Energy Management for IoT

DYNAMIC POWER MANAGEMENT LABORATORY

Alessandro Landra (s284939)
Luca Dalmasso (s281316)

Course: Embedded Systems (Computer Engineering branch)

Date: 20/11/2021

Goal of the lab

Given a PSM and some workloads, evaluate how energy saving changes as a function of

- The DPM policies
  - Timeout policy (Part I)
  - History based policy (Part II)

- The workloads
  Evaluations have been done on two type of workloads that are typical for the IoT world.
  - Sensing : read data from sensors (polling)
  - Computing: elaboration of the data
  - Waiting: repeat every two minutes

- The PSM
  Evaluations have been done on a simple 3 state PSM (Figure 1).

Power State Machine

The PSM under analysis is a simple 3 state PSM where only 2 possible low power states are possible: Idle or Run.

The Idle state is characterized with a very short Transition Time, thus a very short (negligible with the workloads considered) Break Event Time, but not as good as the Sleep state in terms of power consumption.

The Sleep state as mentioned before has an extremely low power consumption with respect to the other two states, but the transition times are very high with respect to the Idle state, this means that the Break Event Time is higher.

Figure1: PSM states with transition times and power states

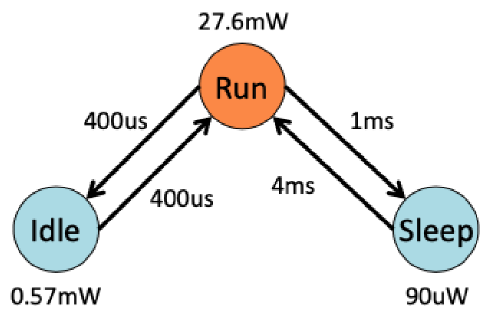


Table 1: Power states and transition costs (mW)

	RUN	IDLE	SLEEP
RUN	0	0.025	0.02
IDLE	0.025 (*)	0	-
SLEEP	0.5	- (*)	0

(*) $Ptr_{idle \to run} = 0.025\ mW$, power wasted for the Idle->Run transition.
$Ptr_{Sleep \to Idle}$ = - mW, Sleep->Idle transition is not possible for the PSM under test.

Table 2: Power states and Break Event Time (ms)

Break event Time (Tbe) is computed using the following formula:

- Tbe= $\begin{cases} Ttr, \quad when\ \ Ptr < Pon \equiv Prun \\ Ttr + Ttr\frac{Ptr - Pon}{Pon - Poff}, \quad when\ Ptr > Pon \equiv Prun \end{cases}$

Transitions considered	Transition power	Break Event Time
RUN→IDLE RUN←IDLE	Ptr=$(Prun_{idle} + Pidle_{run}) = 0.050\ mW \ll Prun$	Tbe = Ttr = 0.8 ms
RUN→SLEEP RUN←SLEEP	Ptr=$(Prun_{sleep} + Psleep\_run) = 0.52\ mW \ll Prun$	Tbe = Ttr = 5 ms

Workloads

In this experiment only two similar workloads have been used, both characterized by a 3 phases that are repeated every 2 minutes (Figure3).

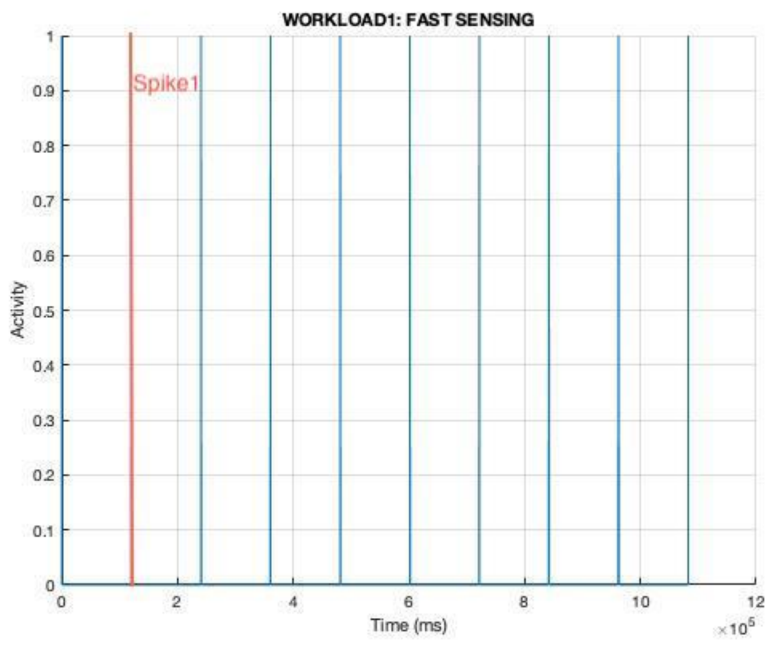
Sensing phase: wait for the sensors for a given period (during which the system should move, if possible, in Idle or Sleep), retrieve data from sensors when they are ready (in this period the system should be in Run state, otherwise data is lost), repeat until all data is retrieved.

Computing phase: once the sensing phase is done there is a computing phase, where the system elaborates the data collected.

Waiting: after Sensing and Computing there is a long period of Waiting, where the system should of course turn in Sleep or Idle because there's nothing to do.
After the waiting phase it will start a new cycle of Sensing→Computing→Waiting→…

Figure3: example of workload where Sensing→Computing→Waiting phases are repeated every 120000 ms (2 minutes).
Due to the fact that Waiting state is huge with respect to the other 2 phases, in this picture those spikes represent Sensing→Computing (Figure2)

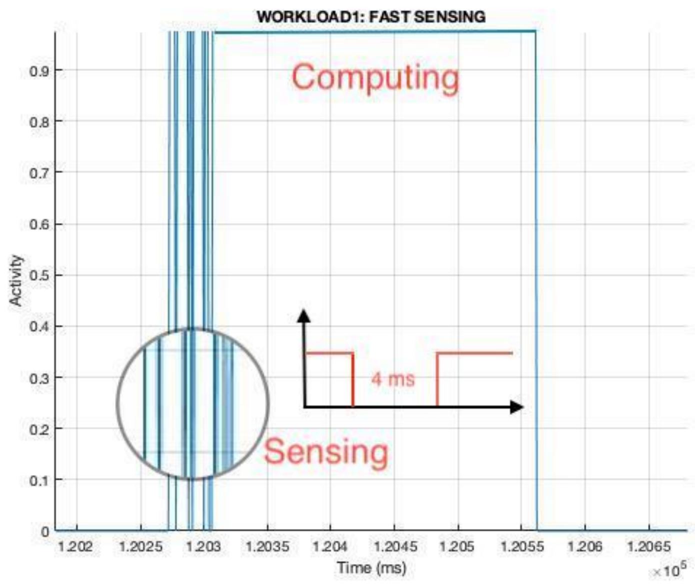
Figure2: Workload1 activity



The differences between the two workloads are in the Sensing phase:

- Workload1: Fast sensing, sensors produce new data every 4 ms (Figure3)
- Workload2: Slow sensing, sensors produce new data every 100 ms (Figure3)

Figure3: Zoom on the Sensing→Computing phase of workload1, after the first 2 minutes (Spike1 in figure2).
For the workload2 instead of having 4ms, there are 100 ms of waiting for sensors to acquire



Part I: Timeout policy

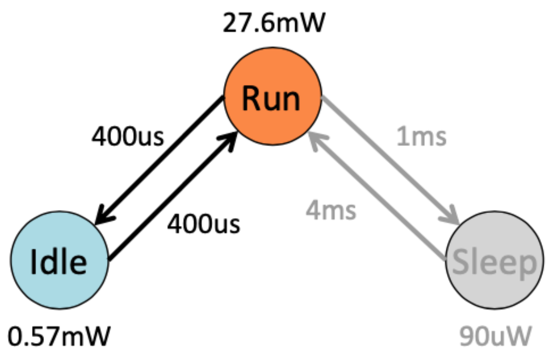
In this first part of the experiment, the evaluation is made by using the Timeout policy, so basically it consists in shutting down the system into one of the two low power states whenever the idle period (period where the activity is 0) observed becomes longer than a given timeout Tto.

