Low-Cost Adaptive Monitoring Techniques for the Internet of Things

The AdaM Framework

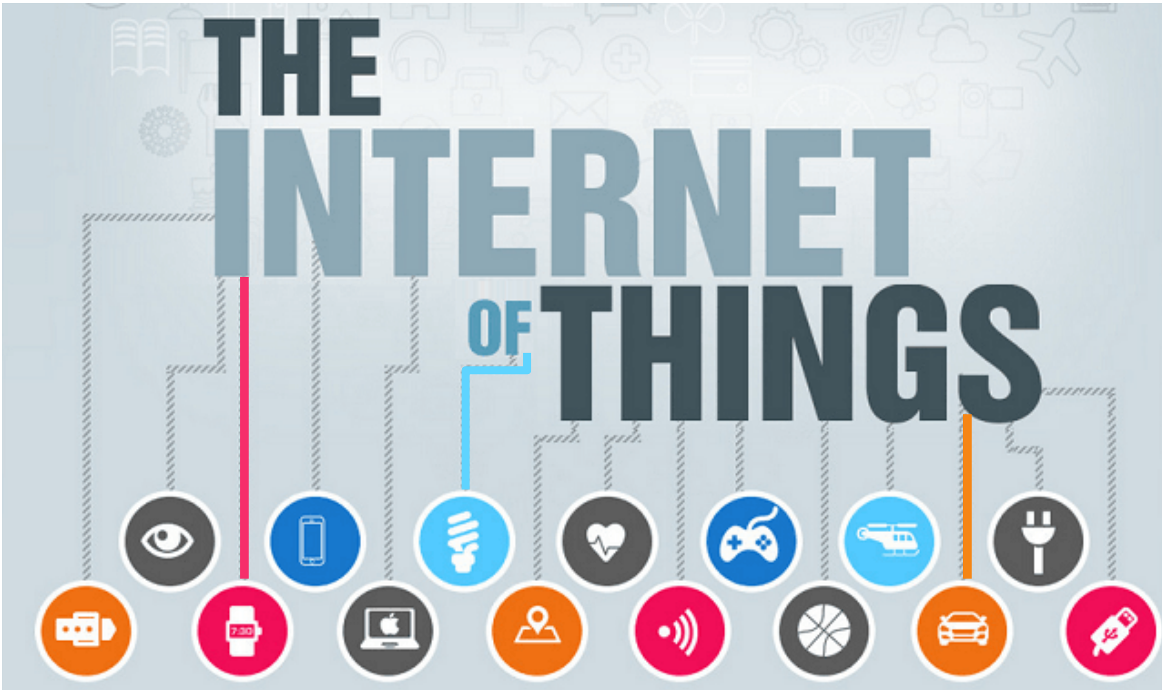
Demetris Trihinas
University of Cyprus
trihinas@cs.ucy.ac.cy

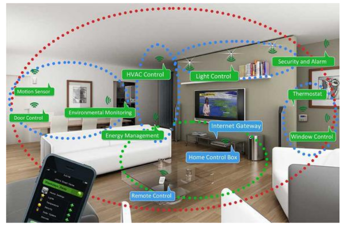
This talk is based on…

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

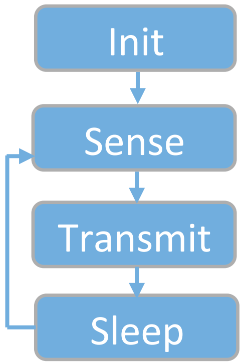
Low-Cost Adaptive Monitoring Techniques for the Internet of Things, **D. Trihinas** and G. Pallis and M. D. Dikaiakos, **Transactions on Big Data**, 2016, (In Review).






IoT **was** initially devices **sensing** and **exchanging** data streams with humans or other network-enabled devices



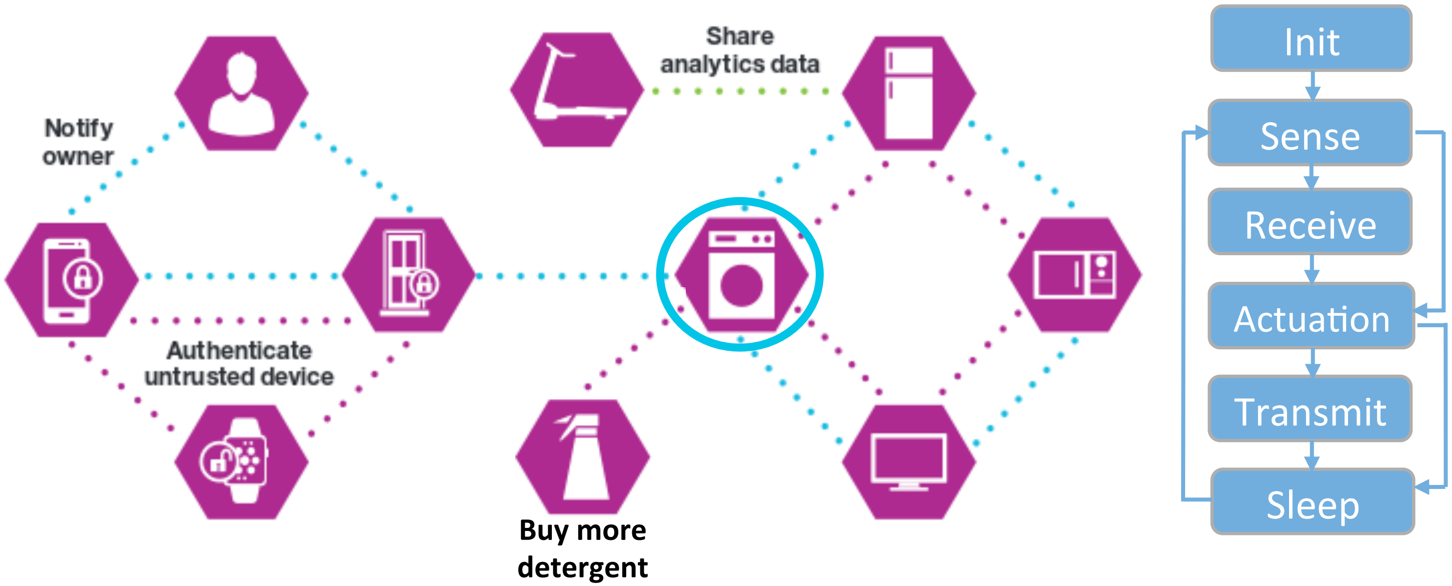
Cisco Blog > Internet of Everything

Growing up with Sensors and Smart Devices: How will the Internet of Everything Impact Our Children?


Sheila Jordan | October 2, 2013 at 4:34 pm PST

Edge-Mining

A term coined to reflect **data processing** and **decision-making** on **"smart" devices** that sit at the edge of IoT networks



…our devices just got a little bit more "smarter"…

The “Big Data” in IoT

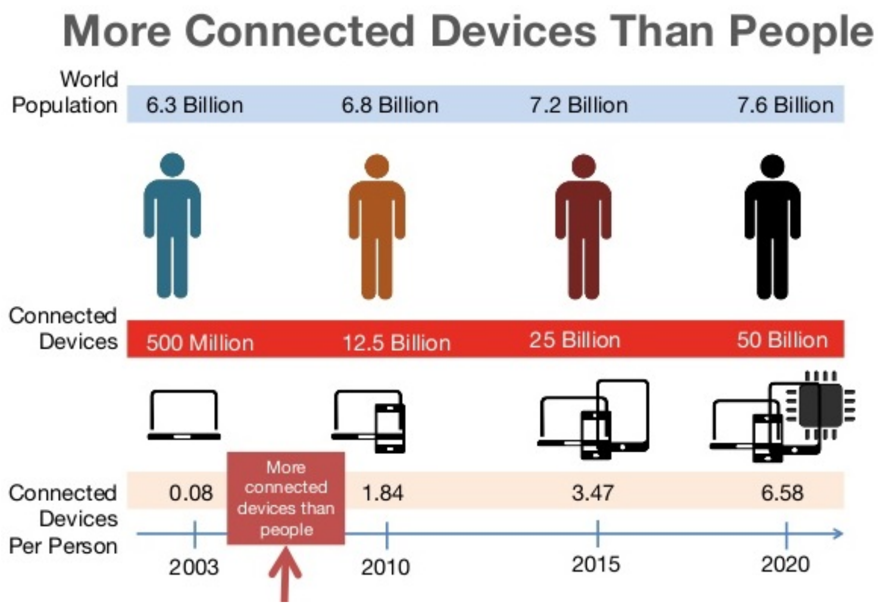
BY THE YEAR 2020, THERE WILL BE [IDC, Big Data in IoT, 2014]

50,000,000,000 connected devices, creating and sharing

40,000,000,000,000 GB

worth of data across the Internet of Things.

[Cisco, IBSG, Apr 2011]



IoT Monitoring Data [Gartner, 2015]

2% of digital universe in 2012

Projections for >10% in 2020

Big Data Challenges Transitioning to IoT

- Taming **data volume** and **data velocity** with **limited processing and network capabilities**

Device	CPU Speed	Memory	Price
Intel NUC	1.3 GHz	16 GB	~$300
Typical Phones	2 GHz	2 GB	~$300
Discarded Phones	1 GHz	512 MB	~$22
BeagleBone Black	1 GHz	512 MB	$55
Raspberry Pi	900 MHz	512 MB	$35
Arduino Uno	16 MHz	512 MB	~$22
mbed NXP LPC1768	96 MHz	32 KB	$10

Zhang et al., Usenix HotCloud, 2015

- IoT devices are usually battery-powered which means **intense processing** results in increased energy consumption (**less battery-life**)

Raspberry Pi 2 Bench Test

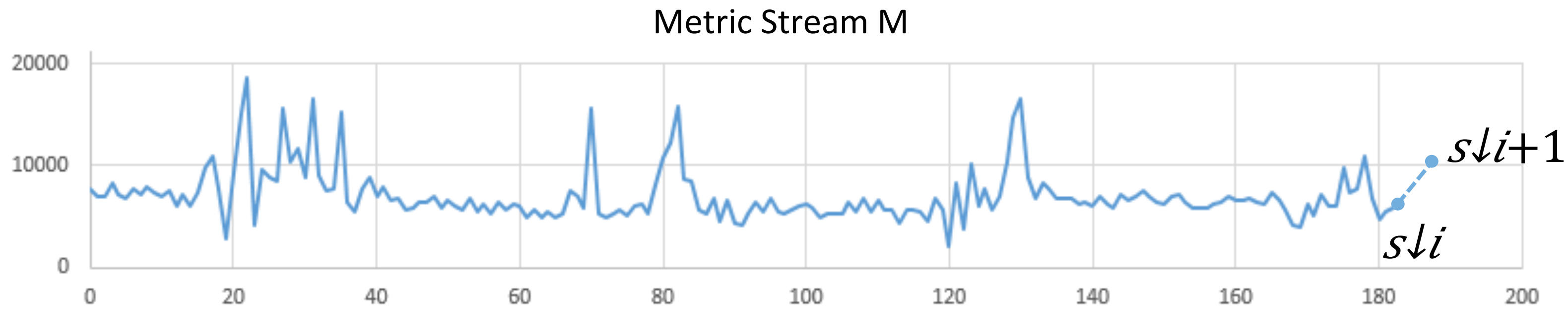
Pi State	Power Consumption
Idle	420 mA (2.1W)
400% CPU load	800-1100 mA (~4W)
400% CPU load + write to disk	900-1200 mA (~4.5W)
400% CPU load + write to disk + send over network	1250-1400 mA (~6.25W)

