ADMin: Adaptive Monitoring Dissemination for the Internet of Things

Demetris Trihinas, George Pallis, Marios D. Dikaiakos

Department of Computer Science
University of Cyprus
Email: { trihinas, gpallis, mdd }@cs.ucy.ac.cy

Abstract—**As more knowledge is vastly added to the devices fuelling the Internet of Things (IoT) energy efficiency and real-time data processing are great challenges that must be tackled. In this paper, we introduce ADMin, a low-cost IoT framework that reduces on device energy consumption and the volume of data disseminated across the network. This is achieved by efficiently adapting the rate at which IoT devices disseminate monitoring streams based on run-time knowledge of the stream evolution, variability and seasonal behavior. Rather than transmitting the entire stream, ADMin favors sending updates for its estimation model from which values can be inferred, triggering dissemination only when shifts in the stream evolution are detected. Results on real-life testbeds, show that ADMin is able to reduce energy consumption by at least 83%, data volume by 71%, shift detection delays by 61% while maintaining accuracy above 91% in comparison to other IoT frameworks.**

I. Introduction

As consumers embrace the prevalence 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) [27]. Network-enabled devices in the form of wearables, home appliances, vehicles and drones differ from traditional sensing devices. In particular, IoT devices feature logic to produce analytic insights from raw monitoring data by incorporating smart algorithms [25]. However, to produce such an unprecedented wealth of insights intense processing is often required and for a battery-powered device processing along with constant data dissemination means less battery life [15]. While IoT hardware capabilities are projected to increase, battery capacity and bandwidth are not growing in the same rate [2].

In spite attempts of augmenting IoT devices with the power of the cloud there still exist numerous inhibitors masked under constant data dissemination such as bandwidth limitations and network latencies [6] [22]. According to Cisco's projections by 2019 IoT devices will be producing 500 ZB of monitoring data [5] with IDC reporting that by 2020 IoT monitoring data will account for 12% of the digital universal [12]. Therefore, it is no wonder why taming data velocity and energy efficiency are considered as great challenges to overcome in IoT [20] [22].

The remedy to reduce data velocity and consequently energy and bandwidth consumed for constant data dissemination is to suppress IoT data with approximation techniques [8] [28]. Ideally, an approximation technique will decide based on an estimation model following the metric stream evolution when values do not differ and can be suppressed within certain accuracy guarantees. However, in practice, current IoT approximation techniques are not suitable for abrupt and volatile metric streams [18] [21]. In turn, current cloud monitoring tools are not designed for this task, assuming the cloud as the center of all connected resources (e.g., VMs) when in reality IoT devices are scattered across the Internet backbone [26] [29]. Hence, the best they can do is to apply aggregation schemes to reduce bandwidth consumption. However, these schemes ignore significant knowledge that can be extracted from the metric stream evolution and in particular trends and seasonality behavior which are highly evident in IoT data (i.e., human body indicators, environmental data) [1].

To address these challenges we introduce ADMin. ADMin is an open-source framework that efficiently adapts, in place, the rate at which IoT devices disseminate monitoring streams to receiving entities based on the evolution and variability of the metric stream. To achieve this, ADMin incorporates low-cost adaptive and probabilistic learning algorithms which approximate the metric stream evolution. Thus, when metric stream values can be inferred from ADMin's estimation model, instead of sending these values, ADMin favors sending updates to its estimation model instead. From the estimation model receiving entities can then infer any missing values. In turn, dissemination to metric receivers is enabled only when shifts are detected and exceed the confidence intervals given by the user. This significantly reduces on device energy consumption allowing the IoT device's network unit to remain for longer periods in idle state and reduces the volume of data disseminated to receivers, easing processing in large-scale streaming networks. In order to reduce shift detection delays and the rate of false alarms (values that initially appear as shifts but are not), ADMin takes into account seasonality knowledge. In case seasonality is not helpful in the estimation process, ADMin detects this using online statistical testing, thus ignoring its contribution.