We considered the following 2 cases:

1. Tto >= Tbe
   In this case the policy is safer because ensures that the cost of shutdowns is amortized.
2. Tto < Tbe
   If timeout is lower than the Break Event Time basically 2 things happen:
   a) The system won't be able to resume the activity in time.
      In order to evaluate this parameter, the delay which affects a non-properly scheduled task, we modified the code to be able to compute the overall delay caused by this phenomenon.
   b) The system may save even more power than the case Tto==Tbe, but, a part from increasing the delays, as expected, consumes more energy than the case Tto==Tbe (see Figure13 and Figure14).

RUN→IDLE TRANSITIONS

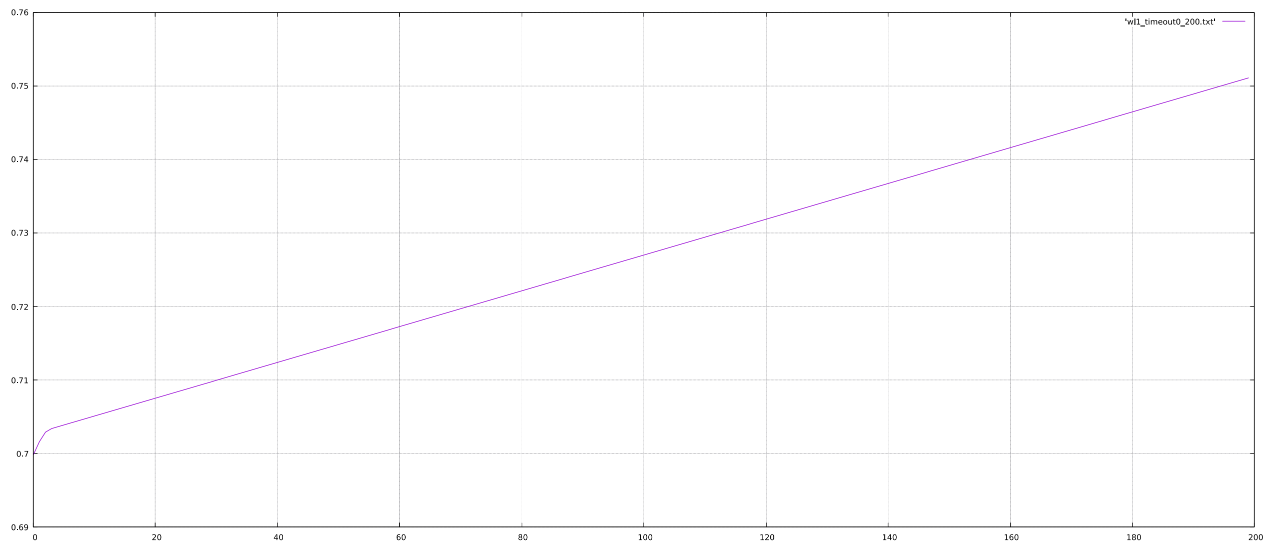
Figure5: PSM is the following



In this case the Tbe is small enough to be considered negligible, so the analysis done here are all for Tto >= Tbe.
What was expected to happen is that by increasing the timeout, as a consequence, the energy increases linearly. (It could eventually saturate, but it has no sense to try, because, due to Waiting phases, the timeout should be too big to be meaningful)

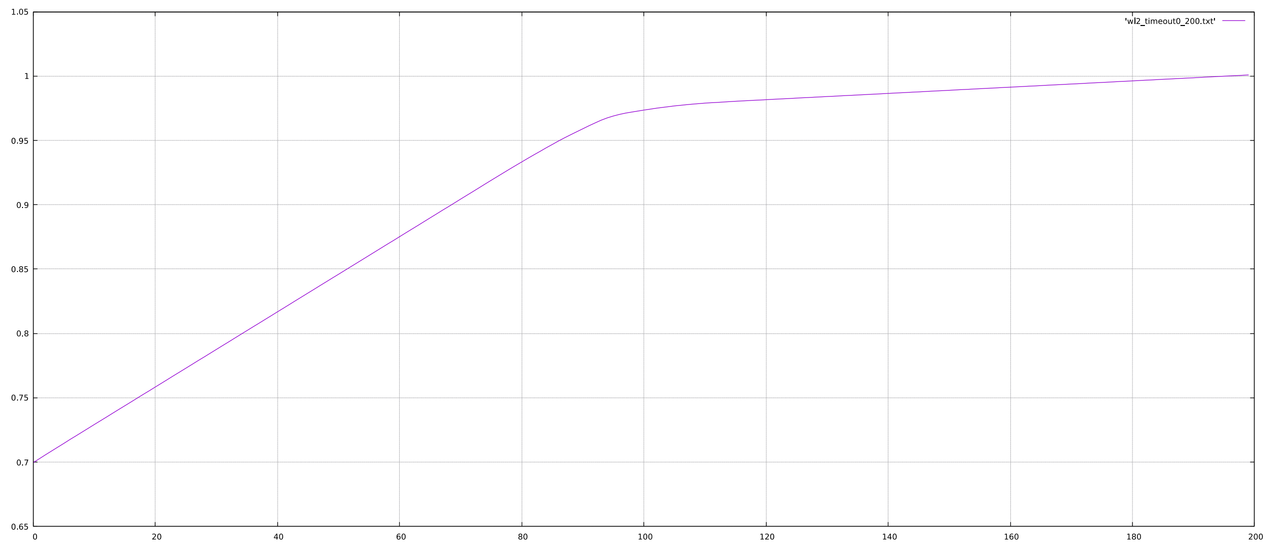
WORKLOAD1:
It is possible to see two distinct points in the curve of the energy (Figure6), those are due to the fact that if Tto<4ms (time of the fast sensing) the PSM is able to save also some of the Sensing energy, in particular the energy of the idle times during which the system is waiting for the sensors (those idle times are 4 ms in average).
If, instead, the Tto>=4ms the PSM is not able to save energy during the Sensing phases, so a lower derivative of the curve is expected after this Tto value.

