filtered, it is stored in the *metric buffer* until metric dissemination is triggered by the *Network Unit* of the IoT device. Also, along with the current measurement and AdaM's estimations, a set of statistics describing the current state of the metric stream are disseminated to receiving entities, including the current mean, variance, and the confidence and prediction error of AdaM's estimations.

4.1 Low-Cost Approximate Stream Estimation

The following considerations were taken into account while developing the estimation process. First, the estimation process must be lightweight. Applying an adaptive algorithm is only meaningful if the process is done inexpensively, meaning the cost of applying an adaptive technique is much less than actually collecting samples and later discarding them. In turn, the process must be performed, in place, on the monitoring source itself. This eliminates the need to distribute values over the network to management endpoints for it to be applicable. Moreover, it must react to highly abrupt fluctuations in the metric stream while the estimation process must ensure the given accuracy guarantees are obeyed so users are timely notified of abnormalities and sudden events. In addition, extensive profiling to identify optimal parameter configuration must not be required. Hence, we base our approach such that the estimation model is maintained in constant O(1) time and space, thus satisfying the above requirements, which require a low-cost estimation model able to run on IoT devices with limited processing capabilities. In turn, adjustments are made only when the estimation process is able to capture the metric evolution within the given accuracy guarantees. Algorithm 1 presents our adaptive estimation model. At first, we compute the distance δ_i between the current two consecutive values:

$$\delta_i = |v_i - v_{i-1}| \tag{3}$$

The distance δ_i is used to update the local reference runtime evolution of the metric stream $\rho(M)$. We compute the

Algorithm 1 Adaptive Estimation Model

```
Input: current sample $s_i$ 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. 3)
        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. 7)
        update estimation model for next time interval
 5:     $\mu_i \leftarrow \text{updPEWMAwithTrend}(P_i, \delta_i, x_i)$           (eq. 9)
 6:     $x_i \leftarrow \text{updHoltTrend}(\mu_i)$                            (eq. 8)
        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. 10)
        compute current estimation confidence
 8:     $c_i \leftarrow \text{calcConfidence}(\sigma_i, \hat{\sigma_i})$                  (eq. 11)
 9: else
10:     $\hat{\delta}_{i+1} = \mu_i \leftarrow v_0,\ \hat{\sigma}_{i+1} \leftarrow 0$    //init values
11: end if
12: return estModel($\hat{\delta}_{i+1}, \hat{\sigma}_{i+1}, c_i, \sigma_{err}$)
```

current metric evolution by using a moving average, denoted as μ_i (steps 2-7). This provides an estimation of the metric stream evolution 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, an increase in the sampling period can be considered. Moving averages provide one-step ahead predictions, are easy to compute and can be calculated with previous value knowledge. Equation 4 presents an example of a cumulative Simple Moving Average (SMA) where values of a sliding window are aggregated evenly:

$$\mu_i = \frac{\delta_i + (i-1)\mu_{i-1}}{i},\ i \geq 1 \tag{4}$$

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 ($\alpha \in [0,1]$) is introduced to decrease exponentially the effect of older values, as presented in Equation 5. 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 [26]. Specifically, the EWMA 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 δ_i's which affect the accuracy of an adaptive technique.

$$\mu_i = \begin{cases} \delta_i, & i = 1 \\ \alpha\mu_{i-1} + (1-\alpha)\delta_i, & i > 1 \end{cases} \tag{5}$$

Therefore, we adopt a Probabilistic EWMA to dynamically adjust the weighting based on the probability density of the given observation. The PEWMA sufficiently acknowledges abrupt transient changes, adjusting quickly to long-term shifts in the metric evolution and when incorporated in the estimation process (steps 2-7), it requires no parameterization, scaling to numerous samples. Equation 6 presents the PEWMA where instead of fixed weighting, we introduce a probabilistic and adaptive weighting factor, denoted as $\tilde{a_i} = \alpha(1 - \beta P_i)$. In this equation, the p-value, is the probability of the current δ_i to follow the modeled distribution of the metric stream evolution. In turn, β is a weight placed on P_i and as $\beta \to 0$ the PEWMA converges to a common EWMA[2].

$$\mu_i = \begin{cases} \delta_i, & i = 1 \\ \alpha(1-\beta P_i)\mu_{i-1} + (1 - \alpha(1-\beta P_i))\delta_i, & i > 1 \end{cases} \tag{6}$$

The logic behind probabilistic reasoning is that the current δ_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 δ_i's. In turn, if an "unexpected" value turns out to be a shift in the data evolution, as the probability kernel shifts, subsequent "unexpected" values are awarded with greater p-values, allowing them to contribute more to the estimation process. To satisfy these requirements we adopt a gaussian kernel $N(\mu, \sigma^2)$ and compare to a normal distribution to obtain the probability estimate (eq. 7) with $\delta_i - \hat{\delta_i}$ denoting the estimation error ϵ_i. The observation is normalized to a zero-mean and unit standard deviation (Z_i) to account for metrics with large data scales. While a Gaussian kernel is assumed, if prior knowledge of the distribution is available then only step 4 in Algorithm 1 must be updated. In line with this, P_i is initialized to 1, as Z_i is undefined for $\hat{\sigma_i} = 0$, which results in $\mu_1 = \delta_1$ irrespective of the considered probability kernel (eq. 6). Nonetheless, in any other time interval where $\hat{\sigma_i} = 0$ (extreme case), the p-value depends on the kernel definition with $P_i = 1/\sqrt{2\pi}$ for a gaussian kernel.