Low-Cost Adaptive Monitoring Techniques

Adaptive Sampling and Filtering

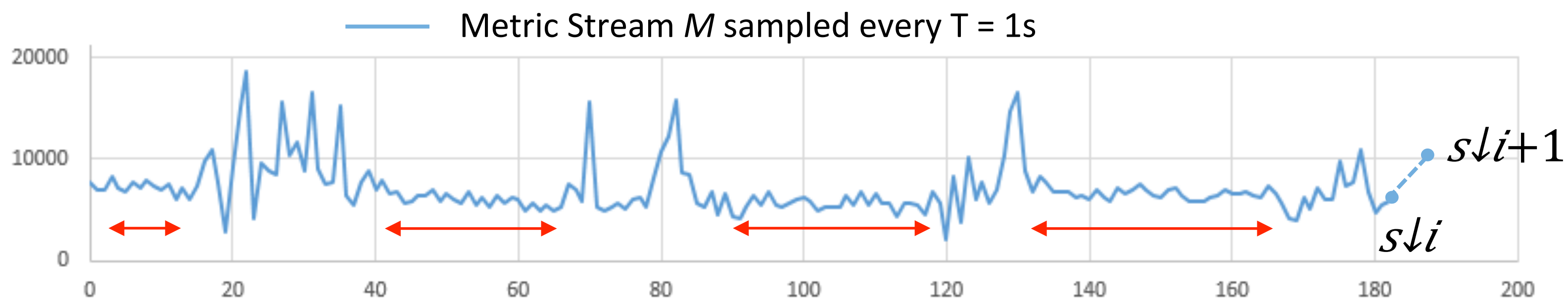
Metric Stream

- A **metric stream** $M=\{s_i\}_{i=0}^{n}$ published from a monitoring source is a large sequence of samples, denoted as s_i , where $i=0, 1,\ldots,n$ and $n\to \infty$
- Each **sample** s_i is a tuple (t_i, v_i) described by a timestamp t_i and a value v_i

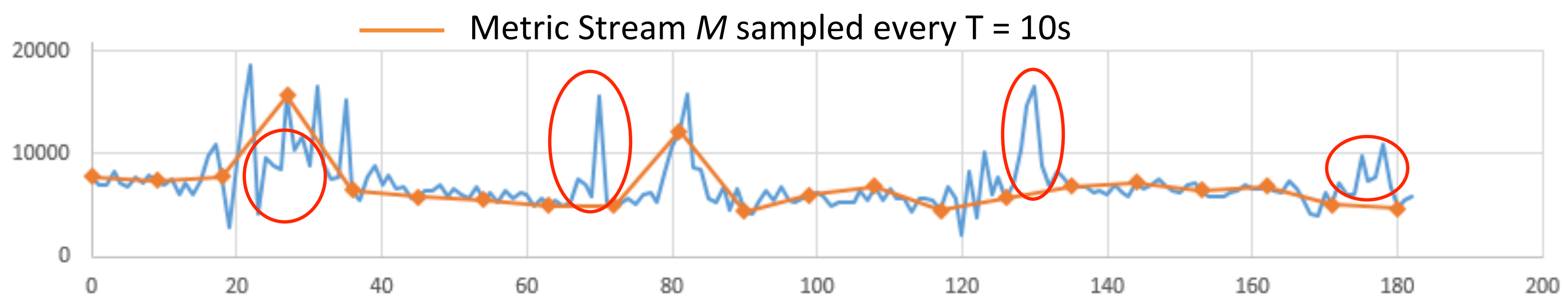


Periodic Sampling

- The process of triggering the collection mechanism of a monitored source every T time units such that the i^{th} sample is collected at time $t_i = i \cdot T$



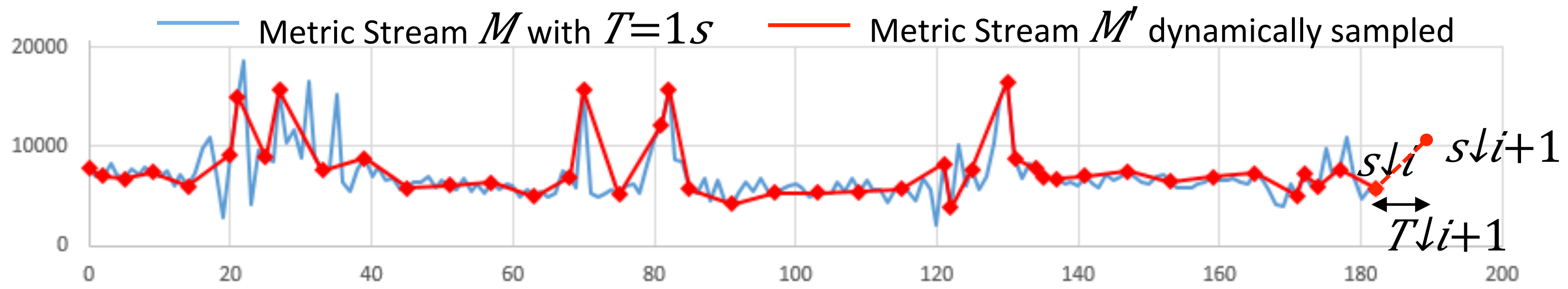
Compute resources and energy are wasted while generating large data volumes at a high velocity



Sudden events and significant insights are missed

Adaptive Sampling

- Dynamically adjust the sampling period T_i based on some function $\rho(M)$, containing information of the metric stream evolution



- Find the max $T \in [T_{min}, T_{max}]$ to collect sample s_{i+1} based on an estimation of the evolution of the metric stream $\rho(M)$, such that M' differs from M less than a user-defined imprecision value γ ($dist < \gamma$)

$$T^* = \arg\max_T \{f(s, T, \rho(M), dist, \gamma) \mid dist < \gamma,\ T \in [T_{min}, T_{max}]\}$$

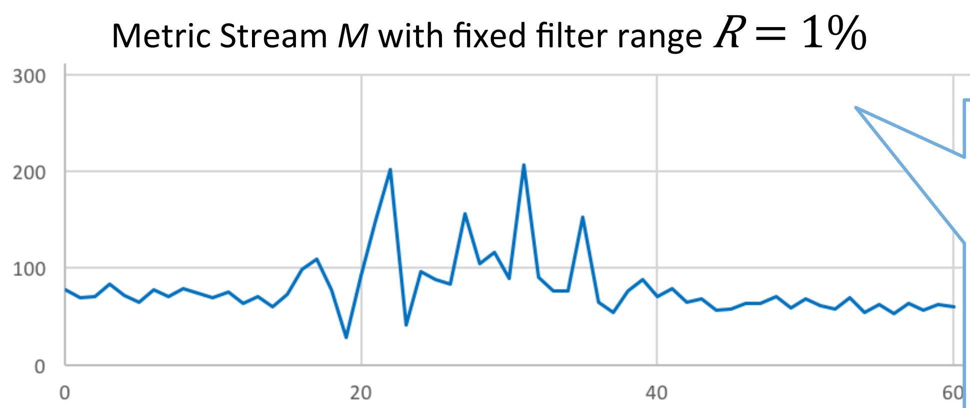
Metric Filtering

- The process of suppressing metric value dissemination when consecutive values do not "change" (e.g. differ less than a range of values)
- **Goal**: Reduce data volume and network overhead in favour of exact precision
- How much "change" is required, depends on the type of filter applied
- The receiver-side (e.g., base station, monitoring server) assumes that the values of any unreported metrics remain unchanged

Metric Filtering

- Fixed Range Metric Filters

```
if (curValue ∈ [prevValue – R, prevValue + R ])
    filter(curValue)
```



Can we know this R beforehand?

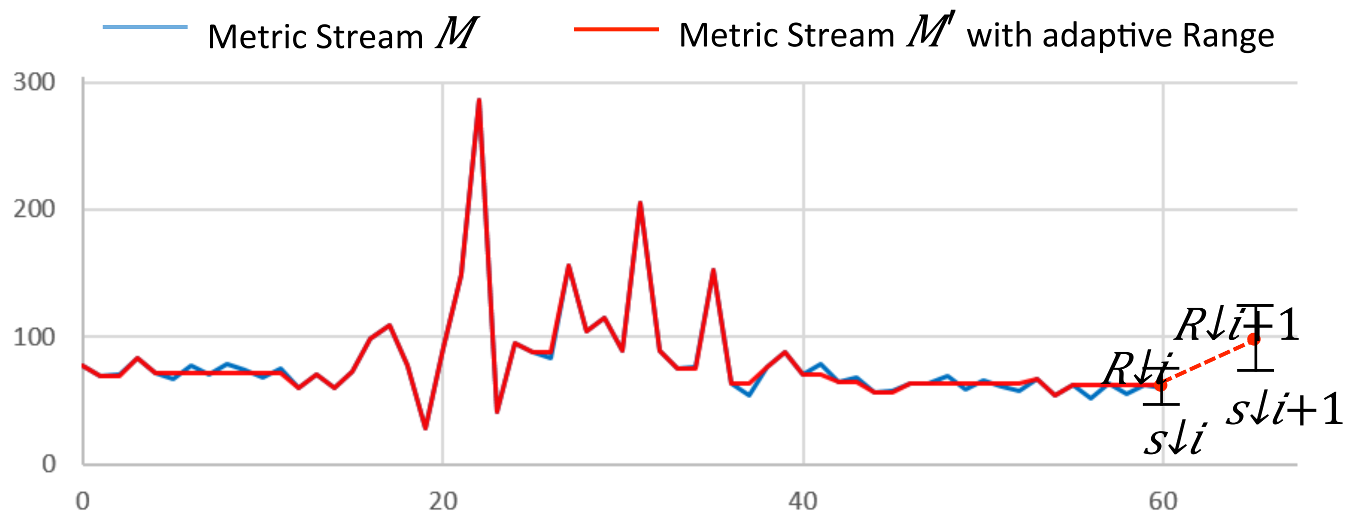
Should it keep the same value forever?

Is it the same for each and every metric?