Figure6: workload1, energy (y) in function of timeout increasing (x)



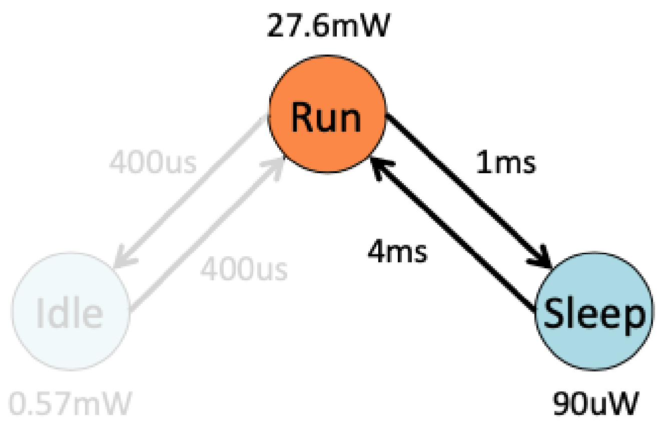
WORKLOAD2:
Same consideration as workload1, but the derivative here changes around 100 ms of Tto, because here the slow sensing produces idle times, in the Sensing phase, that are longer, 100 ms on average.

Figure7: workload2, energy (y) in function of timeout increasing (x)



RUN→SLEEP TRANSITIONS

Figure8: PSM is the following



The PSM now looks very different from before, in particular there are 4 aspects to be considered (remind table1 and table2):

1) $\frac{Ptr_{Run \to Sleep}}{Ptr_{Run \to Idle}} \approx 10$

2) $\frac{Ttr_{Run \to Sleep}}{Ttr_{Run \to Idle}} \approx 6$

3) $\frac{PSleep}{PIdle} \approx \frac{1}{6}$

4) Tbe is not small enough to be neglected

TIMEOUT >= TBE

This first part of the analysis doesn't take into consideration violation of the Break Event Time.

As a consequence of points 1, 2, 3 we can derive the following conclusion:

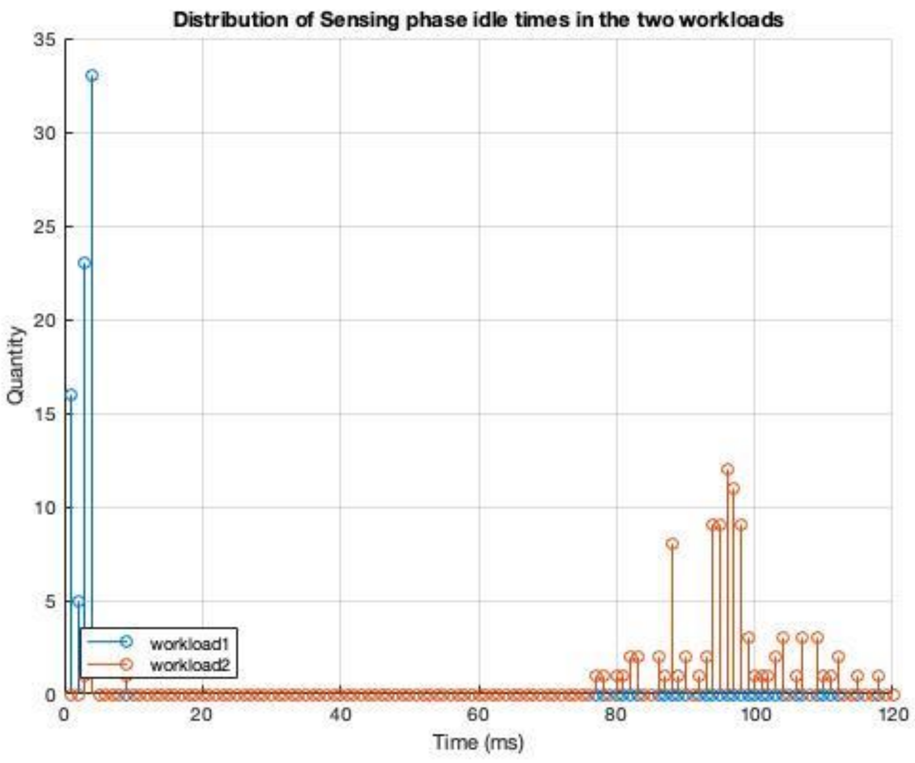
The RUN→IDLE PSM has a transition time (and so transition power) that is almost negligible, so it means that even with a short time spent in Idle, the system can still save power.

Instead, the RUN→SLEEP PSM, with respect to the RUN→IDLE PSM, consumes 1 order of magnitude more!
So, even if the Sleep low power state is much more efficient that the other one, the

timeout must be chosen with more care, because if the amount of time in which the system stays in Sleep is not big enough, it might lead to a non-optimal solution.

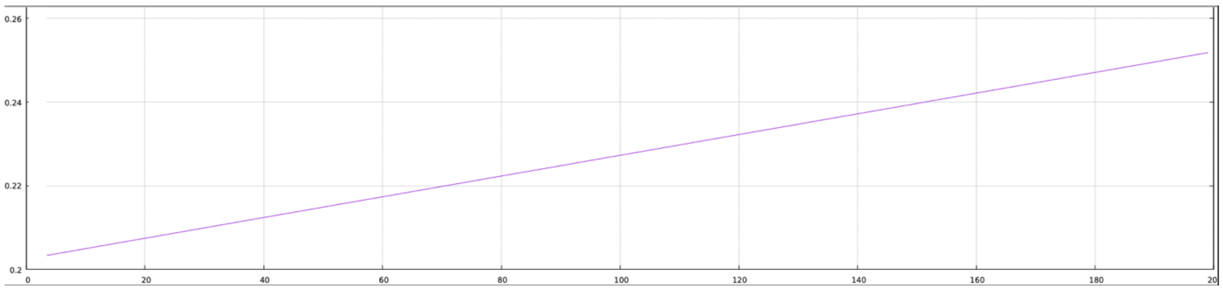
In order to demonstrate this behaviour is better to take a look at the following plot:

Figure9: Distribution of Sensing phases' idle times for the two workloads



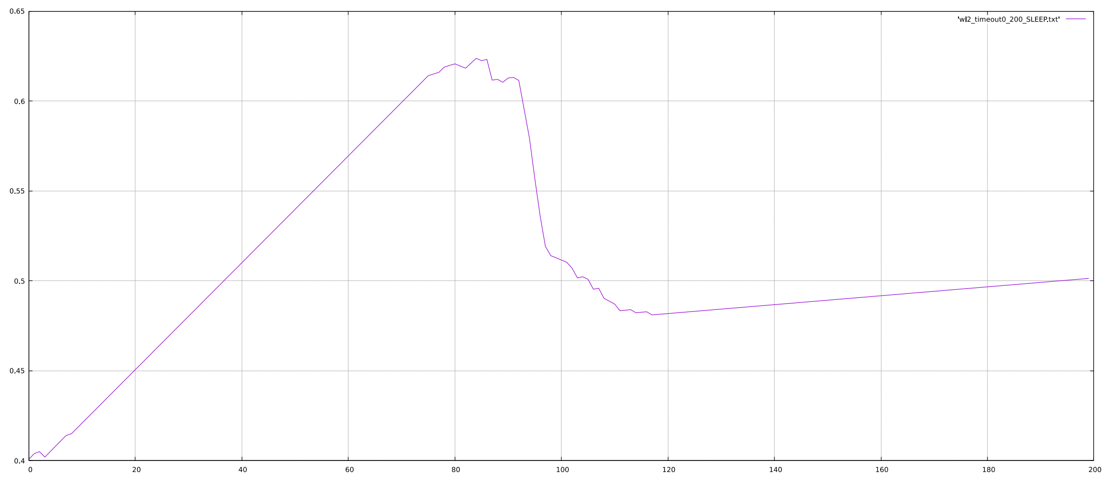
The behaviour discussed before is of course not valid for workload1 because all idle times are less than the Tbe, so the energy for the workload1 always grows linearly with respect to the Timeout (Figure10)