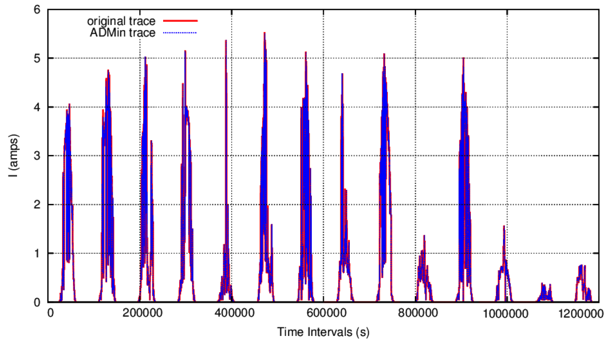


Fig. 1: PV Panel Current Production Trace

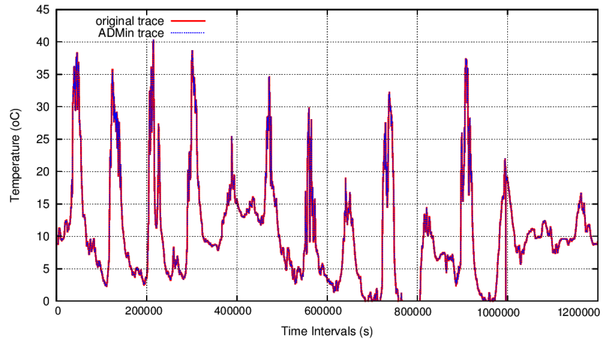


Fig. 2: Weather Station Temperature Trace

Preview of our results. A thorough evaluation was conducted by comparing the performance and accuracy of ADMin to other state-of-the-art IoT frameworks. All testbeds utilize real-life traces from different domains and in specific, from a photovoltaic production panel, a weather station and a wearable device. Figures 1-3 depict these traces in comparison to ADMin. 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. Moreover, our wearable dataset containing raw timestamped data (steps, heartrate, calories, active minutes) for a span of 6 months in 2016 is open-sourced. To the best of our knowledge, this makes it one of the largest activity tracking datasets publically available[1].

The rest of the paper is structured as follows: Section 2 presents the related work. Section 3 the problem statement. Section 4 introduces ADMin while Section 5 presents the evaluation. Section 6 concludes the paper.

II. Related Work

To date, there exists a number of shift detection and data reduction techniques for monitoring metrics. For example, PELT is an algorithm that can be used to detect optimal shifts in the evolution of a metric [16]. In

[1]https://github.com/dtrihinas/FitbitDataExtractor

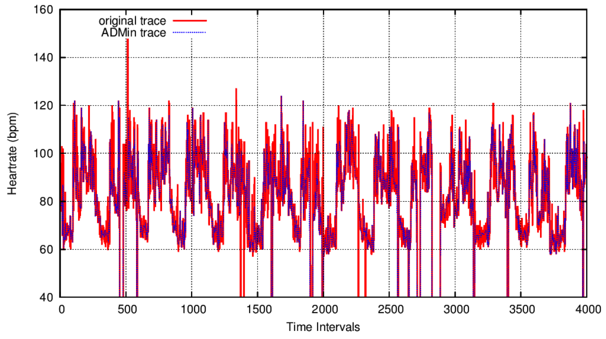


Fig. 3: Heartrate Trace from Wearable Device

turn, Singular Value Decomposition (SVD) and Piecewise Constant Approximation (PCA) techniques can produce compressed metric digests via curve fitting [14]. However, a common denominator for these techniques, is that they were not originally developed for metric streams running on monitoring sources scattered across IoT networks.

In contrast to the above, Silberstein et al. [23] introduce a framework that provides sensor networks with metric suppression for monitoring query responses that do not differ between sensors. The proposed mechanism reduces energy consumption but with the caveat that sensors must have knowledge of the network topology. In turn, Deligiannakis et al. [7] suggest buffering metric values at each sensing device and rather than transmitting all content, a base signal (wavelet) of fewer values is transmitted instead. The original signal is then reconstructed by the base signal within a certain accuracy. However, in order to provide such a base signal, a large portion of the signal must first be stored on the monitoring source.

LANCE [28] is a framework that reduces the bandwidth consumed for metric value dissemination by edge devices. Instead of sending all metric values, LANCE sends summaries of windowed values in the form of an average. The receiver then decides based on user-defined policies if the summarized values are of interest and must be downloaded or not. Another approach is G-SIP, proposed by Gaura et al [8]. G-SIP is an IoT framework that sends metric updates only when the metric stream changes in a way that cannot be predicted from previous value knowledge. As a prediction mechanism, G-SIP uses an exponential weighted average to follow the rate at which the metric stream changes in time. Hence, if the rate of change exceeds a threshold, metric updates are sent to the remote service, otherwise dissemination is suppressed. The downside of these frameworks is that they are slow to react to abrupt and volatile changes and static thresholds are used which are fixed upon initialization.