No samples are filtered

Adaptive Filtering

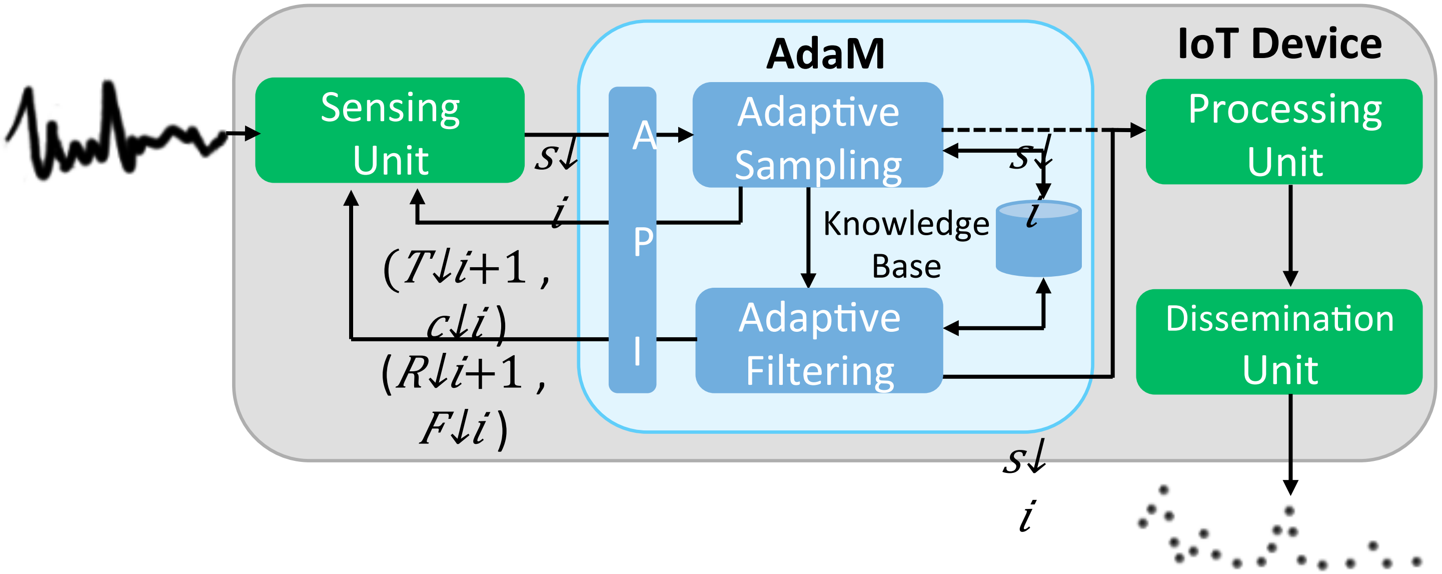
- Dynamically adjusting the filter range R based on the current variability of the metric stream, denoted as $q(M)$

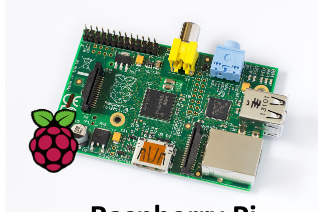


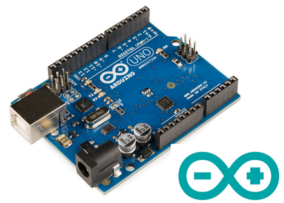
- Find the max filter range $R_{i+1} \in [R_{min}, R_{max}]$ for sample s_{i+1} such that M' differs from M less than a user-defined imprecision value γ based on the variability of the metric stream $q(M)$

The Adaptive Monitoring Framework

The AdaM Framework




Raspberry Pi


Arduino


Beacons

- Software library with no external dependencies embeddable on IoT devices
- **Reduces processing, network traffic and energy consumption** by adapting the monitoring intensity based on the metric stream evolution and variability
- **Achieves a balance between efficiency and accuracy**

It's open-source! Give it a try!
https://github.com/dtrihinas/AdaM

AdaM's Algorithms

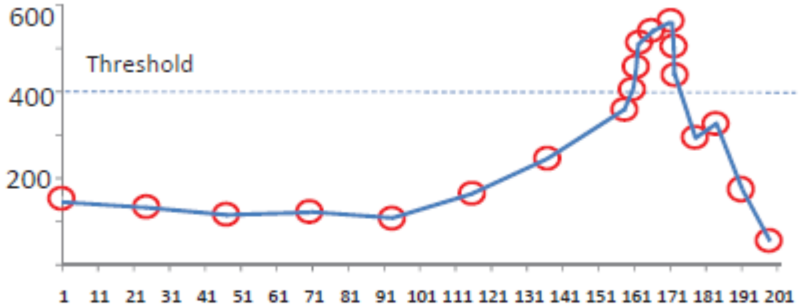
Adaptive Sampling & Filtering

Adaptive Sampling Algorithm

- **Step 1**: Compute the distance δ_i between the current two consecutive sample values

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

Why use the distance instead of the current value?



[Meng, Trans on Computers, 2013]

Threshold-base techniques increase sampling rate while sample values approach a user-defined threshold

- Good for anomaly detection (e.g. DDoS attacks)
- But, what about events away from threshold? (e.g. low rate DDoS attacks)

Adaptive Sampling Algorithm

- **Step 2**: Compute metric stream evolution based on a Probabilistic Exponential Weighted Moving Average (PEWMA) to estimate δ_{i+1} and the standard deviation σ_{i+1} :

$$\delta_{i+1} = \mu_i = a \cdot \mu_{i-1} + (1-a)\delta_i$$

Looks like an exponential moving average, right?

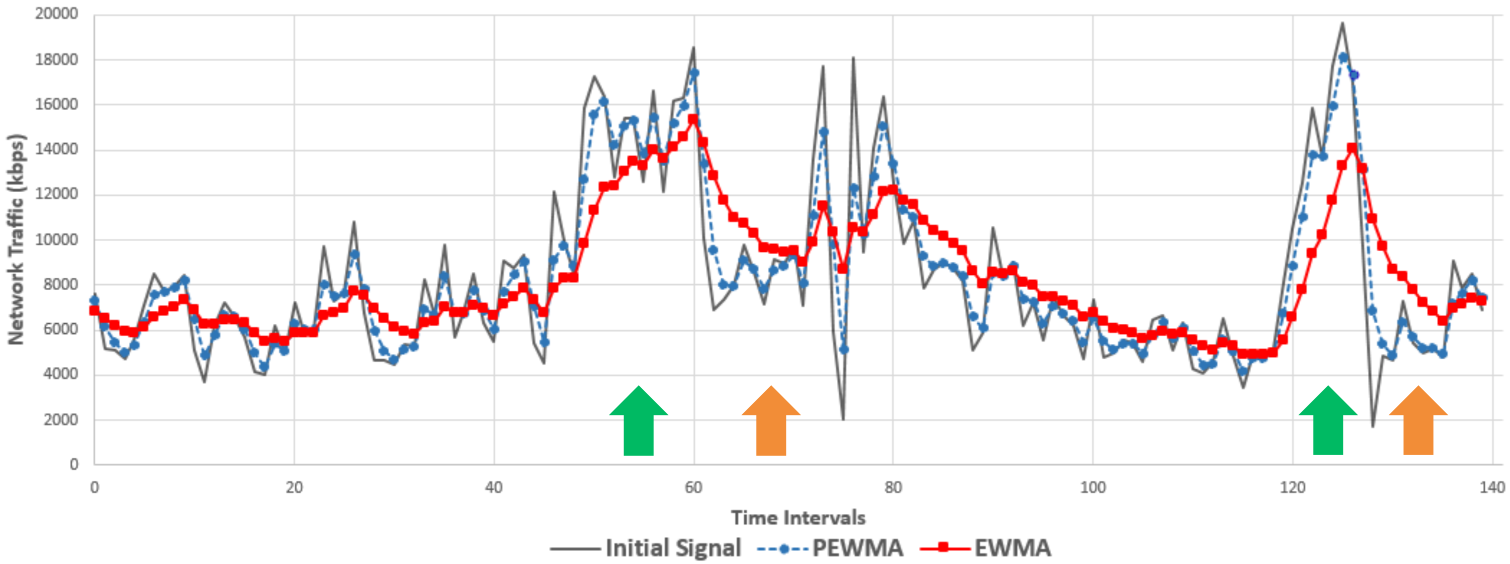
But weighting is **probabilistically** applied!

$$a = \alpha(1-\beta P_i)$$

$$s_1 = \mu_i \leftarrow \tilde{a}_i \cdot s_1 + (1-\tilde{a}_i) \cdot \delta_i$$
$$s_2 \leftarrow \tilde{a}_i \cdot s_2 + (1-\tilde{a}_i) \cdot \delta_i^2$$
$$\hat{\delta_{i+1}} \leftarrow s_1$$
$$\hat{\sigma_{i+1}} \leftarrow \sqrt{s_2 - s_1^2}$$

Moving standard deviation with only previous value knowledge

Why use a Probabilistic EWMA?



Simple EWMA is volatile to abrupt transient changes

- Slow to acknowledge spike after "stable" periods
- If "stable" phases follow sudden spikes, then subsequent values are overestimated

Sounds a lot like a job for a Gaussian distribution!

Adaptive Sampling Algorithm

- **Step 3**: Compute actual standard deviation σ_i
- **Step 4**: Compute the current **confidence** $(c_i \leq 1)$ of our approach based on the actual and estimated standard deviation

$$c_i = 1 - |\sigma_i - \sigma_i| / \sigma_i$$

… the more "confident" the algorithm is, the larger the outputted sampling period T_{i+1} can be…

- **Step 5**: Compute sampling period T_{i+1} based on the current confidence and the user-defined imprecision γ (e.g. 10% tolerance to errors)

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

λ is an aggressiveness multiplier (default λ=1)

O(1) complexity as all steps only use their previous values!

Adaptive Filtering Algorithm

- **Step 1**: Compute the current variability of the metric stream using a moving Fano Factor F_i based on exponentially weighted average μ_i and standard deviation σ_i already computed from adaptive sampling

$$F_i = \sigma_i^2 / \mu_i$$

…a low F_i (due to σ_i) indicates a currently in-dispersed data stream which means low variability in the metric stream…

Why use variability and not follow a stepwise approach?

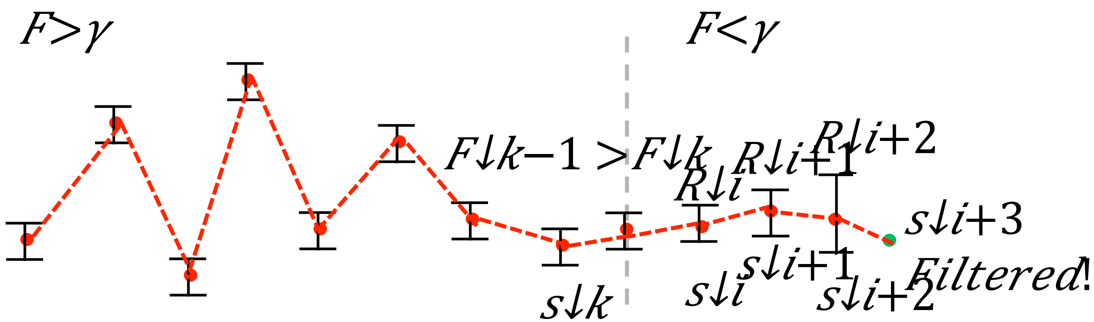
$$R_i = R_{i-1} \pm 0.01 \cdot R_{i-1}$$

Even for a 1% adjustment critical values can be filtered out in biosignal monitoring

Adaptive Filtering Algorithm

- **Step 2**: Compute the new filter range R_{i+1} based on F_i and the user-defined imprecision γ

$$R_{i+1} \leftarrow R_i + \lambda \cdot \left(\frac{\gamma - F_i}{\gamma}\right)$$



O(1) complexity as all steps only use their previous values!