Figure10: workload1, energy (y) in function of timeout increasing (x)



Instead, for the workload2 is a little bit more problematic.
The distribution of idle times, due to the slow sensing, is concentrated between 80 and 120 ms, so, in this case, if the time spent in the Sleep state is not big enough to compensate the cost of the transitions, the solution is no more optimal.
As a consequence, timeouts < 80 or > 100 are expected to have better results.(Figure11)

Figure11: workload2, energy (y) in function of timeout increasing (x)

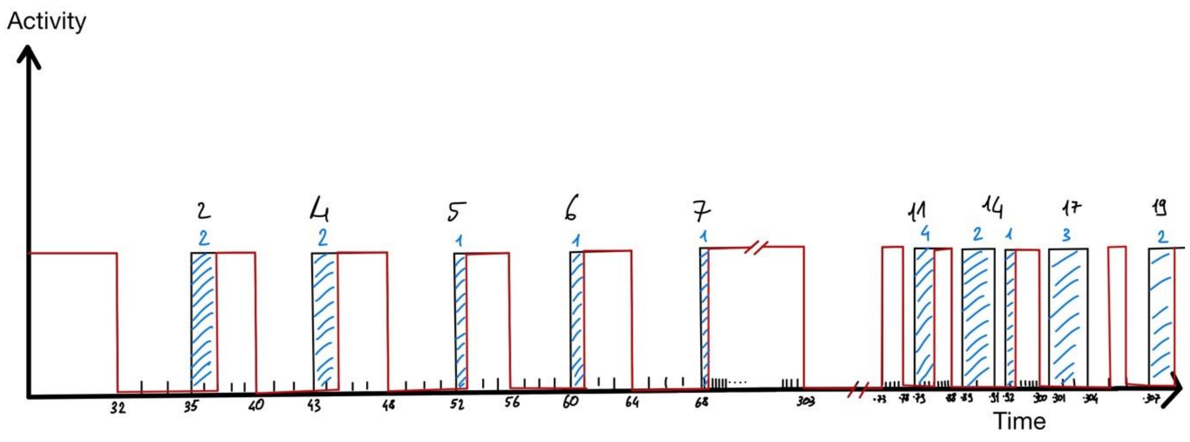


TIMEOUT < TBE

In this extra point, an analysis has been performed to evaluate the effects of both Delay and Energy consumption when Tto<Tbe.
The analysis is effective only for the RUN→SLEEP PSM.

In Particular if Tbe is not respected then the system will never be able to resume to active state when needed (even with a pre-wakeup) and in the worst case some tasks are totally missed due to the delay.(Figure12)

Figure12: First Sensing phase of the workload1, Tto=0, in blue are figured all delays with respect to the ideal case when Tbe=0.



In order to get the real numbers, we had to modify the code in order to make the simulator able to take into consideration the transition time while deciding the next state.

OSS: All the modifications to the original code are detailed and commented in the src/dpm_policies.c source file.

This are the overall results obtained:

Figure13: workload1 comparison.

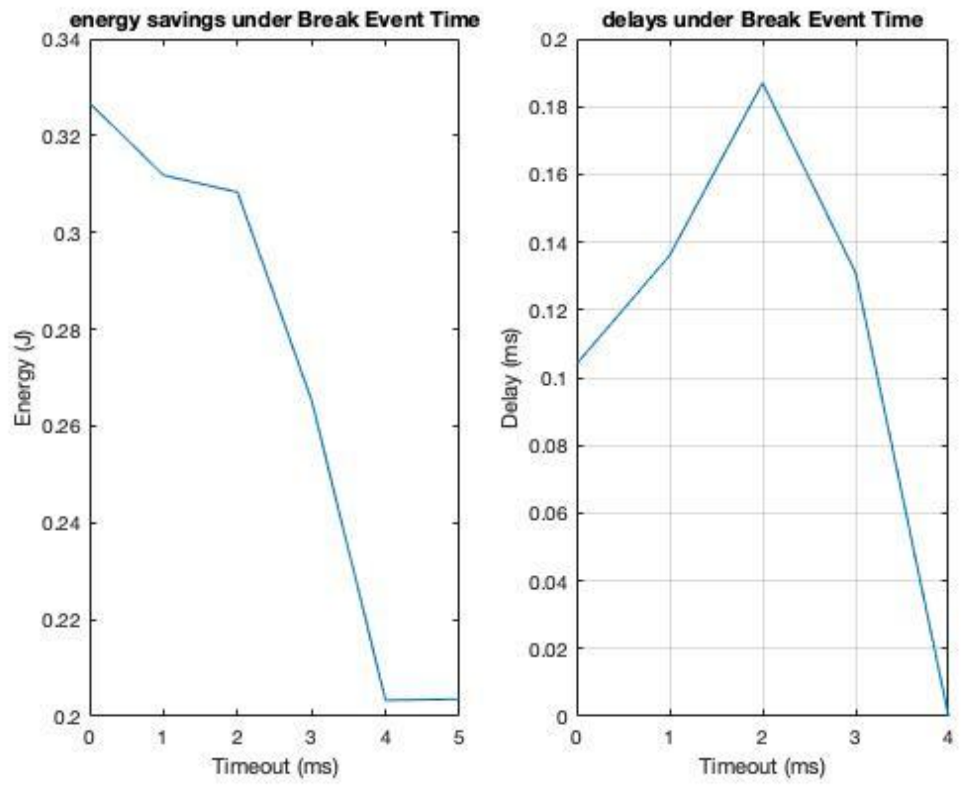
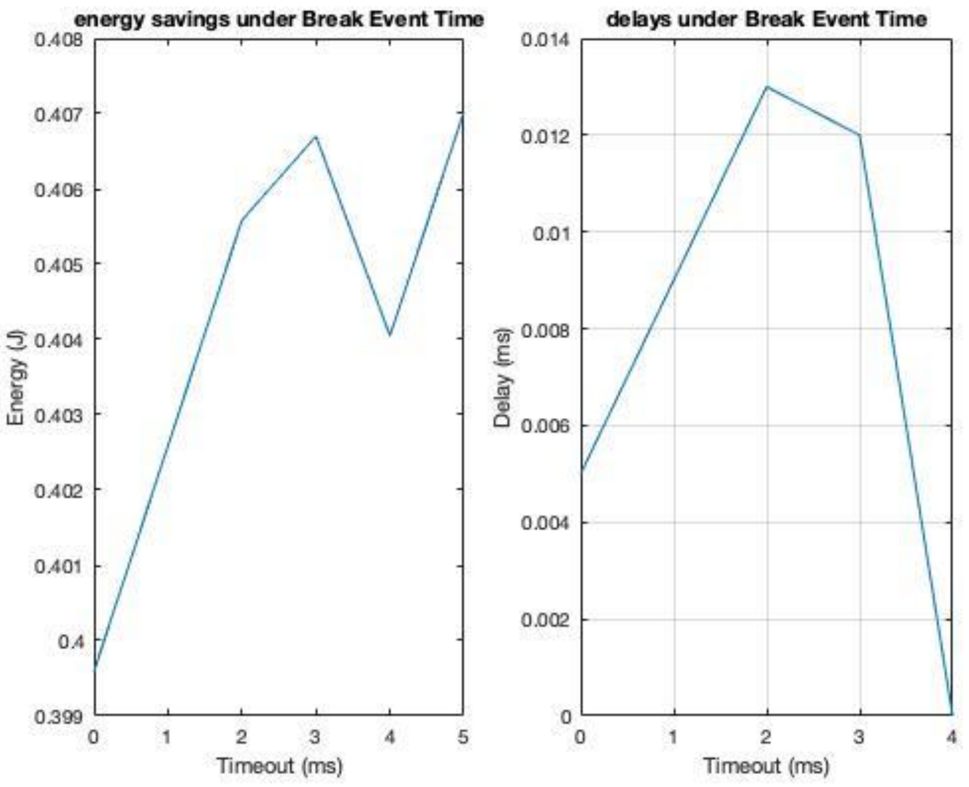