ADWIN [3] 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 [18], a tensor-based 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, RegimeCast is a server-side framework not intended to run on IoT devices.

Finally, techniques such as AdaM [25] and L-SIP [8], reduce on device energy consumption and data volume, but tackle the problem by dynamically adjusting the rate at which metrics are generated based on the monitoring stream evolution. We note that such techniques can complement our framework, thus further reducing energy consumption and the volume of IoT disseminated data.

III. Problem Statement

A. Preliminaries

We define a *metric stream* $M = \{d_i\}_{i=0}^{n}$ published by a monitoring source on an IoT device to a receiving entity, as a large stochastic sequence of independent and identically distributed (i.i.d) datapoints d_i, where $i = 0, 1, ..., n$ and $n \to \infty$. Each datapoint d_i is a tuple $(e_{id},\ t_i,\ v_i,\ ...)$ described, at the minimum, by a unique identifier e_{id}, a timestamp t_i and a value v_i. A datapoint may include a set of other attributes (e.g., location coordinates), although for brevity, when describing a datapoint we will omit these attributes without loss of generality. In turn, no assumptions are made for the type and number of generated datapoints which 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) [26].

B. Adaptive Monitoring Dissemination Overview

Monitoring stream dissemination is a fundamental process of IoT devices commonly implemented by periodically disseminating collected datapoints, such that for a fixed period of time T, the i-th datapoint is reported at $t_i = i \cdot T$. Due to its simplicity, this approach is widely adopted by monitoring systems although for battery-powered sensing devices, metric stream dissemination is the primary energy drain [21]. Thus, we argue that periodically disseminating metric streams features energy and resource constraints, especially when consecutive metric values do not vary.

To accommodate these challenges *adaptive dissemination* is used. Adaptive dissemination is the process of applying approximation techniques to sensed datapoints in order to reduce the communication overhead by suppressing from dissemination consecutive datapoints with “little” change in their metric values. How much “change” is considered as “little” depends on the given confidence $\delta \in [0, 1]$, denoting the probability with which estimated datapoints are approximated from sensed datapoints. Each monitoring source must maintain a runtime estimation model, denoted as $\rho(M)$, capturing knowledge of the monitoring 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 on the assumption that sensed datapoints can be approximated within the given confidence by a forecasting function of the estimation model, denoted as $f()$. Therefore, subsequent k-datapoints reported at $t_k = t_i + k \cdot T \mid k \subseteq \mathbb{Z}^+$, 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 shifts in the metric stream render the model no longer able to describe the metric stream evolution within the given confidence guarantees. At this point the monitoring source must disseminate to the receiver an updated version of the estimation model.

C. Requirements & Objectives

Obviously, absolute guarantees defeat the purpose of adaptive monitoring dissemination. In turn, if a degree of imprecision is tolerable but the estimation model cannot follow the metric stream evolution, then constant model updating will be required. Hence, metric dissemination will be replaced with model dissemination but the energy drain will not be reduced. Thus, the following requirements must be taken into consideration when designing an adaptive monitoring dissemination framework:

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 disseminating all collected datapoints and later discarding them.

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.

Thus, our main objective is: *to provide an estimation model capable of capturing runtime knowledge of the metric stream evolution to produce approximate datapoint values within given confidence guarantees and detect when these guarantees are violated to update the model in real-time. This will allow IoT devices to preserve energy by reducing the volume of disseminated IoT data while ensuring accuracy guarantees are maintained at all times.*

IV. The ADMin Framework

To address the above objective, we have designed the ADaptive Monitoring dissemINation framework. ADMin provides IoT devices with model-based monitoring dissemination, by adapting the rate at which metrics are disseminated to receiving entities based on the evolution, variability and seasonality of the metric stream. ADMin is developed in java as a lightweight framework embeddable in the source code of IoT devices (e.g., raspberry Pi, android devices). It can also be ported to other popular programming frameworks as it has no external source code dependencies. Figure 4 depicts ADMin embedded in the software core of an IoT device. ADMin coordinates metric value dissemination by interacting, as a proxy, between the *Sensing* and *Network Unit* of the IoT device.