$$\begin{aligned} P_i &= \frac{1}{\sqrt{2\pi}} \exp(-\frac{Z_i^2}{2}) \\ Z_i &= \frac{\delta_i - \hat{\delta_i}}{\hat{\sigma_i}} \end{aligned} \tag{7}$$

While probabilistic 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. To fine-tune the estimation by capturing possible trends, we use Holt's Trend Method to estimate the current monotonic growth/decay in the metric stream evolution [27]. Equation 8 depicts how the trend, denoted as x_i, is updated at each time interval, where ξ is a smoothing weight in the range $[0, 1]$ with values near 1 denoting a preference to favor recent trends.

$$x_i = \begin{cases} \delta_i - \delta_{i-1}, & i = 2 \\ \xi\ (\mu_i - \mu_{i-1}) + (1-\xi)\ x_{i-1}, & i > 2 \end{cases} \tag{8}$$

Therefore, any lagging effects in the estimation process are reduced by boosting the moving average to the appropriate value base with an additive trend component as follows:

$$\mu_i = \tilde{a_i}(\mu_{i-1} + x_{i-1}) + (1 - \tilde{a_i})\delta_i \tag{9}$$

At this point, the metric stream evolution $\rho(M)$, encapsulated by $\hat{\delta}_{i+1}$ and $\hat{\sigma}_{i+1}$, is updated with only previous value knowledge and without repeatedly scanning the entire stream, as depicted in Equation 10. We note that θ_i is merely a local variable storing a temporal reference value in the sequence of calculations to compute online the current variance.

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

Algorithm 2 Adaptive Sampling

```
Input: imprecision $\gamma \in [0,1]$ given by user and confidence $c_i$
Output: $T_{i+1}$
Ensure: $\{T_{i+1} \mid T_{i+1} \in \mathbb{Z}^+$ 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. 13)
 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 $T_{i+1}$
```

$$\begin{aligned} \tilde{a_i} &\leftarrow \alpha(1 - \beta P_i) \\ \mu_i &\leftarrow \tilde{a_i} \cdot (\mu_{i-1} + x_{i-1}) + (1 - \tilde{a_i}) \cdot \delta_i \\ \theta_i &\leftarrow \tilde{a_i} \cdot \theta_{i-1} + (1 - \tilde{a_i}) \cdot \delta_i^2 \\ \hat{\delta}_{i+1} &\leftarrow \mu_i \\ \hat{\sigma}_{i+1} &\leftarrow \sqrt{\theta_i - \mu_i^2} \end{aligned} \tag{10}$$

Figure 4 depicts a comparison between AdaM and an estimation model limited to an EWMA. 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 observed standard deviation σ_i for δ_i and σ_{err} for ϵ_i (step 7). We note that, σ_{err} is not used in adaptive sampling but in adaptive filtering. Also, σ_{err} is a useful statistic since with σ_{err}, the current estimation δ_i and a multiplier K; high and low estimation control boundaries can be computed to perform outlier and change detection [11] as follows:

$$B_i^{high},\ B_i^{low} \leftarrow \delta_i \pm K \cdot \sigma_{err} \tag{11}$$

On the other hand, σ_i is used to compute the estimation process 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 observed standard deviation (eq. 12) used as our error evaluation metric denoting 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 satisfy the accuracy guarantees given by the user or rollback to a fixed approach when satisfactory estimations cannot be made. Therefore, as $\hat{\sigma_i} \to \sigma_i$ the confidence $c_i \to 1$.

$$c_i = 1 - \frac{|\hat{\sigma_i} - \sigma_i|}{\sigma_i} \tag{12}$$

4.2 Adaptive Sampling

After updating the estimation model, Algorithm 2 is used to perform adaptive sampling. In particular, the estimated sampling period T_{i+1}, is dependent to the current *sampling*

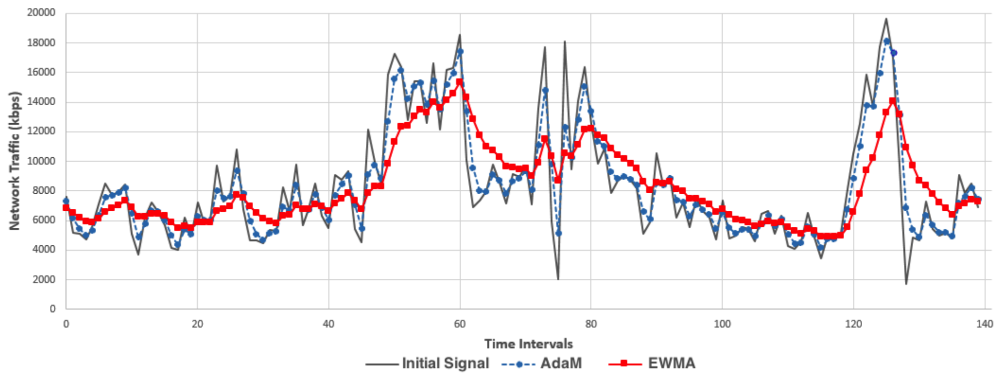


Fig. 4: A Comparison of AdaM Estimation Model to an Exponential Weighted Moving Average