Figure14: workload2 comparison.



CONCLUSIONS OF PART I

This last part of the experiment is an extra work, done just to point out what are the effects of the non-idealities that limit the performances of a PSM.
Even with a perfect oracle who knows exactly when to switch off-on, the overheads due to the Break Event Time cannot be avoided.

An interesting point is also to observe that, in case of the workload2 (Figure14), is possible to reach the best solution in terms of power saving by violating the Tbe.

Part II: History based predictor policy

In this second part of the experiment, the policy is a little bit different from the Timeout: the PSM now decides when to shut down or not, by trying to predict the duration of the coming idle time. This prediction is based on a weighted average of the previous 5 idle times registered.

This is the formula used for the predictions:

Tpred = $0.45 * h[4] + 0.25 * h[3] + 0.15 * h[2] + 0.10 * h[1] + 0.05 * h[0]$;

Where h(*hystory*) is the vector of the last 5 idle times.

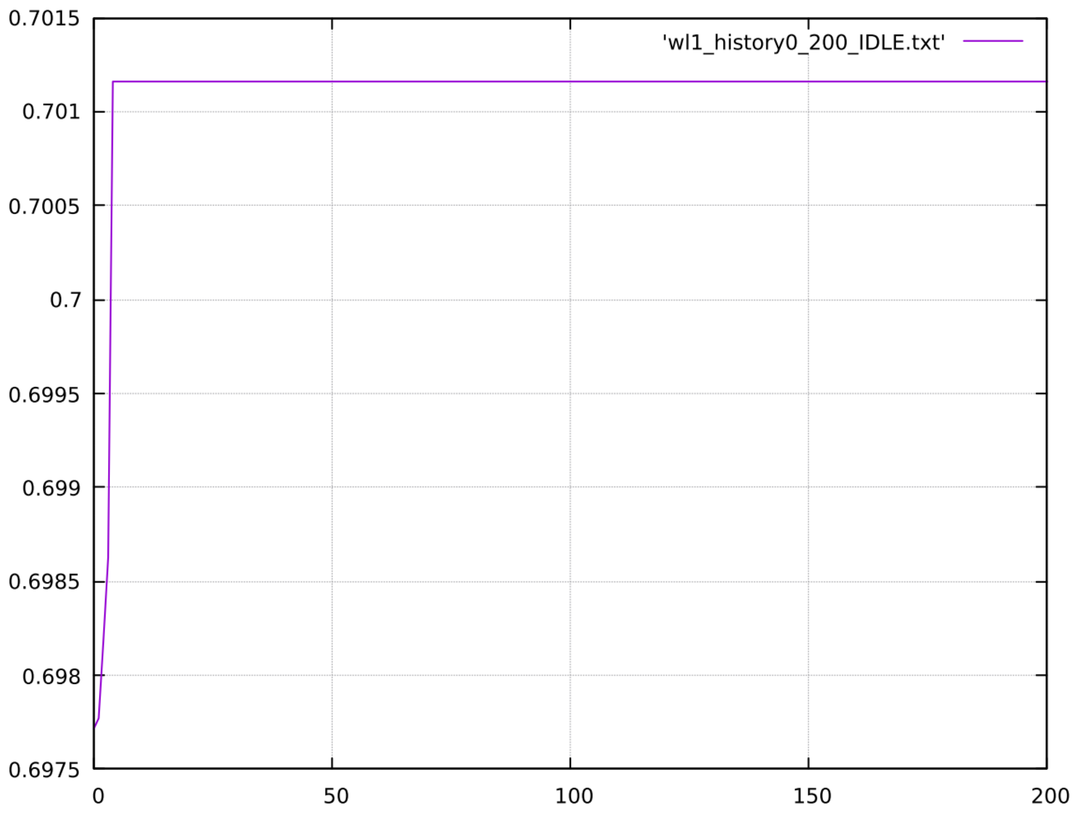
Then, if the idle time calculated in this way is **greater** than a certain threshold, passed as a parameter, the current state is set to IDLE or SLEEP, depending on the desired PSM analysis.

RUN→IDLE TRANSITIONS

In workload 1, long sequences of quick idle periods (3-4ms) are intermitted by few longer ones of about 120000ms; this event causes the predictive idle time to be really long until this period exits the window of considered ones. Therefore, we can notice that for thresholds higher than 4ms, the behavior is the same until we reach considerable higher values, while there is an increasing slope at the beginning due to the presence of different small idle times, which make the predictive idle time oscillate under 4ms.

Concerning workload 2, the behavior described for workload 1 remains, with few 120000ms long idle times, but with an average idle time, in fast sensing, around 100ms. This means that the predicted one oscillate in a range with that middle point, until the long one enters the window. More variations can then be appreciated as expected from the distribution of sensing phases (**Figure9)**

Figure15: workload1, energy (y) as a function of increasing threshold (x)