In particular, when the Sensing Unit collects a new datapoint it is passed through ADMin's API to the *Adaptive Stream Estimation* module. This module updates a local reference estimation model capturing the metric stream evolution and trend. Trend estimation assists in reducing any lagging effects in the estimation process of the current stream evolution. At the same time, this module labels the current datapoint as "expected" or "unexpected", as documented in Section IV-A. If the datapoint is "expected", meaning it can be inferred by the model, it is suppressed; otherwise, the datapoint is locally stored and will be disseminated when dissemination is triggered.

After updating the evolution and trend, the *Seasonality Enrichment* module 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, as shown in Section V, if a metric stream exhibits such behavior, the estimation error and shift detection delay are significantly reduced. However, detecting the optimal *seasonal periodicity* is a complex problem by itself [1]. Thus, the user may enable ADMin to utilize the lightweight tensor-based and parameter-free ComCube framework to determine the near-optimal seasonal periodicity [19].

Next, the *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, the *Network Unit* 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.

A. *Adaptive Estimation Model*

We base our approach such that the estimation model is maintained in constant time and space (O(1) complexity), thus satisfying **R1**, which requires a low-cost estimation model able to run on 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**. ADMin supports model parameterization, although as input it only requires from the user to provide the certain confidence δ. The confidence will be obeyed by the estimation model, thus satisfying **R3**. As mentioned in Section III, no assumptions are made on the type or domain of the data, thus providing a generic framework and satisfying **R4**. Algorithm 1 provides an abstract overview of ADMin's adaptive

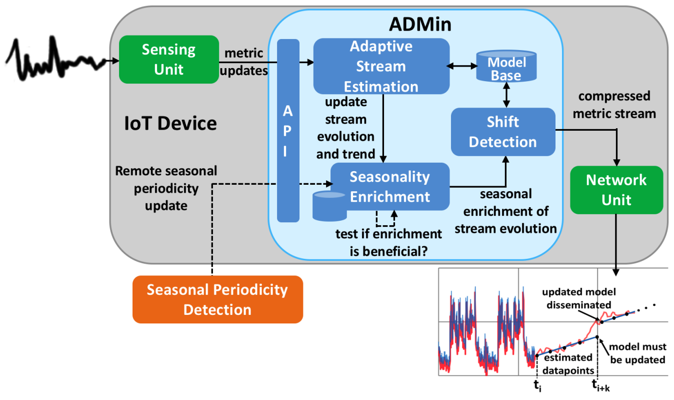

ADMin
Sensing Unit
metric updates
Adaptive Stream Estimation
Model Base
API
IoT Device
update stream evolution and trend
Shift Detection
compressed metric stream
Remote seasonal periodicity update
Seasonality Enrichment
seasonal enrichment of stream evolution
Network Unit
test if enrichment is beneficial?
Seasonal Periodicity Detection
updated model disseminated
model must be updated
estimated datapoints
t_i
t_{i+k}

Fig. 4: The ADMin Framework embedded to IoT Device

estimation model which captures runtime knowledge of the metric stream evolution, trend and seasonality.

Evolution estimation. When a datapoint is made available, ADMin will compute the current metric stream evolution $\rho(M)$, by using a moving average, denoted as μ_i. This will give an initial estimation for the next datapoint value, denoted as $\hat{v}_{i+1}$. Moving averages provide one-step ahead estimations. They are easy to compute, though many types exist, and can be calculated on the fly with only previous value knowledge. A cumulative moving average for streaming data is the Exponential Weighted Moving Average (EWMA), $\mu_i = \alpha\mu_{i-1} + (1-\alpha)v_i$, where a weighting parameter α, is introduced to decrease exponentially the effect of older values. However, the EWMA features a significant drawback; it is volatile to abrupt transient changes [25]. Thus, we adopt a Probabilistic EWMA (PEWMA) [4], which dynamically adjusts the weighting based on the probability density of the given observation. 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 1-2), it requires no parameterization, scaling to numerous datapoints.

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

Equation 1 presents the PEWMA where instead of a fixed weighting factor, we introduce a probabilistically adaptable weighting factor $\tilde{\alpha}_i = \alpha(1-\beta P_i)$. In this equation, the p-value, is the probability of the current v_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]. The logic behind probabilistic reasoning is that the current value v_i depending on it's p-value will contribute respectively to the estimation process. In turn, if a datapoint falls inside the prediction intervals determined from the given confidence, it is labeled as "expected" or "unexpected"