Algorithm 3 Adaptive Filtering

Input: μ_i, σ_i and σ_{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 data 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. 14)
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. 15)
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 $R_{i+1}$
```

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* c_i, denoting the ability of the algorithm process to follow the metric stream evolution. Therefore, when the estimation model is "confident", the adaptive sampling algorithm will award larger sampling periods. Hence, in contrast to threshold-based techniques which adjust the sampling rate solely based on the sample value v_i, our approach considers the metric stream evolution, as well as, the confidence of the estimation. The reason for this lays in the failure of threshold-based techniques to detect variances in the metric stream when values are far from the threshold (e.g. $v_i << \tau$, where τ a user-defined threshold). Hence, events remain undetected such as in the case of low rate DDoS attacks. Similarly, stable phases with high values will fail to receive a sampling period decrement as well, as a violation is still probable.

Thus, having computed the current confidence (eq. 12), we then compare it to the acceptable user-defined imprecision, denoted as γ 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. 13). 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, then our adaptive sampling algorithm will rollback to the default sampling period T_{min} for the next sample s_{i+1}. Moreover, in contrast to stepwise techniques [3] [20] which adjust the sampling rate solely 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 $[T_{min}, T_{max}]$.

$$T_{i+1} = \begin{cases} T_i + \lambda \cdot (1 + \frac{c_i - \gamma}{c_i}), & c_i \geq 1 - \gamma \\ T_{min}, & else \end{cases} \tag{13}$$

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 γ, is the only parameter which is user-defined in the estimation process. Nonetheless, users are free to change: (i) λ 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 α and ξ, although as shown in the evaluation, α and ξ 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.

4.3 Adaptive Filtering

As with sampling, adaptive filtering must be lightweight and capable of running on the monitoring source. We base our approach such that the filter range R is dependent to the metric stream variability. The reason for this lays in the failure of stepwise techniques, which adjust R incrementally based on the number of samples previously filtered, in cases such as biosignal monitoring where precision is required in a small range of values [24]. For example, consider glucose monitoring where ignoring metric stream variability, even for a small 1% stepwise adjustment to a filter ($R_i \leftarrow R_{i-1} \pm 0.01 \cdot R_{i-1}$), will result in filtering out critical values.

Hence, AdaM's adaptive filtering algorithm (Algorithm 3) utilizes the Fano factor, denoted as F_i, and σ_{err}, to follow the current variability $q(M)$ of the metric stream M. In particular, the Fano factor ($F_i \geq 0$) is a normalized measure of the dispersion of a probability distribution, which is used to quantify whether a stream of samples are currently clustered ($F_i < 1$) or dispersed compared to a statistical model. The

Fano factor is calculated as the ratio of the variance σ^2 to the mean μ, as presented in Equation 14:

$$F_i = \frac{\sigma_i^2}{\mu_i} \tag{14}$$