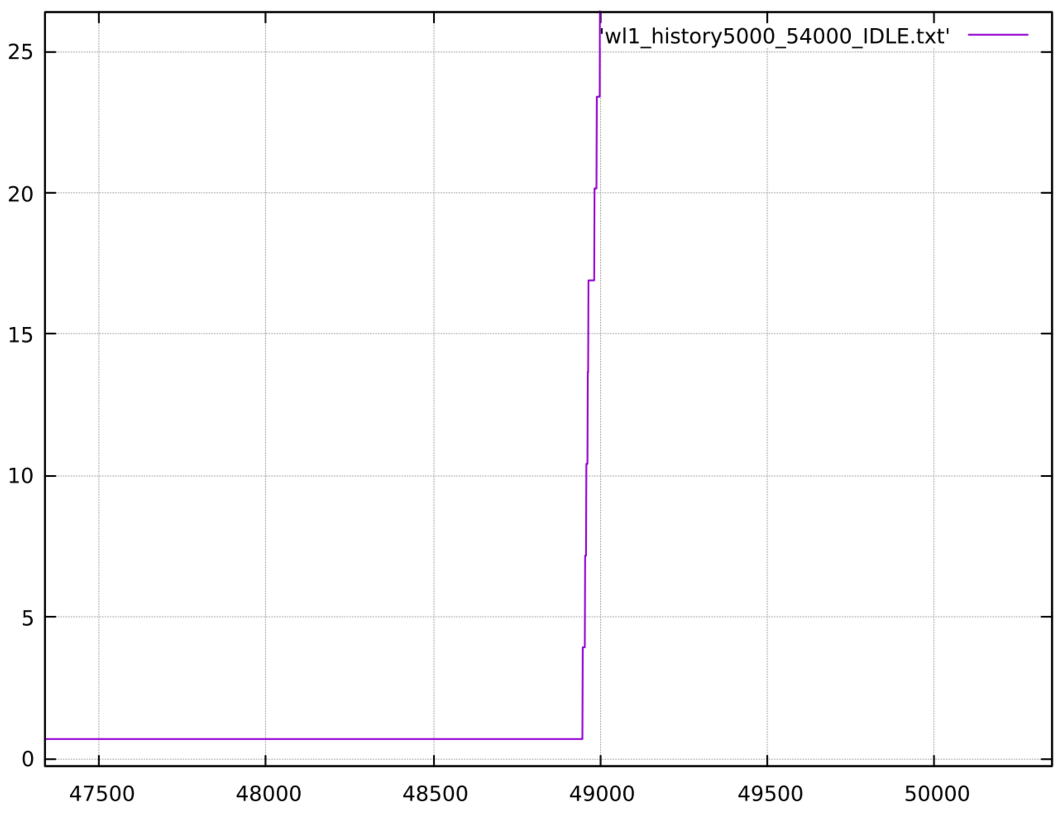
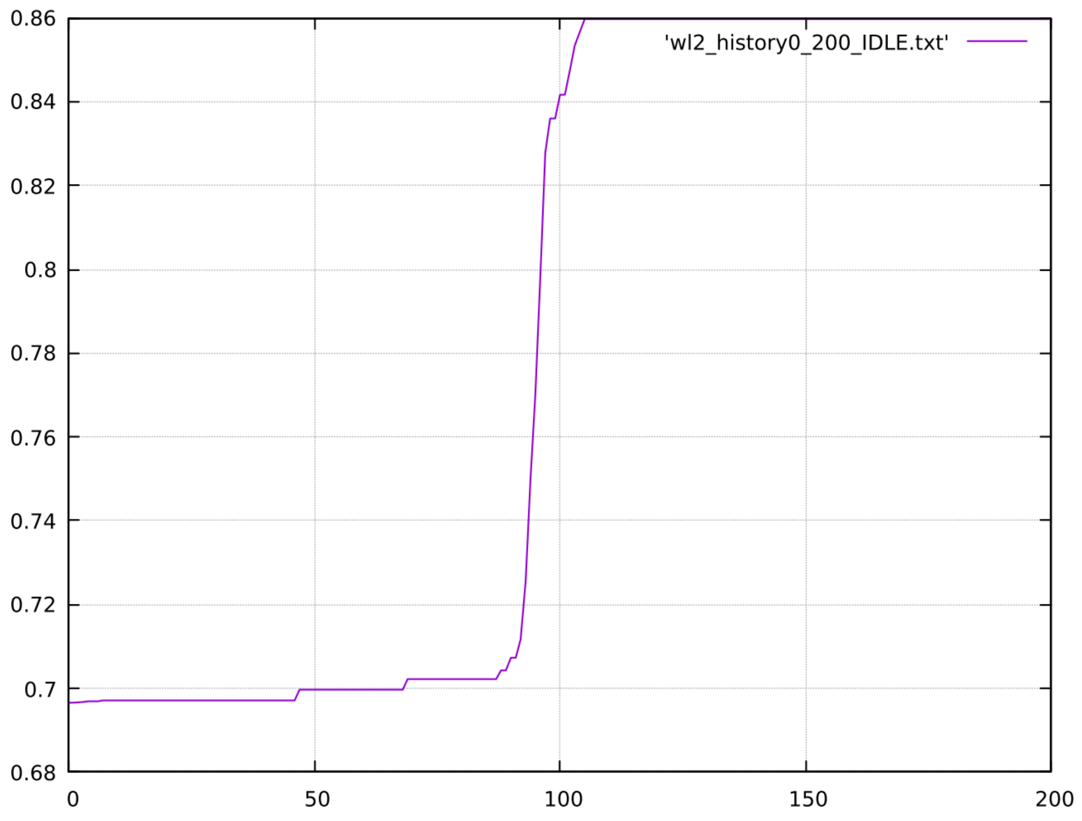
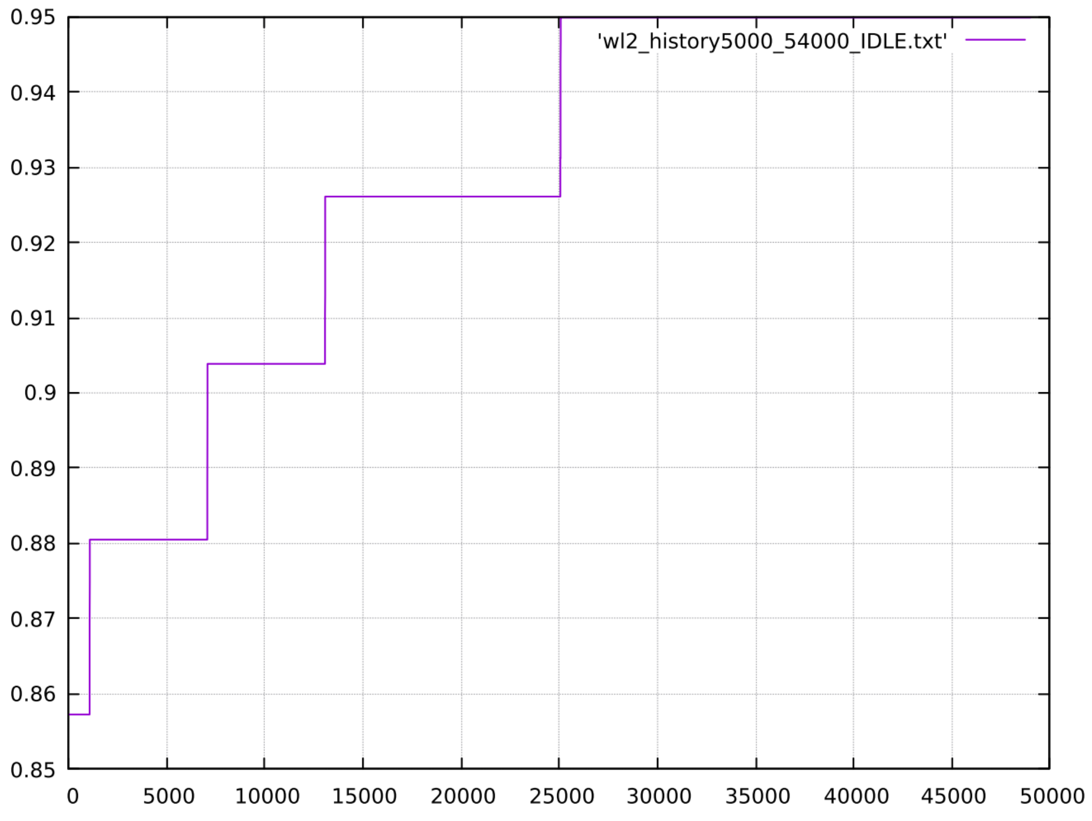