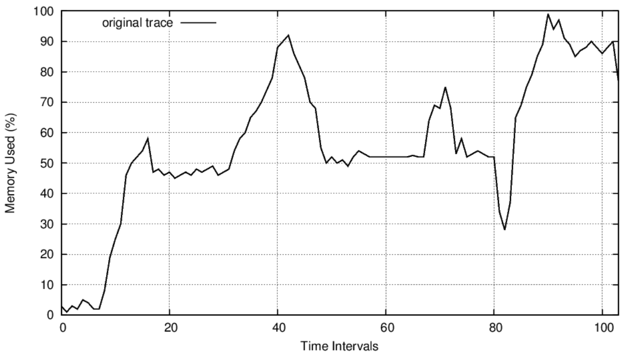
Evaluation

AdaM vs State-of-the-Art

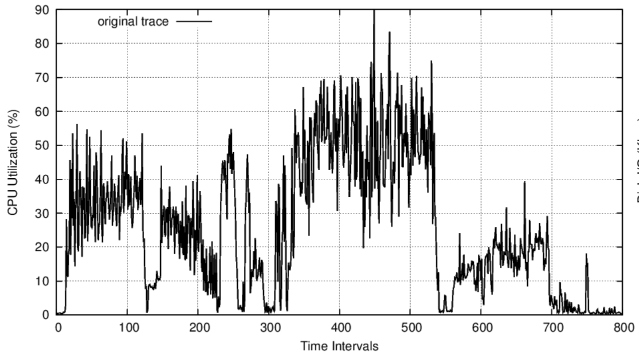
- **i-EWMA** [1]: an EWMA adapting sampling period by 1 time unit when the estimated error (ε) is under/over user-defined imprecision
- **L-SIP** [2]: a linear algorithm using a double EWMA to produce estimates of current data distribution based on rate values change
  - Slow to react to highly transient and abrupt fluctuations in the metric stream
- **FAST** [3]: an (aggressive) framework using a PID controller to compute (large) sampling periods accompanied by a Kalman Filter to predict values at non sampling points

[1] Trihinas et al., CCGrid, 2014 [2] Gaura et al., IEEE Sensors Journal, 2013 [3] Fan et al., ACM CIKM, 2012

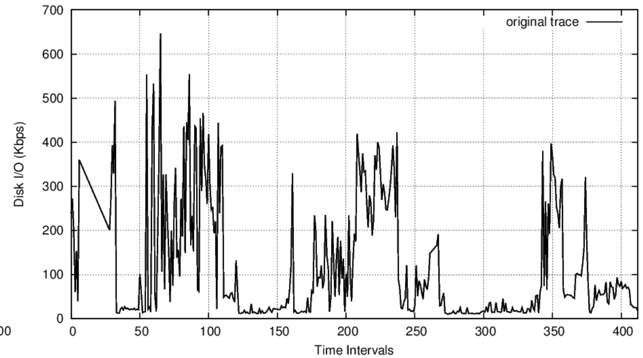
Datasets



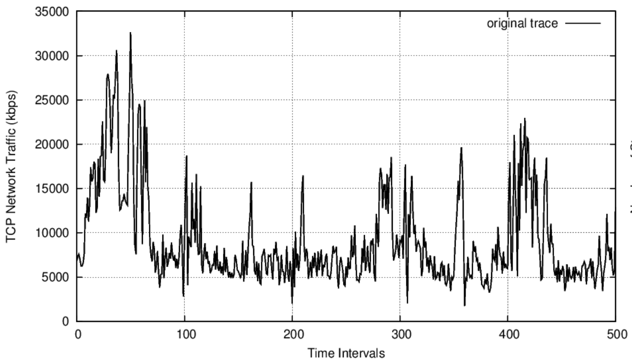
Memory Trace
Java Sorting Benchmark



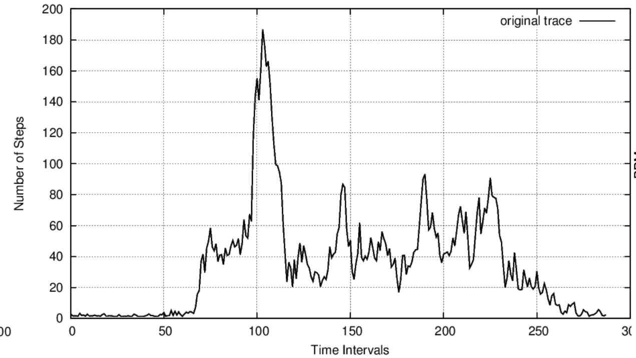
CPU Trace
Carnegie Mellon RainMon Project



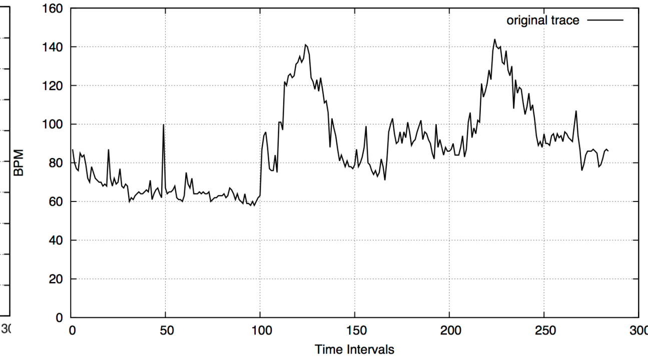
Disk I/O Trace
Carnegie Mellon RainMon Project



TCP Port Monitoring Trace
Cyber Defence SANS Tech Institute

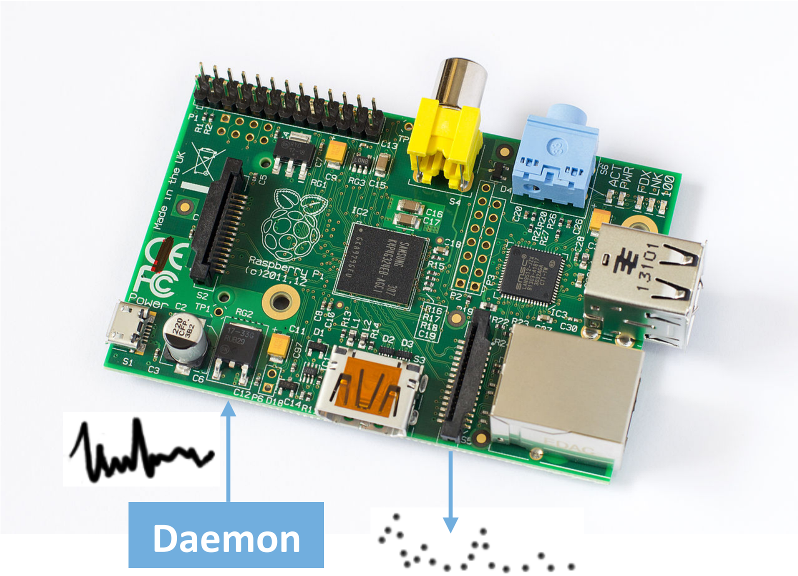


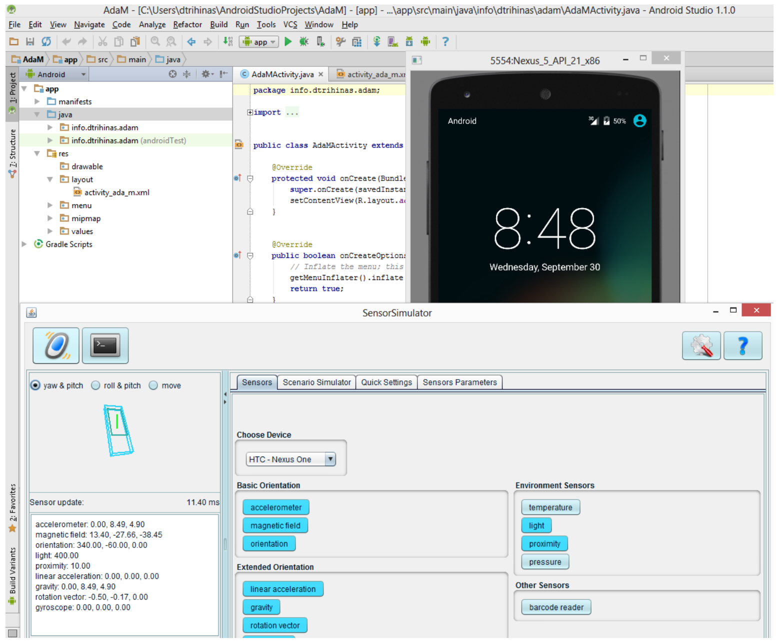
Step Trace
Fitbit Charge HR Wearable



Heart Trace
Fitbit Charge HR Wearable

Our Small "Big Data" Testbeds





Raspberry Pi 1st Gen. Model B
512MB RAM, Single Core ARM 700MHz

Daemon on OS emulates traces while feeding samples to each algorithm

Android Emulator + SensorSimulator
128MB RAM, Single Core ARM 32MHz

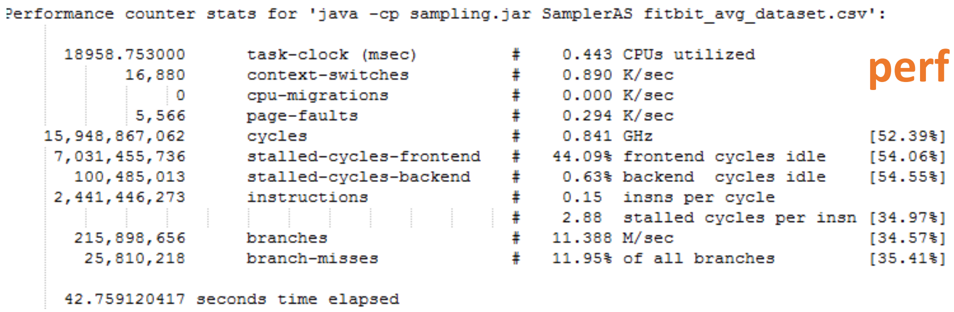
SensorSimulator script emulates traces by feeding samples to **Android Wear** emulator for Step and Heartrate processing

Evaluation Metrics

- Estimation Accuracy – Mean Absolute Percentage Error (MAPE)

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

- CPU Cycles



```
Performance counter stats for 'java -cp sampling.jar SamplerAS fitbit_avg_dataset.csv':

    18958.753000      task-clock (msec)         #    0.443 CPUs utilized
          16,880      context-switches          #    0.890 K/sec
               0      cpu-migrations            #    0.000 K/sec
           5,566      page-faults               #    0.294 K/sec
  15,948,867,062      cycles                    #    0.841 GHz                     [52.39%]
   7,031,455,736      stalled-cycles-frontend   #   44.09% frontend cycles idle    [54.06%]
     100,485,013      stalled-cycles-backend    #    0.63% backend  cycles idle    [54.55%]
   2,441,446,273      instructions              #    0.15  insns per cycle
                                                #    2.88  stalled cycles per insn [34.97%]
     215,898,656      branches                  #   11.388 M/sec                   [34.57%]
      25,810,218      branch-misses             #   11.95% of all branches         [35.41%]

    42.759120417 seconds time elapsed
```

perf

- Outgoing Network Traffic

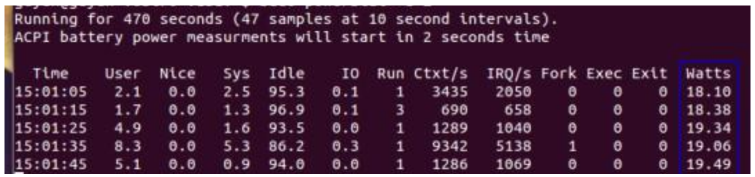
nethogs

```
  PID USER      PROGRAM                            DEV     SENT      RECEIVED
2692  dtrihinas/usr/bin/python                     eth0    0.351      1.102 KB/sec
2726  dtrihinas./insync                            eth0    0.290      1.073 KB/sec
8013  dtrihinas/usr/lib/firefox/firefox            eth0    0.627      0.642 KB/sec
7991  dtrihinas/opt/google/chrome/chrome           eth0    0.191      0.164 KB/sec
2702  dtrihinas..ox-lnx.x86-3.10.8/dropbox         eth0    0.105      0.089 KB/sec
2990  dtrihinas..b/thunderbird/thunderbird         eth0    0.000      0.000 KB/sec
?     root      unknown TCP                                0.000      0.000 KB/sec

  TOTAL                                                    1.564      3.071 KB/sec
```