To provide both the variances σ^2, σ_{err} and the mean μ of the latest samples of a metric stream, no additional computations are required, as both μ_i and σ_i are the exponentially weighted output of the PEWMA estimation provided by the adaptive estimation model. Intuitively, when σ_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 σ_{err} to the user-provided maximum tolerable imprecision, denoted as γ. If F_i indicates the metric stream is not dispersed and σ_{err} is less than γ, 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. 15). 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.

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 μ_i, σ_i and σ_{err} are the output of the runtime estimation model described in the previous section. Additional parameter configurable by users, is λ, a multiplier used as an aggressiveness indicator.

$$R_{i+1} = \begin{cases} R_i + \lambda \cdot (\frac{\gamma - F_i}{\gamma}), & F_i < 1 \; and \; \sigma_{err} < \gamma \\ R_i, & F_i < 1 \; and \; \sigma_{err} > \gamma \\ R_{min}, & else \end{cases} \tag{15}$$

5 Evaluation

We will compare AdaM's efficiency and accuracy to other adaptive techniques based on public data from cloud and internet security services, wearables and intelligent transportation systems. Next, a scalability evaluation is conducted based on two streaming services which benefit by using AdaM to lower data volume and velocity while preserving accuracy.

5.1 On Device Accuracy & Efficiency Evaluation

We will evaluate AdaM in comparison to three state-of-the-art adaptive techniques suitable for IoT devices:

- L-SIP [3], a linear framework 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 [17], 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 ϵ is under a user-defined imprecision value γ, and decreases it ($T_{i+1} \leftarrow T_i - T_{unit}$), when $\epsilon > \gamma$;
- FAST [19], 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 $\xi = 0.7$.

5.1.1 Traces, Testbeds and Evaluation Metrics

Table 3 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 [8a-8f] 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. The Fitbit Step and Heart 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 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 IoT device resources.

Accuracy: We evaluate an adaptive technique estimation accuracy by measuring the mean absolute percentage error (MAPE) from the original timeseries ground truth for each trace. Equation 16 depicts how the MAPE is calculated, where A_i is the actual value for the i^{th} sample and E_i is the estimated value. For each adaptive technique, when a sample is not collected, E_i is considered the last reported value.

$$MAPE_n = \frac{1}{n} \sum_{i=1}^{n} |\frac{A_i - E_i}{A_i}| \cdot 100\% \tag{16}$$

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 sample, if not filtered, is disseminated to the receiver-end; and (iii) *energy consumption*, based on the model adopted from [8] and presented in Equation 17. Power measurements were acquired with *powertop* and *wattch* [31]. In this model, P_{idle} denotes power in idle state; P_{cpu} processor power (including L1 cache and memory); τ_{cpu} the CPU time; $P_{i/o}$ power for I/O; τ_{wait} the I/O time; while P_{net} power consumed for disseminating samples over the network.

$$E = P_{idle} \cdot \tau_{idle} + P_{cpu} \cdot \tau_{cpu} + P_{io} \cdot \tau_{wait} + P_{net} \cdot \tau_{net} \tag{17}$$

5.1.2 Experiments

At first, we compare AdaM sampling (λ=1, λ=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 (α) 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 intervals. We note that FAST is not presented in this test, as it does not use a moving average. Also, FAST sampling is aggressive producing only a few sampling points and without applying a Kalman