Figure16: workload2, energy (y) as a function of increasing threshold (x)





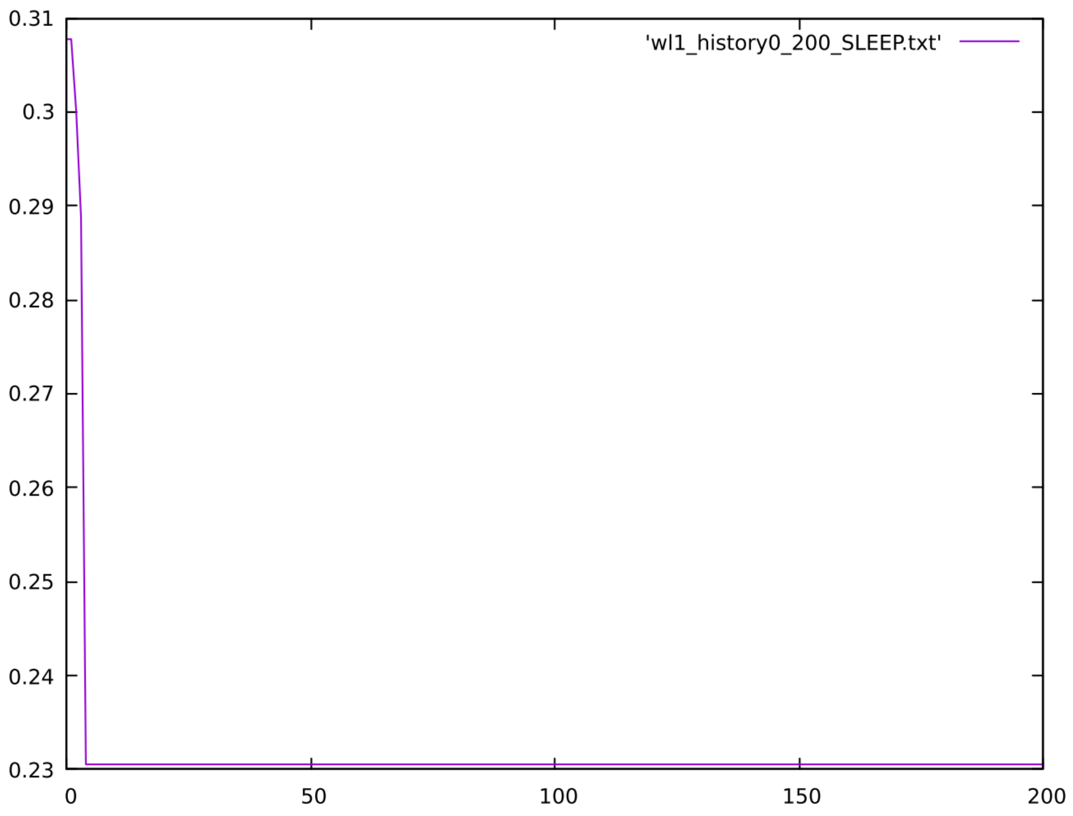
RUN→SLEEP TRANSITIONS

All the previous consideration about the workloads and the prediction window are still valid, but a non-negligible Tbe must be taken into account.

In particular, all the predictive idle times below the currently considered threshold are switched to the SLEEP state, even the ones that are not long enough to guarantee the Tbe to be respected. This causes a reduction in the number of transition, as the system needs to wait for Tbe even when the supposed idle time is smaller, but the cost of every transition is about 10 times the one for the RUN-IDLE/IDLE-RUN. This is the reason of the descending slope of the beginning of the energy curves, in which the high number of transitions increases the power consumption.

The overall power will be reduced, with a consequent introduction of delays, as discussed in the *TIMEOUT < TBE* section.

Figure17: workload1, energy (y) as a function of increasing threshold (x)