- Edge Device Energy Consumption*

Powerstat and Wattch**



```
Running for 470 seconds (47 samples at 10 second intervals).
ACPI battery power measurments will start in 2 seconds time

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
15:01:05   2.1   0.0   2.5  95.3   0.1    1   3435   2050    0    0    0  18.10
15:01:15   1.7   0.0   1.3  96.9   0.1    3    690    658    0    0    0  18.38
15:01:25   4.9   0.0   1.6  93.5   0.0    1   1289   1040    0    0    0  19.34
15:01:35   8.3   0.0   5.3  86.2   0.3    1   9342   5138    1    0    0  19.06
15:01:45   5.1   0.0   0.9  94.0   0.0    1   1286   1069    0    0    0  19.49
```

$$E = P_{idle} \cdot \tau_{idle} + P_{cpu} \cdot \tau_{cpu} + P_{io} \cdot \tau_{cpuwait} + P_{net} \cdot \tau_{net}$$

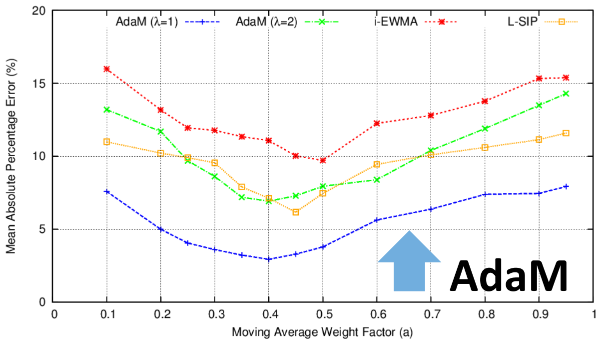
*Xiao et al., ACM DAC, 2010
**Brooks, ACM SIGARCH, 2000

Other than error evaluation no other study goes through an overhead study!

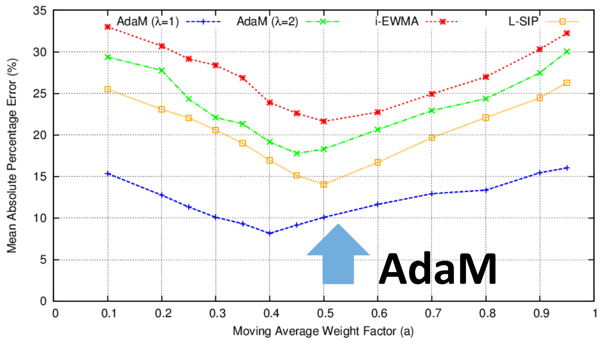
Error Comparison

AdaM (λ=1) AdaM (λ=2) i-EWMA L-SIP

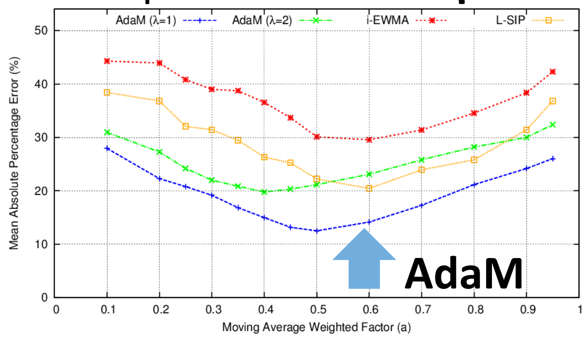
User-defined accepted imprecision set to **γ = 0.1**



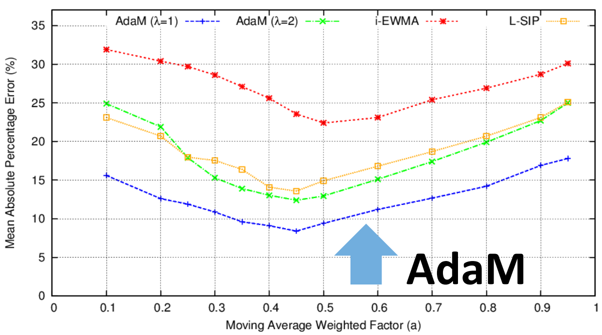
Memory Trace



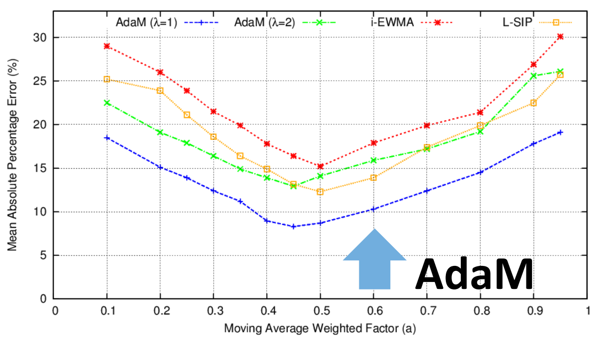
CPU Trace



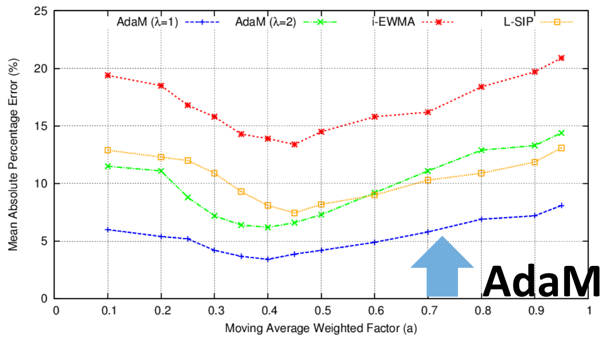
Disk I/O Trace



TCP Port Monitoring Trace



Fitbit Step Trace



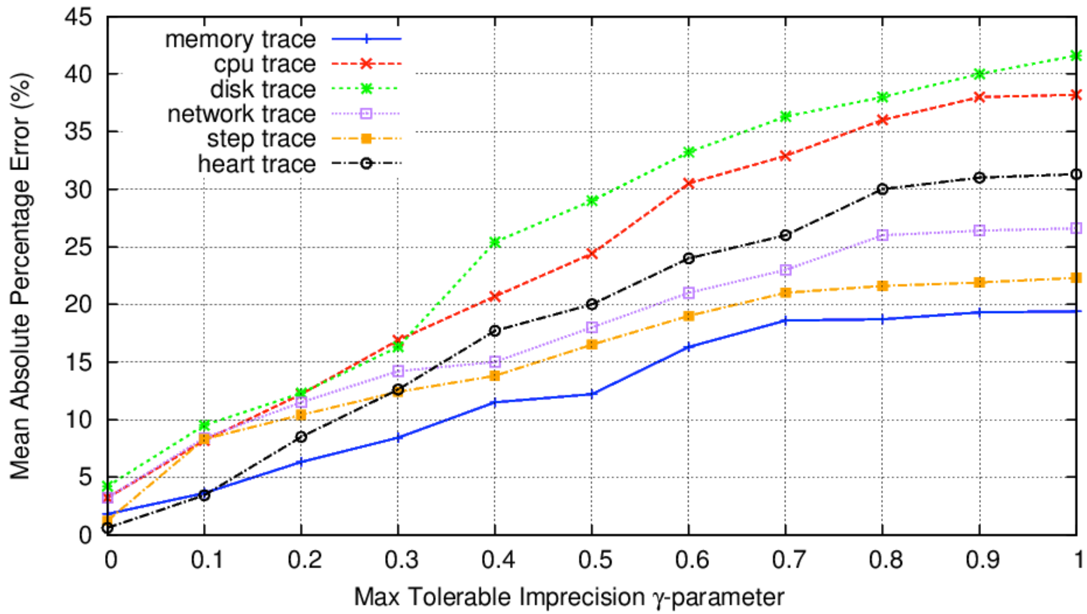
Fitbit Heartrate Trace

For all traces AdaM has the smallest inaccuracy (<10%)

Even in a more aggressive configuration (λ=2) AdaM is still comparable to L-SIP

AdaM γ-Parameter Evaluation

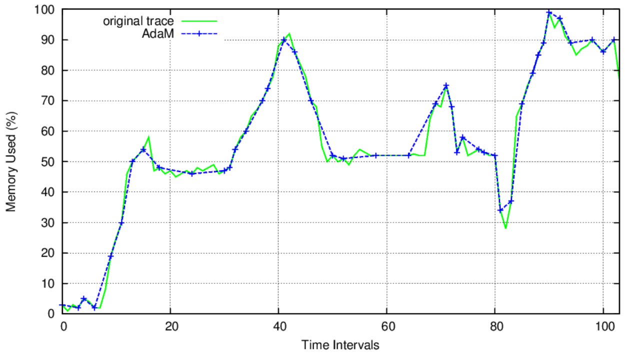
For a wide range of γ-parameter values we compute AdaM's MAPE per trace



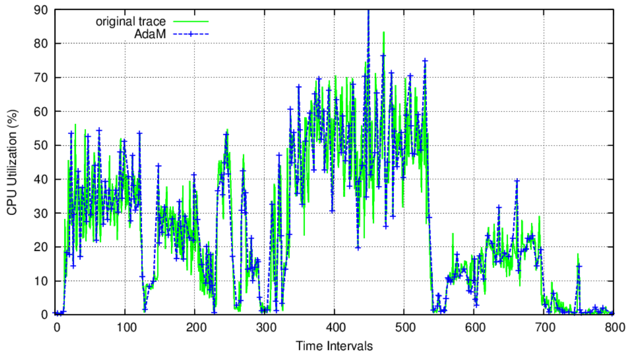
Even for extreme imprecision values (>0.3) AdaM can still take correct decisions signifying the importance of the confidence metric

So How Well Does AdaM Perform?