Trace Name	**Origin**	**Description**
Memory Trace	Cloud Server	A memory trace of 105 samples originating from a physical server running the Java MicroBenchmark Kit.
CPU Trace	Cloud Server	A CPU trace from a physical server of 800 samples from the Carnegie Mellon RainMon project [28].
Disk IO Trace	Cloud Server	A Disk I/O trace from a physical server of 415 samples from the Carnegie Mellon RainMon project [28].
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 [29].
Step Trace	Wearable	A fitness step trace of 287 samples from a Fitbit Charge device [30].
Heart Trace	Wearable	A fitness heartrate trace of 287 samples from a Fitbit Charge device [30].
Calorie Trace	Wearable	A fitness calorie trace of 287 samples from a Fitbit Charge device [30].

TABLE 3: Datasets Used for Adaptive Techniques Performance and Accuracy Evaluation

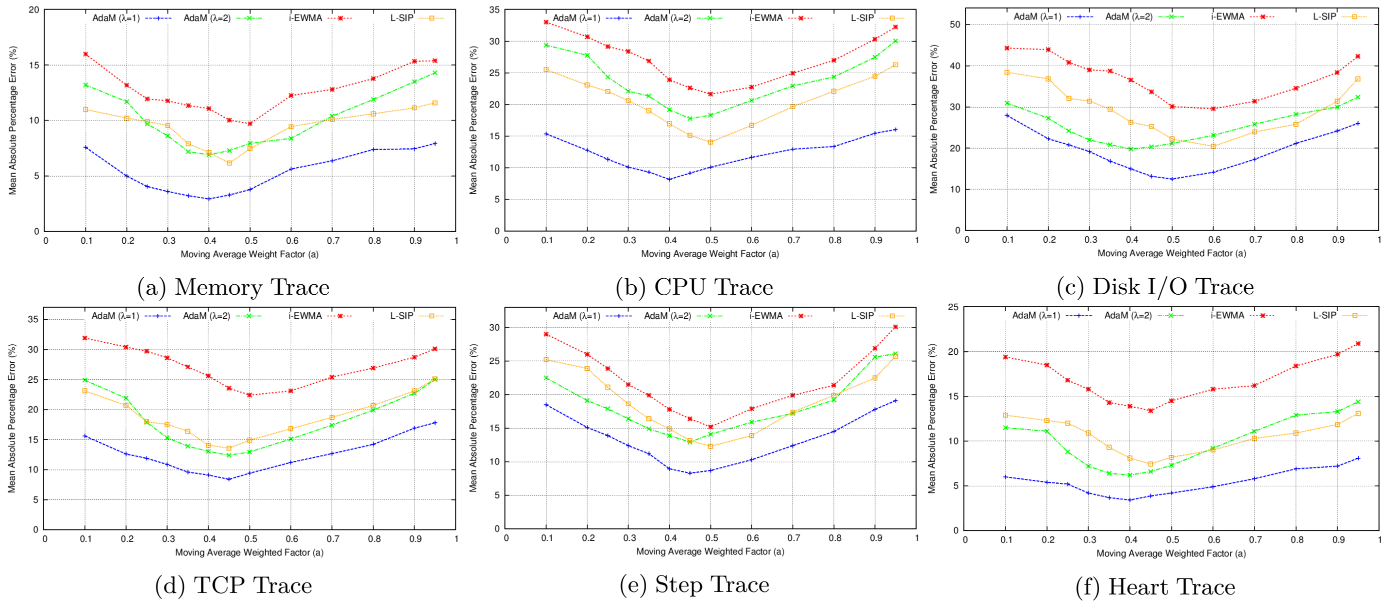


Fig. 5: Mean Absolute Percentage Error (MAPE) Comparison of Techniques Using a Moving Average Estimator

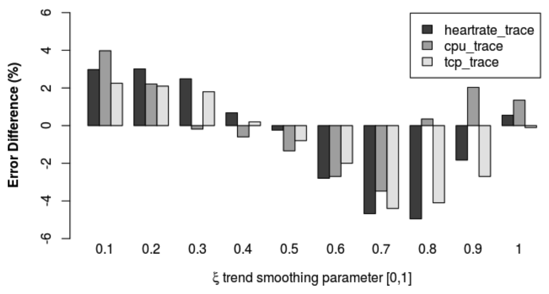


Fig. 6: Difference in MAPE when using Trend in Estimation

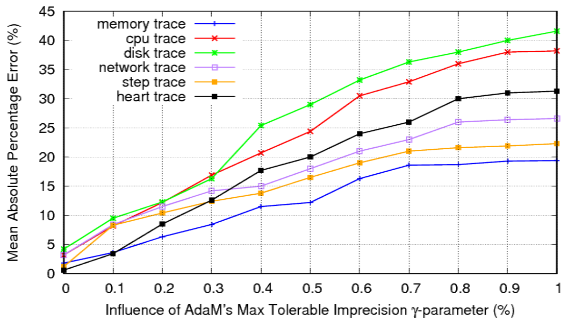
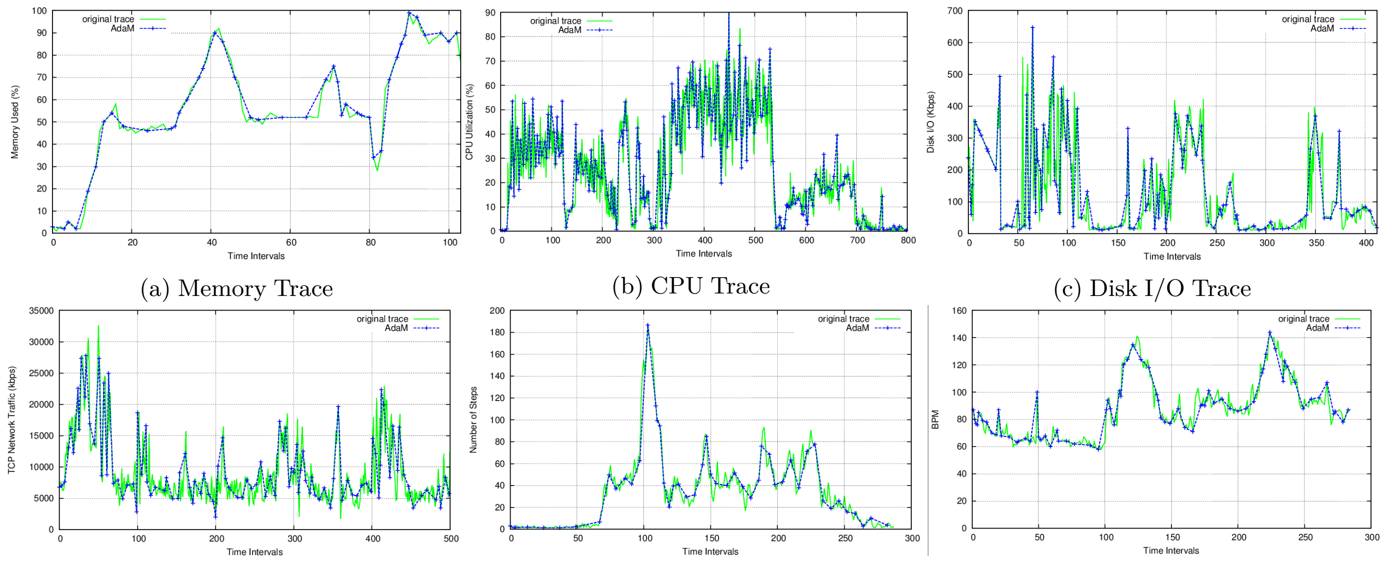


Fig. 7: Influence of Max Tolerable Imprecision to Estimation

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. 9d). Figures [5a-5f] depict the MAPE metric of each trace for the under comparison techniques.

First, we observe that AdaM (λ=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 slightly above (11%). Even, in a more aggressive setting (λ=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 α 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, AdaM accuracy is improved by shedding 2-5% of the error. This is depicted in Figure 6, where for a wide range of settings for the ξ parameter the error is reduced compared to not considering trends at all (Fig. 6 depicts only the three most challenging traces due to limited space). Thus, *the analysis conducted shows that for AdaM, profiling to find the optimal parameter settings is not always required if slight imprecision is acceptable.*

Next, we evaluate the influence of the maximum tolerable impression (γ), the only parameter that must be set by the user, to the overall estimation error (MAPE). Figure 7 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 γ-values which indicate a high imprecision tolerance.



(a) Memory Trace (b) CPU Trace (c) Disk I/O Trace

(d) TCP Trace (e) Step Trace (f) Heart Trace

Fig. 8: A Comparison of Traces Generated via AdaM Towards the Original Traces

Furthermore, Figure 8 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 imposed to the monitoring source. We note that for all subsequent experimentations AdaM incorporates trend detection in its estimation process with ξ 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. 9a) 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. 9c) 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. 9d). However, as the user-defined inaccuracy budget (γ) permits further approximation, accuracy is slightly sacrificed. This improves device efficiency by significantly reducing the network overhead, and consequently energy consumption (Fig. 9b), with a slightly 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 (γ=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. 9b).*

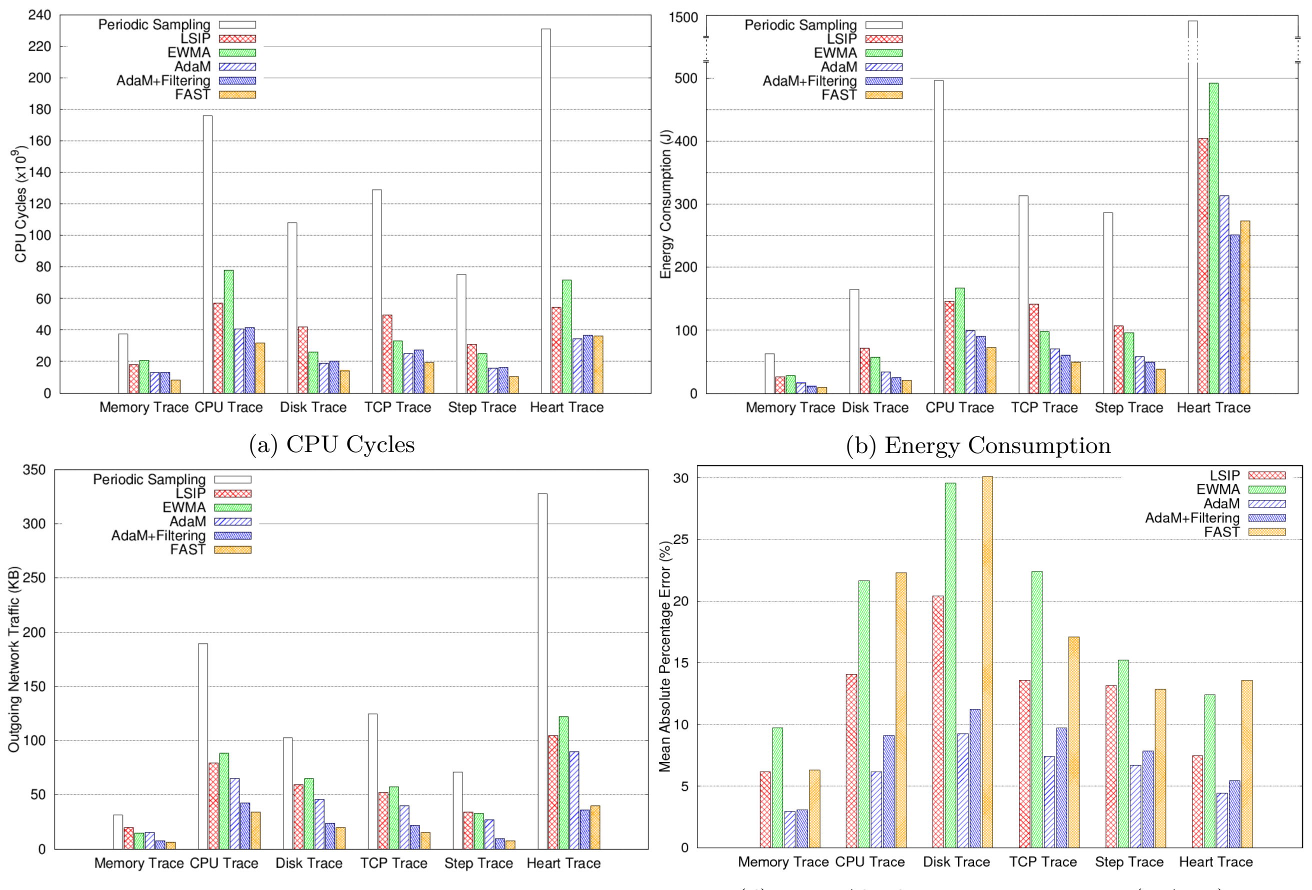