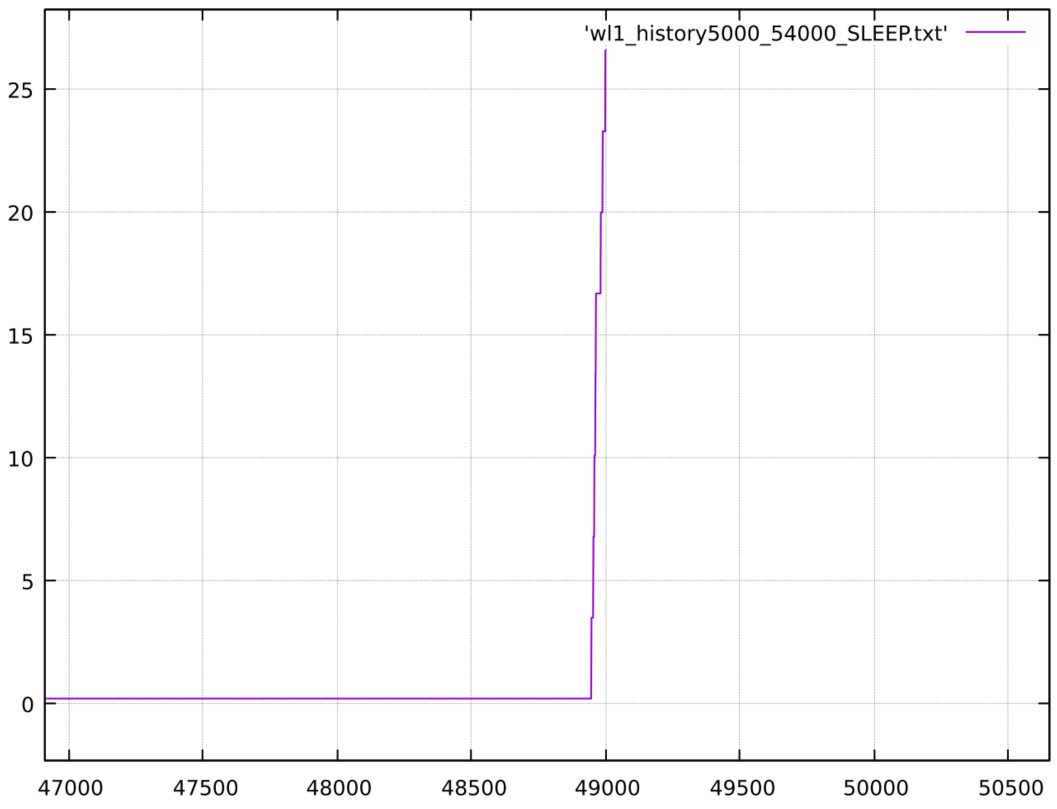
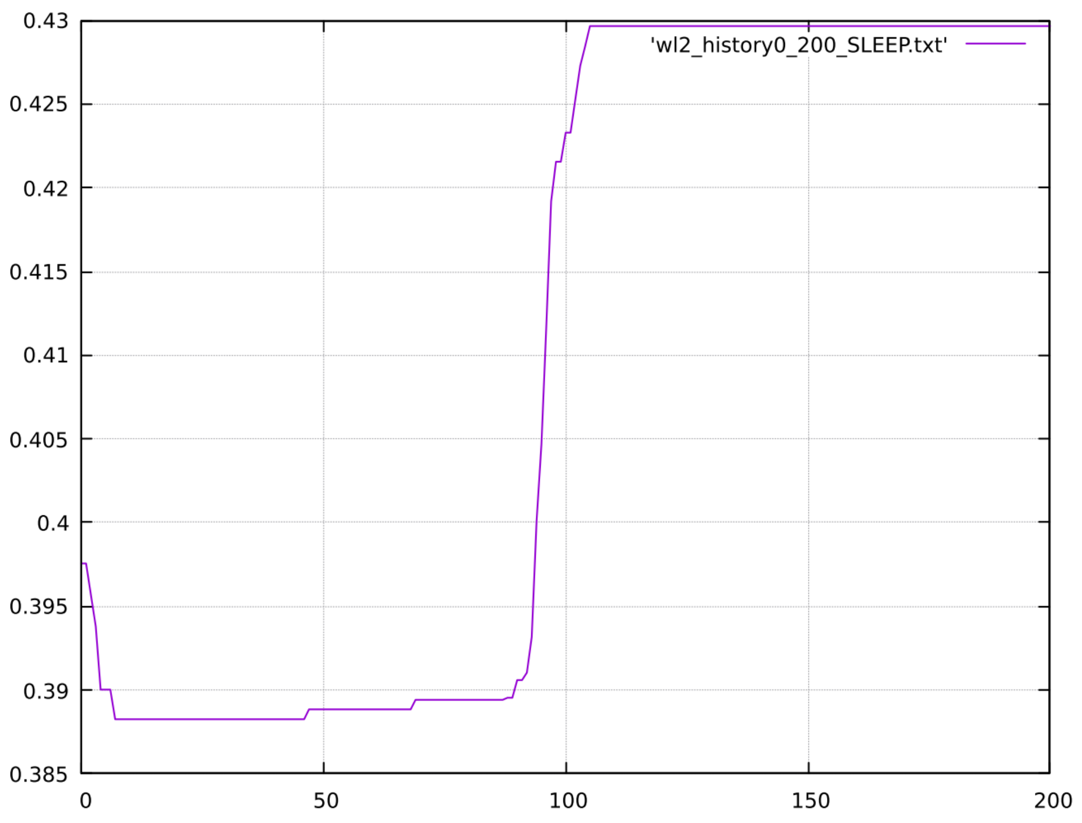
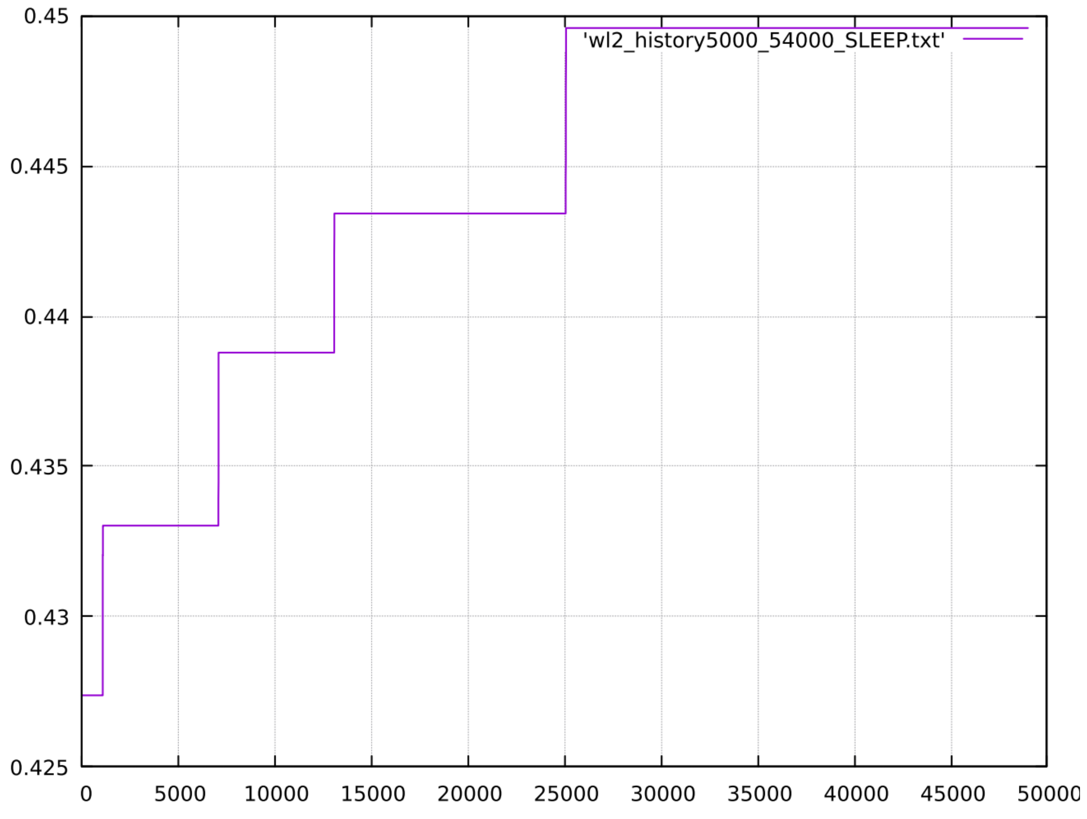


Figure18: workload2, energy (y) as a function of increasing threshold (x)





Additional comments

The main part of the code that has been used to perform all the analysis is contained in the *dpm-simulator* folder, especially *dpm_policies* has been edited to include all the policy implementation part and a c file, *src/dpm_sim_auto.c* has been added to automatically perform simulations on different timeout values and history thresholds.

The simulation launcher script, *dpm_simulator*, has been modified to allow the launch of the functions contained in the previously mentioned file.

Then, in order to make the analysis we used some script written in bash and matlab.

Bash scripts are in charge of executing the application and applying some filters ('grep', 'tr' and 'cut' unix commands) to the output of the psm (which can be disabled by commenting the directive #PRINT in the *src/dpm_policies.c*) with the aim to produce a file in csv format, which will be further analysed with Matlab.

- scripts/tester_delay.sh: execute the PSM with all timeout<Tbe for a given workload and produces a csv file with this format: *delay,timeout*
- script/tester_timeout.sh: execute the PSM with all timeout within a certain range for a given workload and produces a csv file with this format: *energy,timeout (the first line contains only the energy, the total energy without using any policy).*

Matlab scripts are in charge to make further analysis (like the plots in Figure14, Figure13, etc…) by reading the csv files produced by the bash scripts.

- scripts/timeout_energy.m: reads the two csv file produced by the two bash scripts and produce the graphs showed in the Figure14, Figure13.
- scripts/stem_idles.m: reads two workloads (like the workload1 and workload2, described in the file workloads/workload_1.txt and workloads/workload_2.txt) and plots the distribution of idle times that are less than a given threshold. Useful for producing graphs like the one in Figure9.
- scripts/plot_workload.m: reads a workload file, and plots it.
  Useful to produce a graph like those in Figure2 and Figure3.

It is also possible to produce csv files and "Matlab-like" graphs by using the src/dpm_sim_auto.c utilities, for instance some graphs showed in this report (like Figure11 and Figure10) are generated in this way.