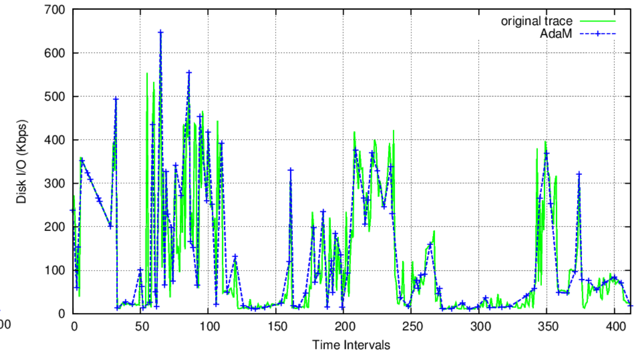
original trace ——— AdaM ---+---



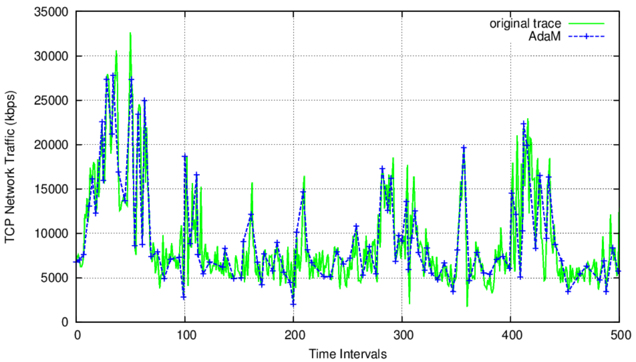
Memory Trace
Java Sorting Benchmark



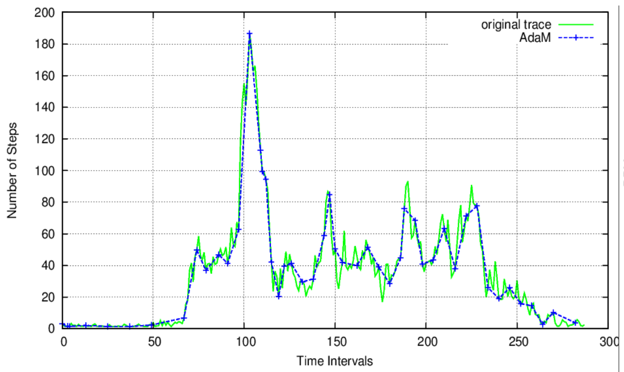
CPU Trace
Carnegie Mellon RainMon Project



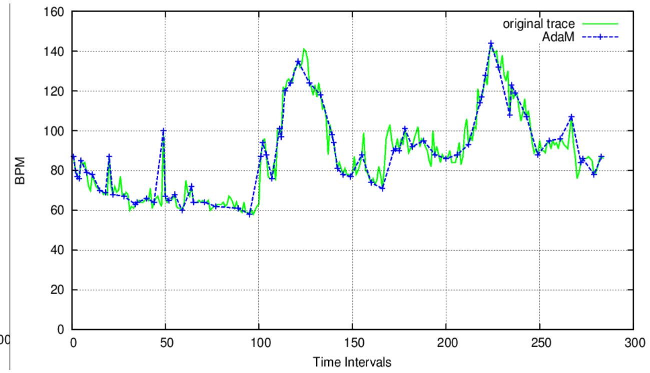
Disk I/O Trace
Carnegie Mellon RainMon Project



TCP Port Monitoring Trace
Cyber Defence SANS Tech Institute

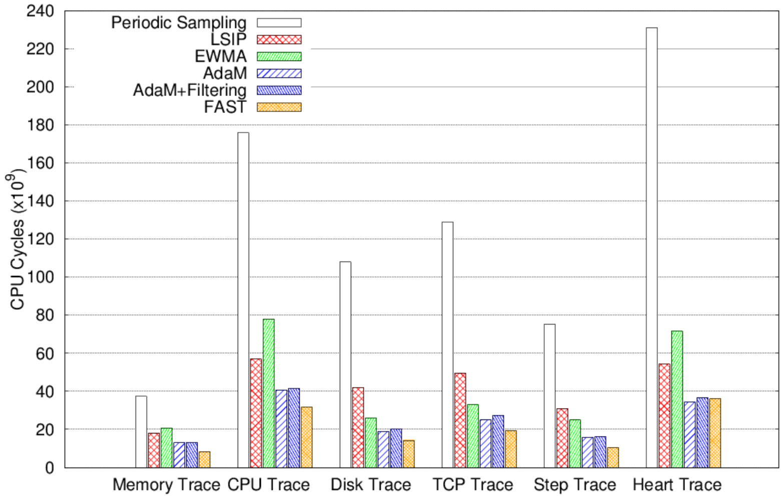


Step Trace
Fitbit Charge HR Wearable

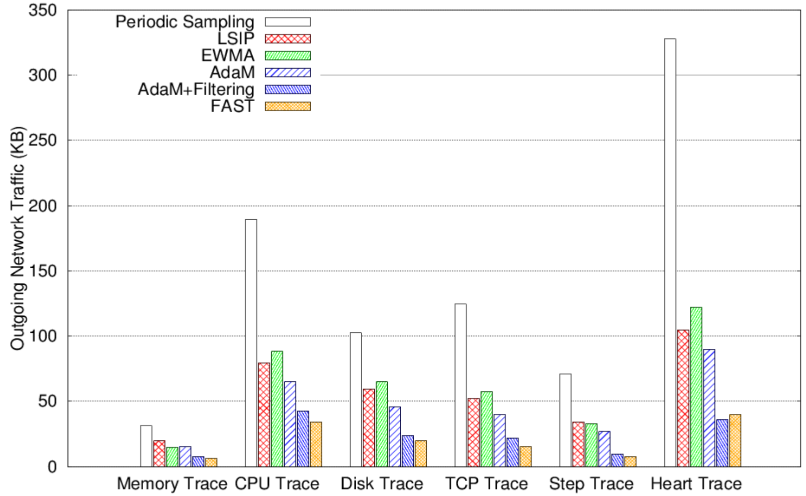


Heartrate Trace
Fitbit Charge HR Wearable

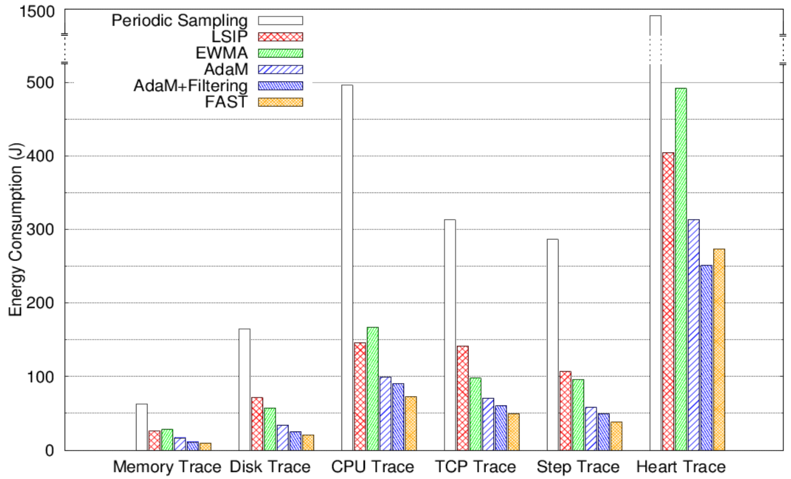
Overhead Comparison (1)



CPU Cycles



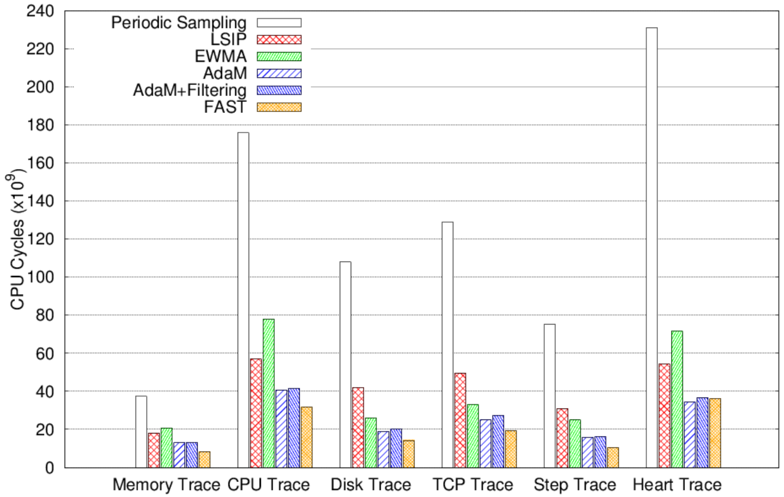
Network Traffic



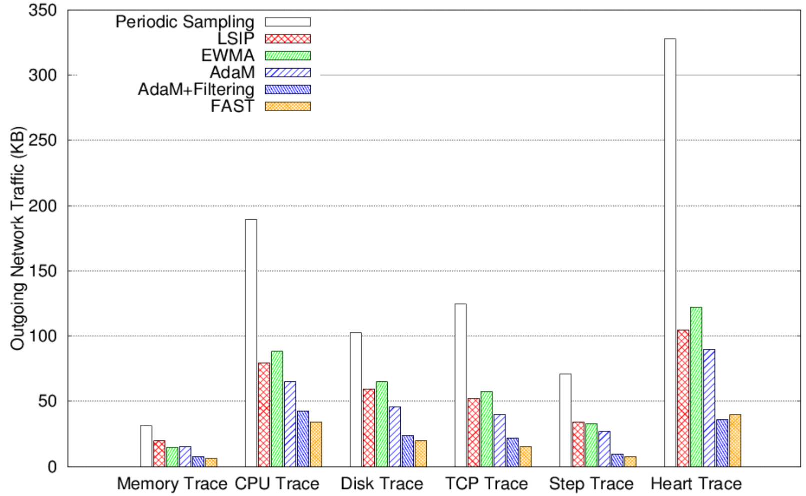
Energy Consumption

Filtering adds an overhead <2% to AdaM but benefits network traffic reduction by at least 32% and energy consumption by 12%

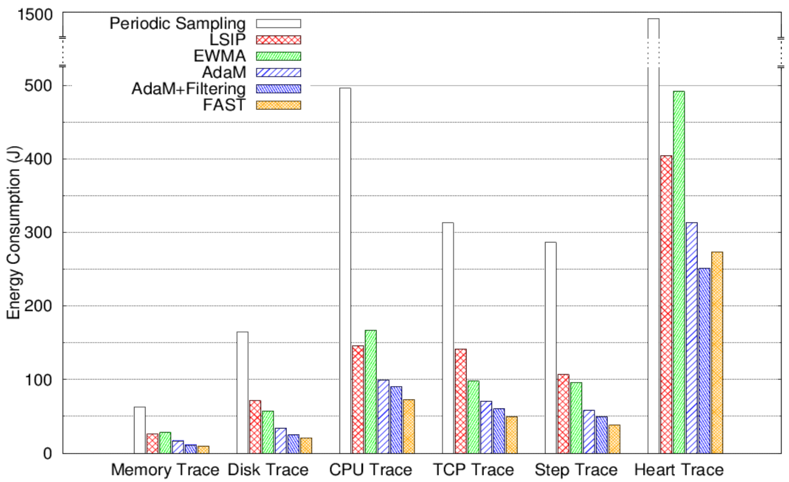
Overhead Comparison (2)