Moreover, as with initial MAPE comparison, AdaM outperforms i-EWMA and L-SIP. AdaM is able to achieve this due to its low complexity and the introduction of the confidence metric which supports the estimation process to select the appropriate T. Nonetheless, its overhead is slightly larger in some traces (e.g., cpu trace) than the 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 9d, 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 [32]. Figure 10 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 driving the device (eq. 18). 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 β is set to 0.7 which is industry standard practice to account for external factors affecting battery runtime (e.g., temperature).

$$I_{DC} = \frac{1}{n} \cdot \sum_{i}^{n} \frac{E_i \cdot \tau_i}{V_i} \tag{18}$$

$$BL = \beta \cdot \frac{B_C}{I_{DC}} \tag{19}$$

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.*



(a) CPU Cycles

(b) Energy Consumption

(c) Outgoing Network Traffic

(d) Mean Absolute Percentage Error (MAPE)

Fig. 9: Overhead Comparison of the Techniques Under Evaluation

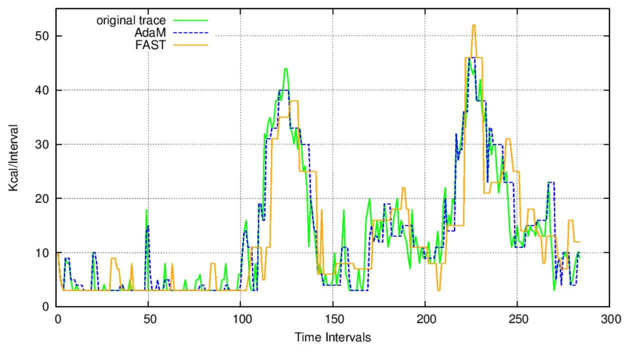


Fig. 10: Wearable Device Calories Comparison

5.2 Streaming Services Scalability Evaluation

In the next set of experiments, we show the benefits of integrating AdaM with two different streaming services in regards to reducing data velocity and increasing service scalability.

5.2.1 *VM Cluster Monitoring in a Private Cloud*

In this case, we study how a monitoring service used to monitor a cluster of VMs deployed over an Openstack private cloud can benefit by embedding AdaM in the monitored VMs to reduce both the volume and velocity of the monitoring data and ease processing on the monitoring service. To show this we take advantage of the open-source JCatascopia monitoring framework [25] which offers integration endpoints for adaptive algorithms and is capable of running on linux-based VMs, IoT devices and gateways. 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. Particularly, 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 an IoT cloud gateway. For each trace we set the minimum sampling period to $0.5s$, 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 11 depicts the topology of our testbed. Initially the testbed 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) residing in the same tenant network, 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 12 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.

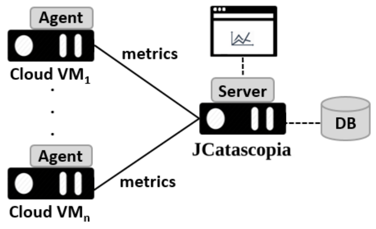


Fig. 11: Cloud Monitoring Topology

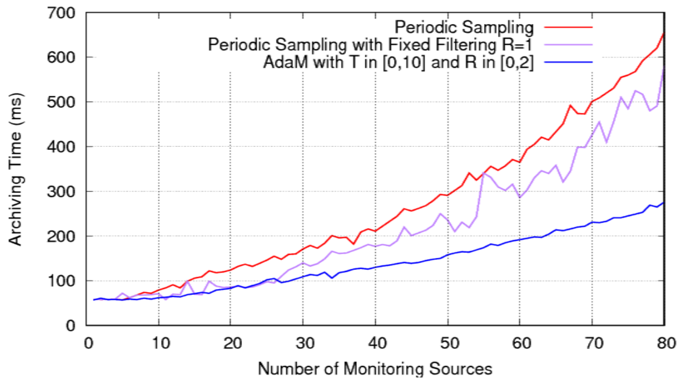


Fig. 12: Cloud Monitoring Scalability Evaluation

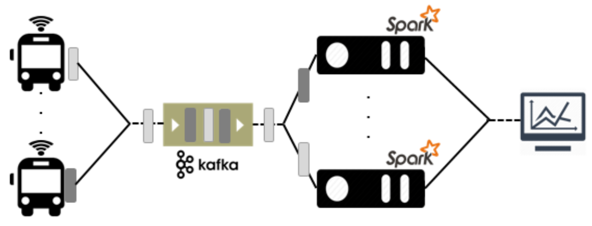


Fig. 13: ITS Topology

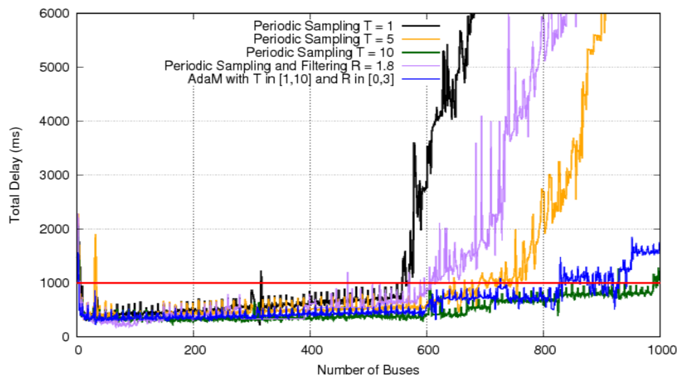


Fig. 14: Apache Spark Streaming Total Delay

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.*

5.2.2 Intelligent Transportation Service (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 13 and is a (simplified) replica of the Dublin Smart City Bus Network 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. 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[3] 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 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 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 5min 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 continuously increasing, batches are queued and not processed immediately as the system is unable to keep up. Therefore, the system is characterized as unstable.

Figure 14 depicts the total delay metric as the number of buses and message velocity increase. For T=1 (ground truth periodicity) the ITS becomes unstable after 572 buses with the total delay increasing exponentially and at maximum capacity (1000 buses) requiring for 23.8GB of data to be processed per Spark node. Similarly, for T=1 with filtering enabled and with the sampling period set to T=5, the ITS becomes unstable