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

Algorithm 1 Adaptive 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 (see Section IV-A)
    and $k \leftarrow 0$ after dissemination

    compute p– and z–value
 1: $P_i, Z_i \leftarrow$ probDistro($v_i, \hat{v_i}, \sigma_i$)    (eq. 2)
    update estimation model
 2: $\mu_i, \sigma_i \leftarrow$ updPEWMA($P_i, v_i$)    (eq. 1)
 3: $X_i \leftarrow$ updTrend($\mu_i$)    (eq. 3)
 4: $S_i \leftarrow$ updSeasonality($\mu_i, X_i, L$)    (eq. 5)
    if datapoint is unexpected then store in buffer
 5: if isDatapointUnexpected($\delta, P_i, Z_i$) then
 6:     $buf \leftarrow v_i$
 7: end if
 8: $y_1 \leftarrow \mu_i + kX_i$    (eq. 4)
 9: $y_2 \leftarrow \mu_i + kX_i + S_i$    (eq. 6)
    is seasonality enrichment beneficial?
10: if T–Test($\delta, \sigma_i, y_1$) > T–Test($\delta, \sigma_i, y_2$) then
11:     $\hat{v}_{i+1} \leftarrow y_1$
12: else
13:     $\hat{v}_{i+1} \leftarrow y_2$
14: end if
15: $k \leftarrow k + 1$
16: return estModel($\hat{v}_{i+1}, \mu_i, \sigma_i, X_i, S_i$)
```

otherwise. 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 v_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. Moreover, in [25] the authors show that for the PEWMA, the α parameter can take a wide range of values if a small imprecision can be tolerated as most of the error is absorbed by the probabilistic weighting.

Assuming, as stated in Section III, 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 v_i evaluated under a Gaussian distribution, which is computed by Equation 2. Nonetheless, we note that while a Gaussian distribution is assumed, if prior knowledge of the distribution is available and given by the user then only step 1 must change in the estimation process.

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

To reduce the volume of disseminated data, ADMin will suppress "expected" datapoints that can be approximated by the estimation model. 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[3] (steps 5-6).

Trend estimation. First-order moving averages only perform one-step ahead estimations. Thus, solely using the mean to forecast subsequent k-estimations is error prone. Therefore, having updated the evolution of the metric stream we then compute the current trend (step 3), denoted as X_i and shown in Equation 3:

$$X_i = \begin{cases} v_i - v_{i-1}, & i = 2 \\ \gamma \ (\mu_i - \mu_{i-1}) + (1 - \gamma) \ X_{i-1}, & i > 2 \end{cases} \tag{3}$$

Equation 3 adopts Holt's Trend method [10] which captures an estimation of the metric stream growth/decay at the end of each i-th time interval. In this equation, γ is a smoothing parameter for the trend in the range $[0, 1]$ with γ usually set to 1 for a perfect linear effect or a value > 0.9 for a damped effect [24]. The ultimate goal of this method is to reduce lagging effects in the estimation process which are highly evident in time intervals with trends. Thus, any lagging effects are reduced by bringing the moving average to the appropriate value base. Therefore, after updating the trend, it can now be used with the PEWMA to approximate and forecast subsequent k-values as follows:

$$\hat{v}_{i+k|i} = \mu_i + k \ X_i \tag{4}$$

Seasonality estimation. Seasonality is defined as the tendency of the metric stream to exhibit behavior that repeats itself every L periods (e.g., hourly, daily) [1]. IoT data, such as human body indicators and environmental data, present seasonality behavior [11]. In this context, we compute the seasonality contribution (step 4), denoted as S_i, by adopting the Holt-Winter's Seasonality method (Equation 5) [9]. In this method, the seasonal contribution S_i, is computed based on the seasonal factor S_{i-L} of the last season (e.g., mean of the datapoints collected the previous hour), while ω is a smoothing parameter in the range $[0, 1]$ and usually set to a value < 0.5.

$$S_i = \begin{cases} 0, & i < L \\ \omega \ (v_i - \mu_i - X_i) + (1 - \omega) \ (v_i - S_{i-L}), & i > L \end{cases} \tag{5}$$

With the addition of seasonality knowledge Equation 4 can now be formed as:

$$\hat{v}_{i+k|i} = \mu_i + k \ X_i + S_i \tag{6}$$

However, in real-life systems perfect seasonal behavior is rarely observed. Rather, seasonality behavior is exhibited although irregularities in the form of noise are introduced

[3]Anomalies may be a sign of quality degradation for an IoT device

Algorithm 2 Adaptive Shift Detection