CPU Cycles



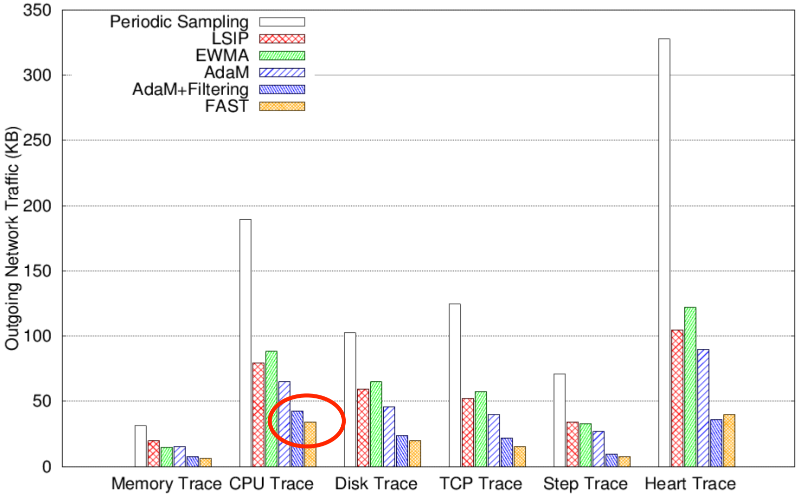
Network Traffic



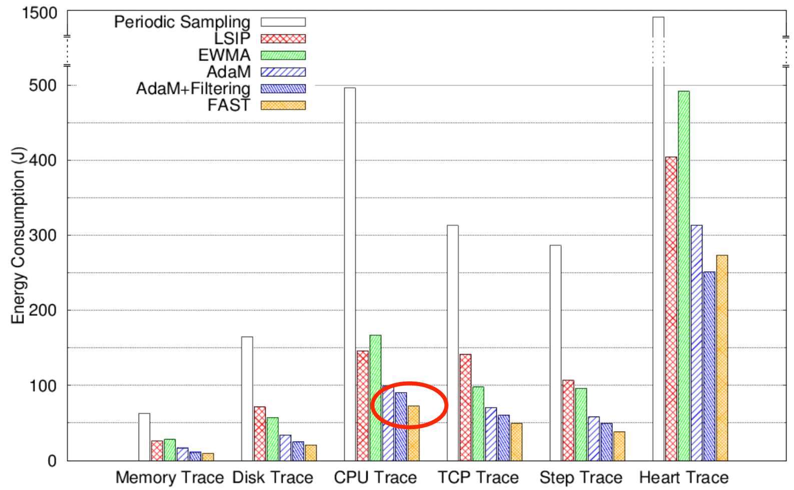
Energy Consumption

Data volume is reduced by 74%, energy consumption by 71%, while accuracy is always above 89% and 83% with filtering enabled

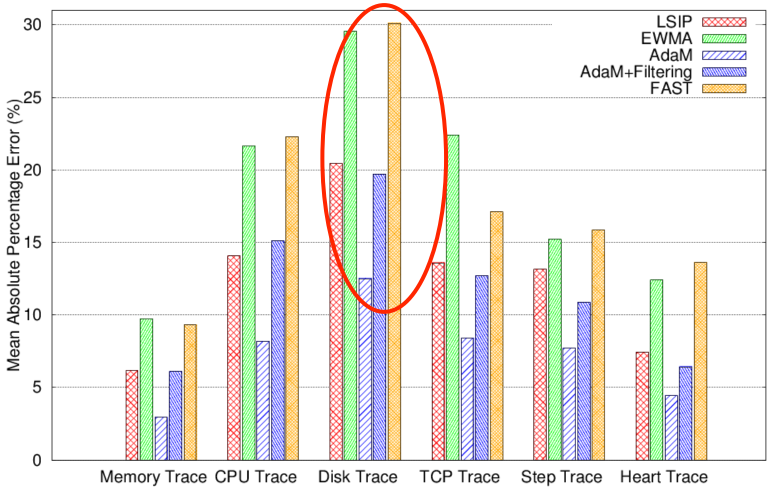
Overhead Comparison (2)



Network Traffic



Energy Consumption



Error (MAPE)

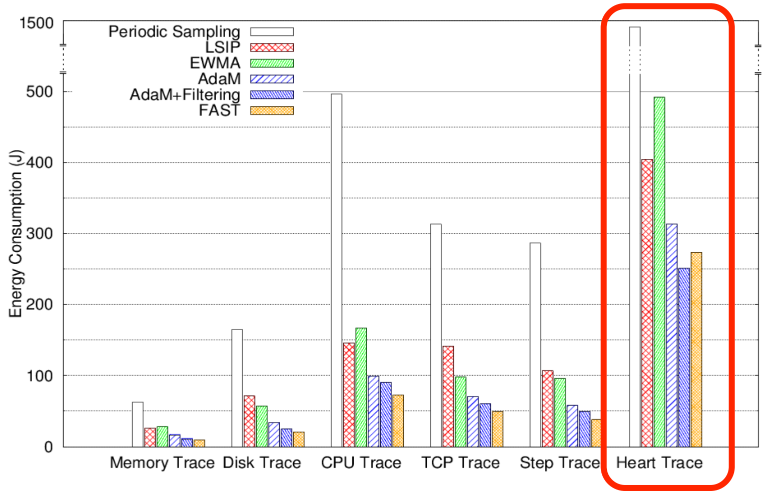
FAST's aggressiveness results in slightly lower energy consumption and network traffic

But, this does not come for free…

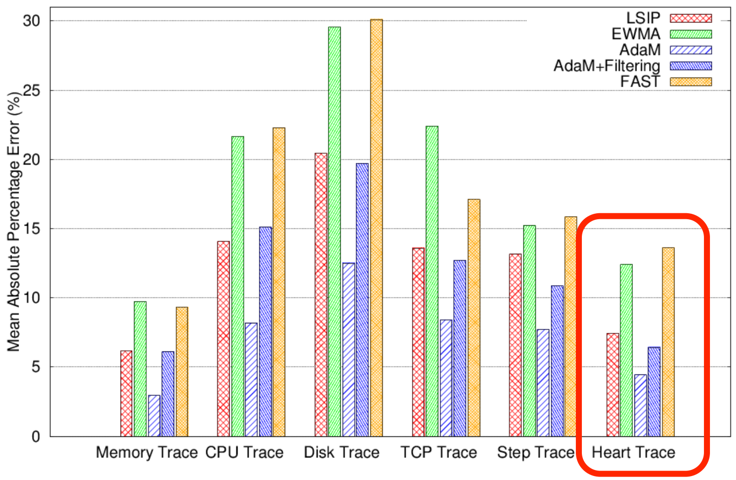
FAST features a large inaccuracy

AdaM achieves a balance between **efficiency** and **accuracy**

Overhead Comparison (3)



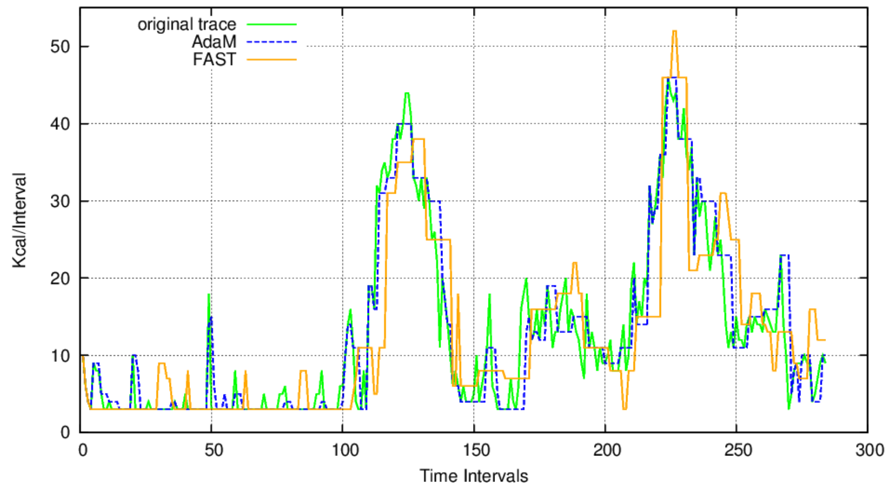
Energy Consumption



Error (MAPE)

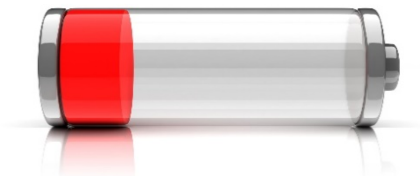
In the case of **heartrate monitoring** where signal analysis on AC wavelets reflected on wrist arteries is needed, **AdaM reduces energy consumption by 86%**

Calorie Counting is based on human body indicators (age, weight, height) and heartrate monitoring

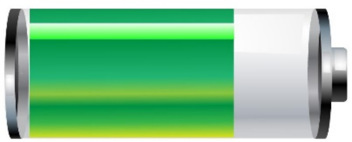


AdaM's MAPE grows from 6.42% in heartrate monitoring to 9.07% in calorie counting in contrast to FAST with 13.61% and 21.83% respectively

Overhead Comparison (4)



3 - 5 days

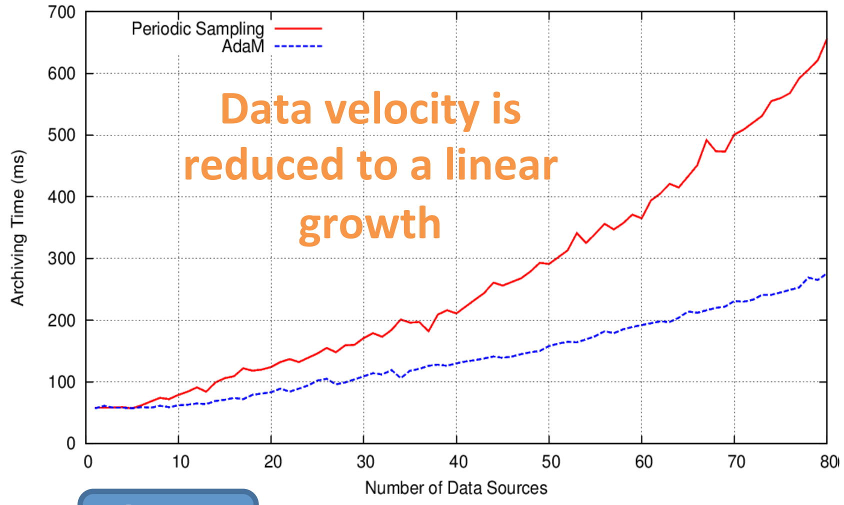


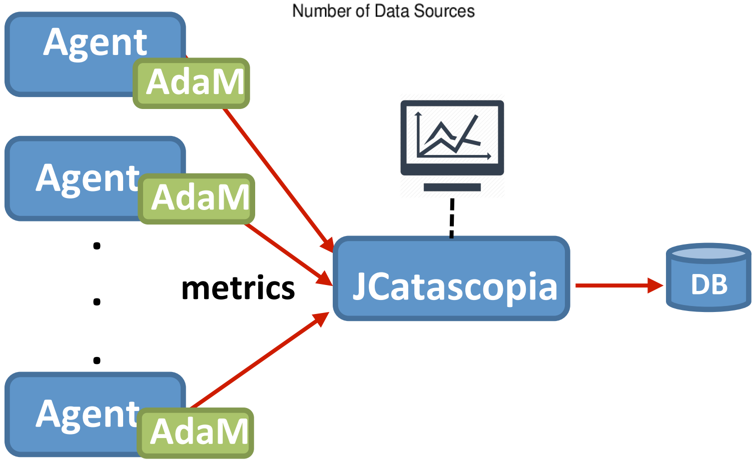
+ AdaM

7-8 days

Scalability Evaluation (1)

- Integrated AdaM to a data streaming system
- JCatascopia Cloud Monitoring System
  - JCatascopia Agents (data sources) use AdaM to adapt monitoring intensity
  - Archiving time is measured at JCatascopia Server to evaluate data velocity
- Compare **AdaM** over **Periodic Sampling**