3. https://github.com/dtrihinas/JobEmulator

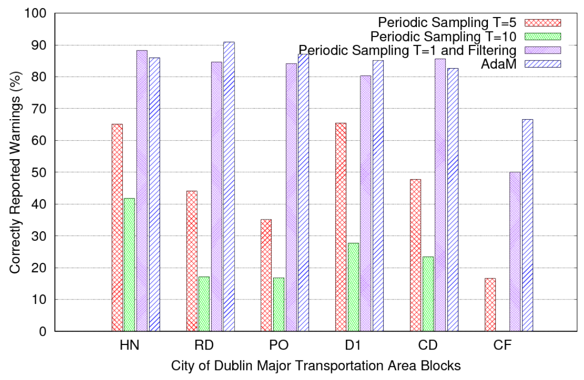


Fig. 15: Percentage of Correctly Reported Warnings Per Area

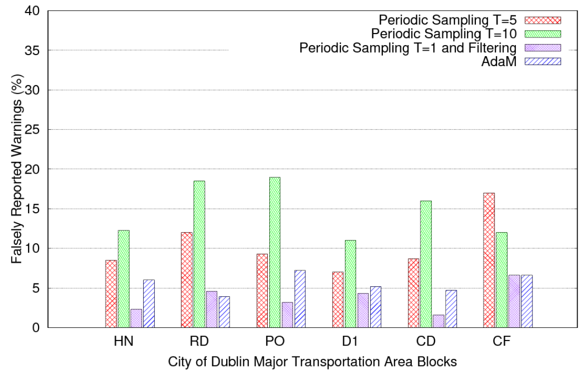


Fig. 16: Percentage of Falsely Reported Warnings Per Area

again after 632 and 746 buses respectively. 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 without the system becoming unstable at maximum capacity, while each Spark node only processes 8.7GB of data. *This results in a 64% reduction of the total volume of data processed per node when monitoring sources embed AdaM in their software core.* At this point one may argue that a solution to the high data influx is auto-scaling [25]. However, auto-scaling is a "blessing" which should be used wisely. 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 [33]. *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 14-16 we observe that although a T=10 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-

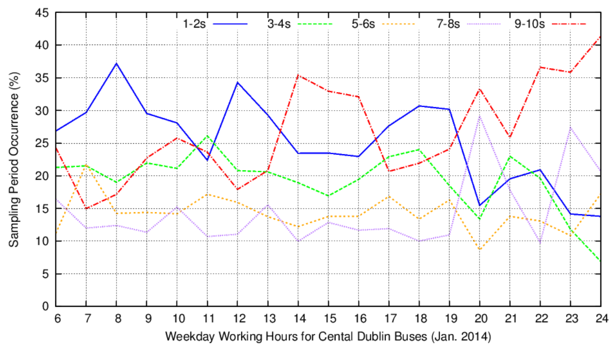


Fig. 17: Sampling Periods Used by Buses in the Central Dublin Area per Hour of the Day

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 17 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.*

6 Conclusions

In this paper we have presented AdaM, an open-source adaptive monitoring framework for IoT devices. Our main idea is to provide IoT devices with a framework that inexpensively and in place dynamically adapts the monitoring intensity and the amount of data disseminated through the network based on a runtime estimation model of the monitoring stream evolution and the variability. To achieve this, AdaM incorporates two algorithms, one for adaptive sampling and one for adaptive filtering. Both algorithms provide estimations, adjusting the sampling rate and the filter range based on the confidence of each algorithm to correctly estimate what will happen next in the monitoring stream. With real-world complex testbeds from cloud services, internet security services, wearables and intelligent transportation services, we show that in contrast to other techniques, AdaM achieves a balance between efficiency and accuracy. Specifically, AdaM reduces data volume by at least 74%, energy consumption by at least 71%, while preserving a greater than 89% accuracy. In turn, the velocity at which data are received by streaming services is significantly reduced, offering IoT services greater scalability with less resources and significantly lower costs.

Acknowledgement. This work is partially supported by the EU Commission in terms of Unicorn 731846 H2020 project (H2020-ICT-2016-1).

References

[1] V. Cerf and M. Senges, “Taking the Internet to the Next Physical Level,” *Computer*, vol. 49, no. 2, Feb 2016.

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

[3] 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.

[4] 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.

[5] 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.

[6] W. Shi and S. Dustdar, “The Promise of Edge Computing,” *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[7] 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.

[8] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski, “A System-Level Model for Runtime Power Estimation on Mobile Devices,” in *IEEE/ACM Int’l Conference on Cyber, Physical and Social Computing (CPSCom)*, Dec 2010.

[9] 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.