```
Input: $estModel$, confidence $\delta$, local storage $buf$ and
    $actTime$ denoting if actual shift time is returned
Output: $msg$ for dissemination
Ensure: length($buf$) < max($buf$)
 1: if dissemination triggered then
 2:     $h_i \leftarrow$ updShiftThres($\delta, \sigma_i$)                      (eq. 10)
 3: end if
 4: $c_i \leftarrow$ updLikelihood($v_i, \hat{v}_i, \mu_i, \sigma_i$)              (eq. 9)
 5: $C_{i,low},\ C_{i,high} \leftarrow$ updCusum($c_i$)                      (eq. 7)
 6: $G_{i,low},\ G_{i,high} \leftarrow$ updDecision($c_i$)                   (eq. 8)
 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. 8)
10:         return msg($t_s, buf, estModel$)
11:     else
12:         return msg($buf, estModel$)
13:     end if
14: end if
```

in the seasonal cycle (e.g. Fig. 2-3) [11]. Thus, considering prior knowledge, may result in overestimating subsequent v_i's, if the current evolution completely differs from the seasonal behavior [1]. To address such irregularities, two online pairwise T-tests (steps 10-14) are conducted to evaluate and select which contribution –before (y_1) or after (y_2)– seasonality enrichment, allows ADMin to deliver a more accurate estimation.

B. Adaptive Shift Detection

Algorithm 2 introduces an abstract overview of ADMin's adaptive shift detection approach, which is based on the lightweight and online Cumulative Sum test (CUSUM). The CUSUM, denoted as C_i, is a hypothesis test for detecting shifts in i.i.d timeseries [17]. In particular, there are two hypothesis θ' and θ'' with probabilities $P(M, \theta')$ and $P(M, \theta'')$, where the 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:

$$c_i = \ln \frac{P(M_i, \theta'')}{P(M_i, \theta')} \qquad (7)$$
$$C_{i,\{low,\ high\}} = C_{i-1,\{low,\ high\}} + c_i$$

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:

$$G_{i,,\{low,\ high\}} = \{G_{i-1,\{low,\ high\}} + c_i\}^+$$
$$t_s = \underset{j \le s \le i}{\arg\min}\,(C_{s-1}) \qquad (8)$$

In Equation 8, $z^+ = sup(z, 0)$, 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 an IoT metric stream constituted of i.i.d datapoints following a Gaussian kernel with the metric stream supposed to undergo possible shifts in its evolution modelled by a moving average. Thus, θ' and θ'' can be rewritten as μ' and μ'' respectively, with μ' representing the current evolution, while μ'' the output of the estimation model with $\mu'' = \mu' + \epsilon$, and ϵ 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 computed from Equation 2. With some calculations [13], c_i in Equation 7 is rewritten as follows to perform the decision-making process with only previous value knowledge:

$$c_{i,\{low,\ high\}} = \pm \frac{|\epsilon|}{\sigma_i^2}\ (v_i - \mu' \mp \frac{|\epsilon|}{2}) \qquad (9)$$

However, the CUSUM test features two significant drawbacks: (i) determining the actual t_s requires linear time; and (ii) the threshold h is never updated to reflect the runtime evolution shift of the metric stream. In regards to the first 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, t_i may greatly differ from t_s [17]. 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, the decision function is able to reduce shift detection delays. In regards to the second downside, the rationale for choosing h is primarily based on both reducing the risk of falsely indicating a shift in the metric stream distribution while also preserving the ability of the CUSUM to promptly detect shifts that matter. Hence, we follow an adaptive approach where h is updated after a dissemination is triggered, based on the number of standard deviations respecting the given user-defined confidence [17]. In turn, an optional positive value (h_{min}) may be used to restrict the sensitivity of the CUSUM so as to not oscillate between low values when the metric stream is relatively stable.