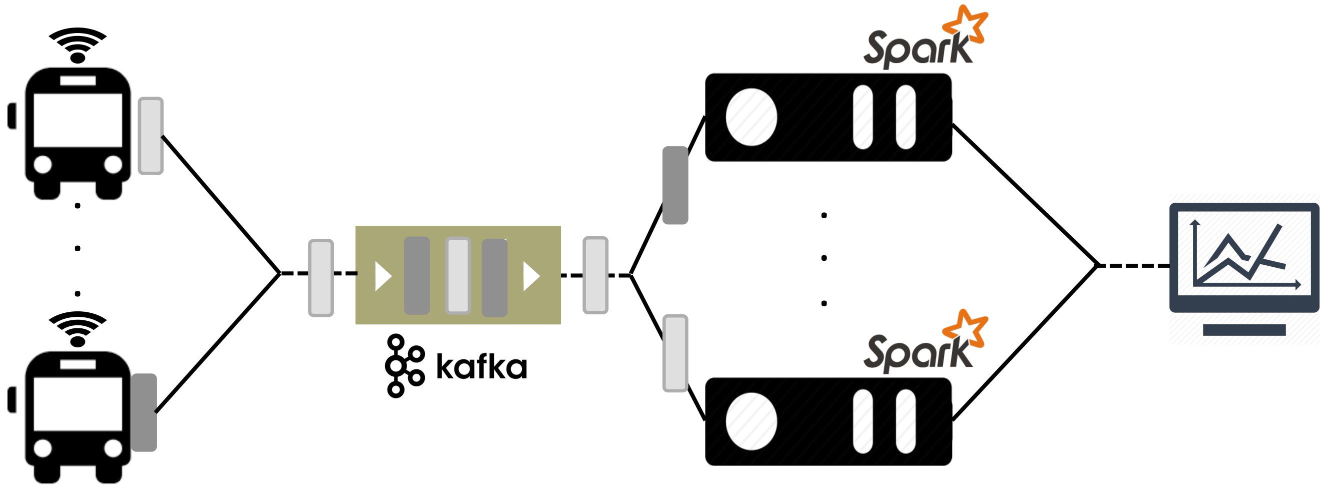


https://github.com/dtrihinas/JCatascopia

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

Scalability Evaluation (2)

- **Dublin Smart City Intelligent Transportation Service** (Dublin ITS)



1000 Buses* with GPS tracking sending updates to ITS with 16 params (e.g. busID, location, **current route delay**)

*Real data from Jan. 2014

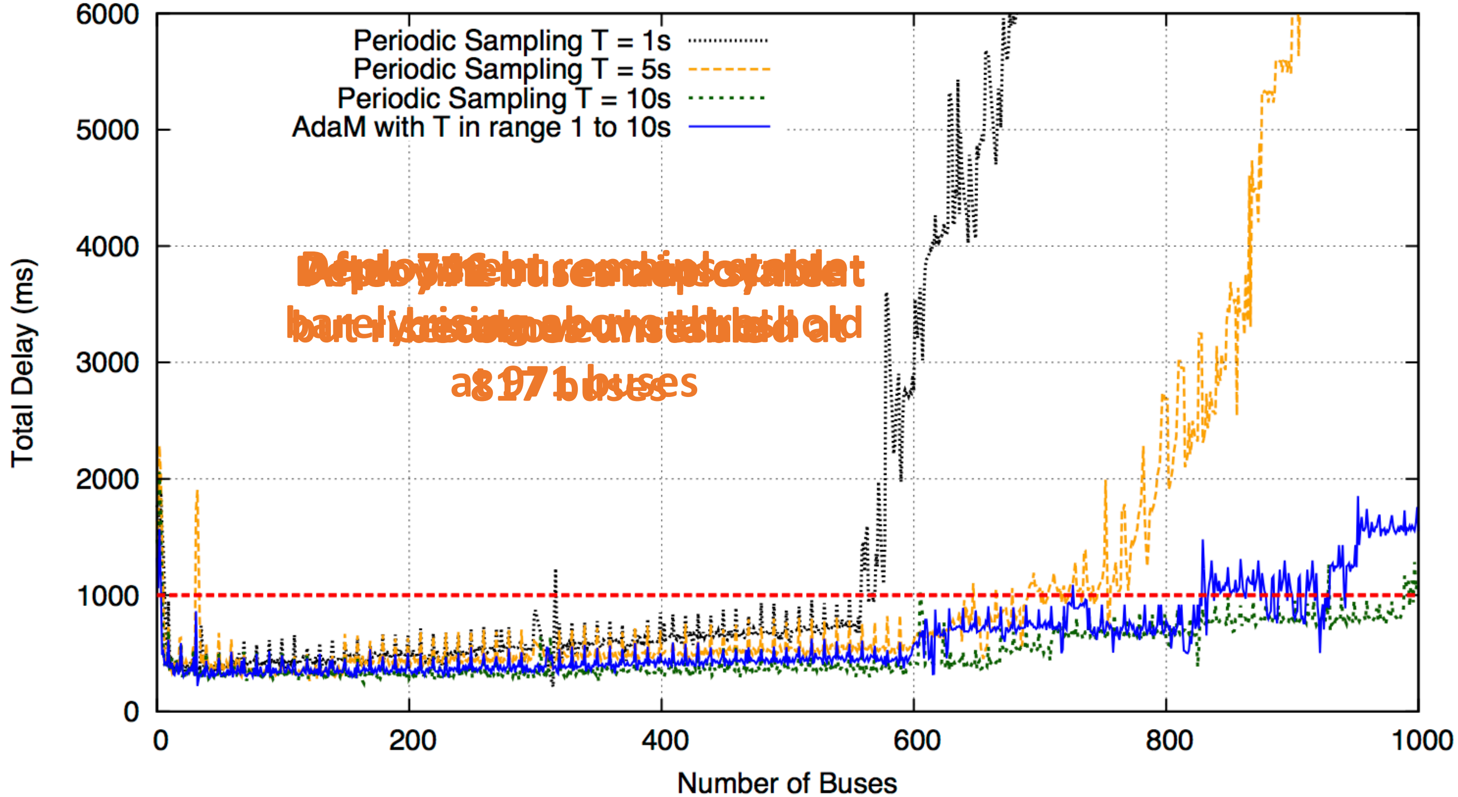
Apache Kafka queuing service on x-large VM (16VCPU, 16GB RAM, 100GB Disk)

Apache Spark cluster with 5 workers on large VMs (8VCPU, 8GB RAM, 40GB Disk)

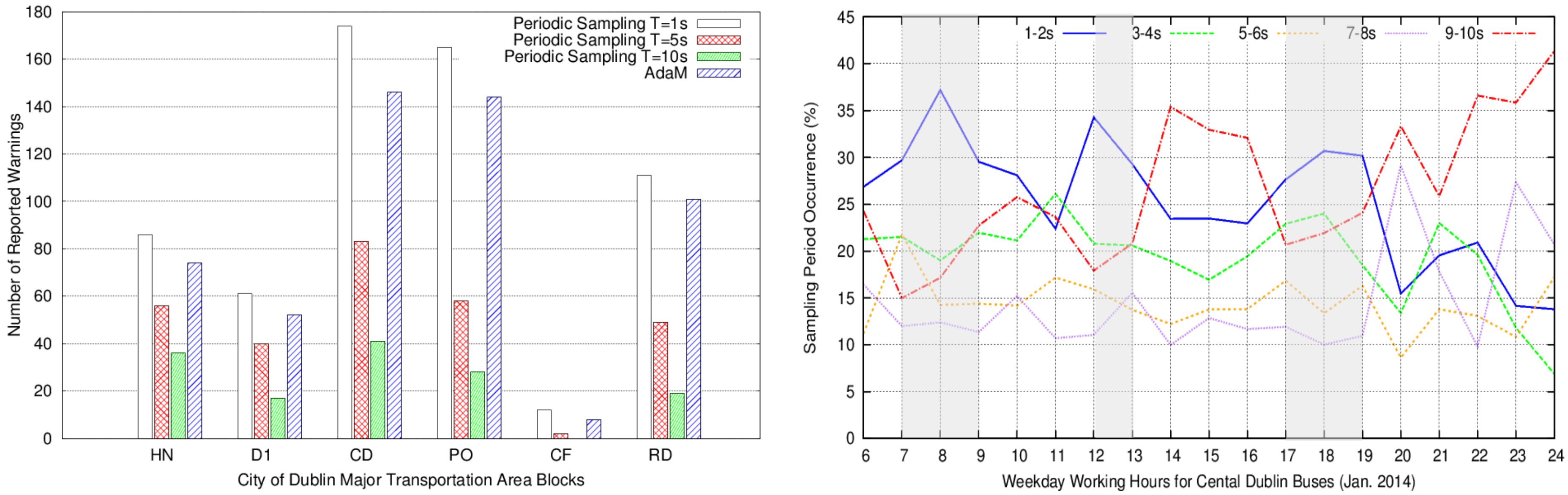
Alerts ITS operators when more than 10 buses in a Dublin city area, per 5min window, are reporting delays over 1 standard deviation from their weekly average

Scalability Evaluation (3)

- Spark total delay (processing + scheduling) for T=1, 5, 10 intervals and AdaM with max imprecision γ = 0.15



Scalability Evaluation (4)



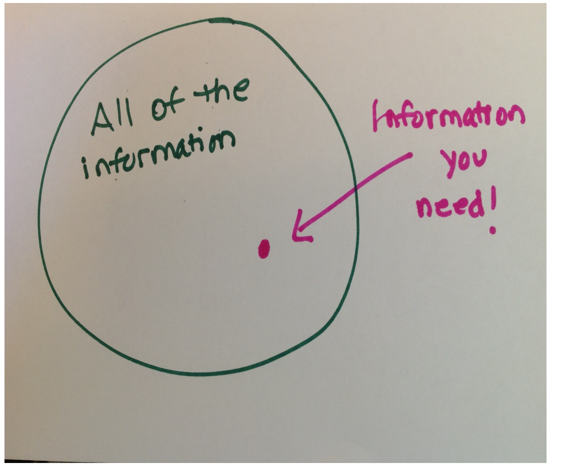
AdaM achieves an 83% average accuracy with >85% in all major Dublin areas

compared to 46% for T=5s and 17% for T=10s

High variety of sampling rates used throughout the hours of the day showing the need of adaptive sampling

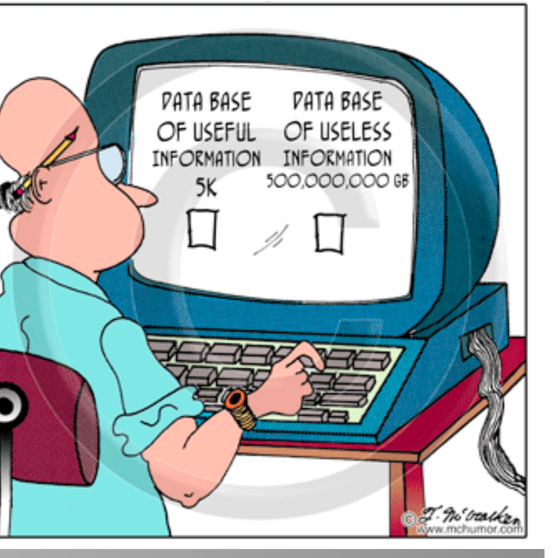
Conclusions

- Edge-mining on IoT devices is both resource and energy intensive
- Big data streaming engines struggle to cope as the volume and velocity of IoT-generated data keep increasing



The AdaM Framework

- Adapts the monitoring intensity based on current metric evolution and variability
- Reduces processing, network traffic and energy consumption on IoT devices and the IoT network
- Achieves a balance between efficiency and accuracy



Acknowledgements

PaaSport

co-funded by the
European Commission

Laboratory for Internet Computing
Department of Computer Science
University of Cyprus
http://linc.ucy.ac.cy