[10] 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, 2015.

[11] D. Trihinas, G. Pallis, and M. D. Dikaiakos, “ADMin: Adaptive monitoring dissemination for the Internet of Things,” in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[12] ——, “AdaM: an Adaptive Monitoring Framework for Sampling and Filtering on IoT Devices,” in *IEEE International Conference on Big Data*, 2015, pp. 717–726.

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

[14] Ganglia Monitoring, http://ganglia.info/.

[15] 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.

[16] 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. ACM, 2010.

[17] 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.

[18] 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, 2015.

[19] 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.

[20] 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.

[21] M. Du and F. Li, “ATOM: Automated tracking, orchestration and monitoring of resource usage in infrastructure as a service systems,” in *2015 IEEE International Conference on Big Data*, Oct 2015, pp. 271–278.

[22] 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, 2004, pp. 527–538.

[23] 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.

[24] 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 ’03. ACM, 2003, pp. 563–574.

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

[26] 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.

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

[28] 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.

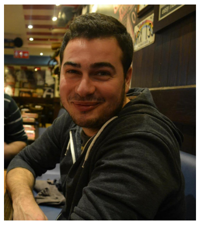
[29] SANS Technology Institute, https://isc.sans.edu/port.html.

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

[31] 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.

[32] 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.

[33] G. Copil, D. Trihinas, H.-L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. Dikaiakos, “ADVISE – A Framework for Evaluating Cloud Service Elasticity Behavior,” *12th Int’l Conference on Service-Oriented Computing (ICSOC 2014)*, 2014.



Demetris Trihinas is a Senior Researcher at the Computer Science Department, University of Cyprus. He holds a PhD in Computer Science from the University of Cyprus and a Dipl.-Ing. in Electrical and Computer Engineering from the National Technical University of Athens. His research interests include Distributed and Internet Computing with focus on Cloud Monitoring and Elasticity, and the Internet of Things. His work is published in IEEE/ACM conferences such as INFOCOM, CCGrid, BigData, TCC and ICSOC.

George Pallis is assistant professor at the Computer Science Department, University of Cyprus. His research interests include Cloud Computing with focus on Cloud Elasticity and Monitoring, Content Delivery Networks and Online Social Networks. His publication record is now at more than 60 research papers which have appeared in journals, book chapters, and conferences. He is Associate Editor in Chief in the IEEE Internet Computing magazine and he is also editing the "View from Cloud" department in this magazine.

Marios D. Dikaiakos is a professor of Computer Science at University of Cyprus and Director of the University’s Centre for Entrepreneurship. He received his Ph.D. in Computer Science from Princeton University in 1994. His research focuses on large-scale distributed systems with emphasis on cloud computing, online social networks and the world-wide web. He directed many funded research projects, published over 160 original peer-reviewed scientific papers, and leads the development of (open-source) software tools.