$$h_i = \max\{h_{min},\ h(\delta_i, \sigma_i)\} \qquad (10)$$

V. Evaluation

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

We compare ADMin to three frameworks: G-SIP [8], LANCE [28] and ADWIN [3], described in Section II. Specifically, G-SIP 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. Similar to G-SIP is LANCE, 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. On the other hand, ADWIN 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 will compare all frameworks to ADMin under different configurations and conduct each experiment with a tight confidence parameter of $\delta = 0.9$. For the frameworks using a moving average, we set the smoothing parameter to $\alpha = 0.45$, which is the best configuration for both G-SIP and LANCE. We will also leave the trend and seasonality smoothing parameter for G-SIP and ADMin to a default value of $\gamma = 0.95$ and $\omega = 0.35$ respectively.

A. Traces, Testbeds and Evaluation Metrics

Table I presents an overview of the real-life traces used to evaluate the under comparison frameworks. In this table we also present the 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 [16]. In turn, Figures 1-3 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 (1st gen, 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 IoT "smart" devices (e.g., home monitors, activity trackers). 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 HR wearable device (single-core ARM 32MHz processor, 128MB Memory).

We evaluate each framework towards: (i) *shift detection accuracy*, meaning both the number of correctly detected shifts (true positives) and the number of false alarms (false positives); (ii) *shift detection delay*, which is the difference in time to when a framework detects a shift from the actual time of occurrence; (iii) *data volume reduction* and *accuracy* at the receiver-side; and (iv) *total energy consumption*, based on the model depicted in Equation 11, where P_{idle} denotes the power in idle state; P_{cpu} the processor power (including CPU, L1 cache and memory); τ_{cpu} the CPU time; $P_{i/o}$ the power for I/O; $\tau_{cpuwait}$ the I/O time; while P_{net} strictly denotes the power consumed for disseminating packets over the network.

$$E = P_{idle} \cdot \tau_{idle} + P_{cpu} \cdot \tau_{cpu} + P_{io} \cdot \tau_{cpuwait} + P_{net} \cdot \tau_{net} \quad (11)$$

B. Experiments

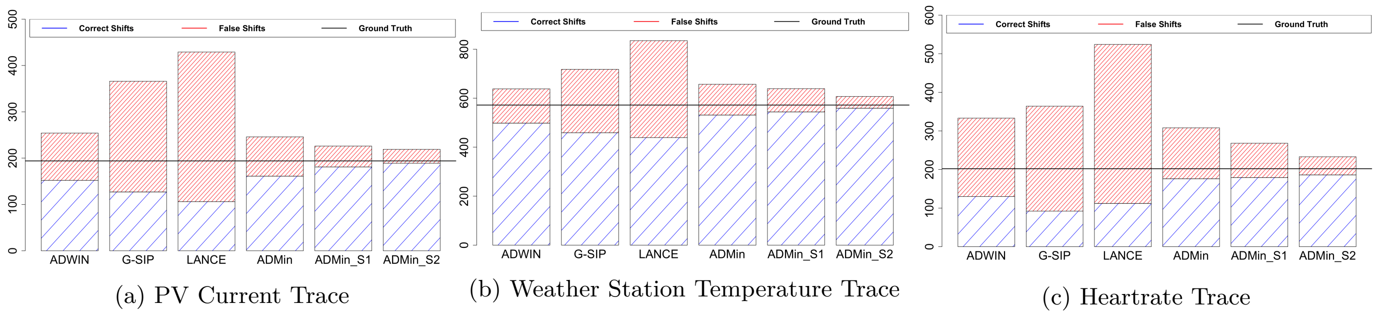
At first, let us denote the different configurations for our framework. ADMin denotes our framework without any seasonality enrichment while ADMin_S1 and ADMin_S2 feature seasonality configurations. Specifically, ADMin_S1 uses a static seasonal period configured once upon initialization and representing 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). On the other hand, ADMin_S2 uses the output of the ComCube framework [19], which provides a near-optimal approximation of the seasonal periodicity (L in eq. 6).

In our first evaluation we compare each framework ability to detect the actual shifts in the evolution of the metric stream. From Figures 5a-5c, 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 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 II 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%.* 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*

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 HR wearable device monitoring beats per minute (bpm) of the person wearing the device for a month (Jun 2016)

TABLE I: Traces Used for Performance and Accuracy Evaluation



(a) PV Current Trace (b) Weather Station Temperature Trace (c) Heartrate Trace

Fig. 5: Correct and False Metric Stream Shift Detection Comparison

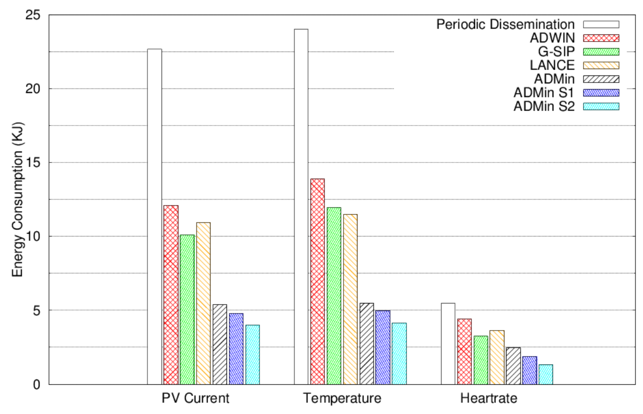


Fig. 6: On Device Energy Consumption Comparison

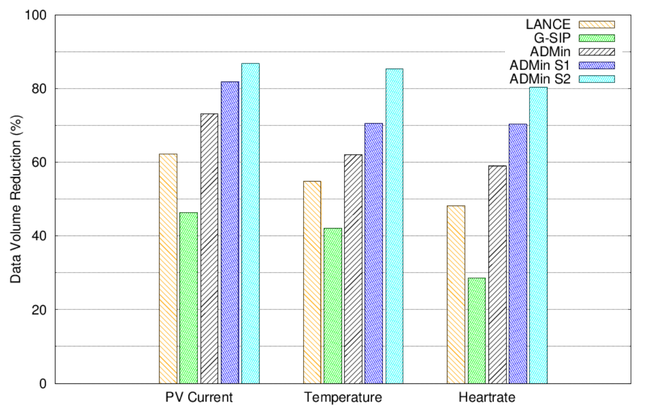


Fig. 7: Data Reduction Comparison

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 II: Metric Stream Evolution Shift Detection Delay

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), as well as, the volume of IoT data and estimation error at the receiver-side (Figures 7-8). 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 to the comparison where dissemination is withhold by applying a 10 time interval aggregation scheme.

From Figure 6 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 7 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 labelled 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

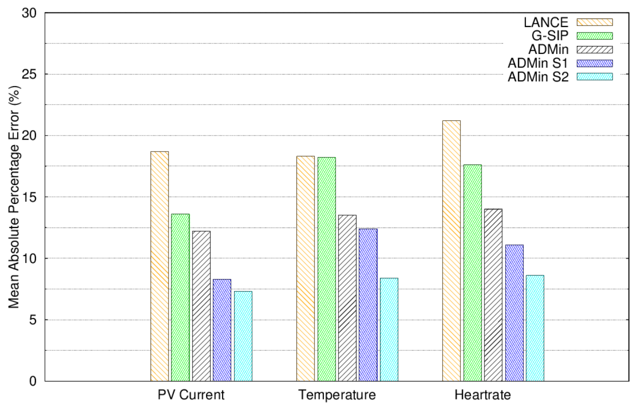


Fig. 8: Receiver-Side Mean Absolute Percentage Error

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 8 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.*

VI. Conclusion

In this paper we have presented ADMin, an adaptive monitoring dissemination framework for IoT devices. Our main idea is to provide IoT devices with a lightweight and model-based framework capable of adapting the rate at which monitoring streams are disseminated to receiving entities based on the evolution, variability and seasonality of the stream. The ADMin framework reduces IoT device energy consumption, as well as, allocated bandwidth and the volume of data disseminated through IoT networks. Exploiting the variability and seasonality of IoT monitoring streams, ADMin incorporates novel low-cost and probabilistic learning algorithms which efficiently model and estimate at runtime the monitoring stream evolution. Results show that ADMin is a viable solution that is lightweight, practical and achieves a balance between efficiency and accuracy for numerous diverse and real data.

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

References

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

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

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

[4] 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.

[5] Cisco, "Forecast and methodology 2014–2019," *Global Cloud Index*, 2013.

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

[7] 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*, 2004, pp. 527–538.

[8] 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.

[9] 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.

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

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

[12] IDC, "Rich Data and the Increasing Value of IoT," 2014.

[13] 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.

[14] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," *SIGMOD*, vol. 30, no. 2, pp. 151–162, 2001.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[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] J. W. Taylor, "Exponential smoothing with a damped multiplicative trend," *International Journal of Forecasting*, vol. 19, no. 4, pp. 715 – 725, 2003.

[25] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "AdaM: an Adaptive Monitoring Framework for Sampling and Filtering on IoT Devices," in *IEEE International Conference on Big Data*, 2015, pp. 717–726.

[26] 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.

[27] H.-L. Truong and S. Dustdar, "Principles for engineering iot cloud systems," *IEEE Cloud Computing*, vol. 2, no. 2, 2015.

[28] 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.

[29] 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 Hot Topics in Cloud Computing (HotCloud 15)*. USENIX, Jul. 2015.