Energy Efficient Scheduling in IoT Networks

Smruti R. Sarangi
Computer Science and Engineering
Indian Institute of Technology
New Delhi
srsarangi@cse.iitd.ac.in

Sakshi Goel
Computer Science and Engineering
Indian Institute of Technology
New Delhi
sakshigoel26693@gmail.com

Bhumika Singh
Computer Science and Engineering
Indian Institute of Technology
New Delhi
bhumikasingh1811@gmail.com

ABSTRACT

The Internet of Things (IoT) is poised to be one of the most disruptive technologies over the next decade. It is speculated, that we shall have billions of devices with communication capabilities very soon. Minimizing energy consumption is one of the most important problems in such IoT networks mainly because IoT nodes are distributed in the field with limited, unreliable, and intermittent sources of power. Even though the area of reducing power for stand-alone machines is very rich, there are very few references in the area of co-operative power minimization in a system with many IoT nodes. We propose two algorithms in this paper, which are at the two ends of the spectrum: *Local* exchanges information between neighboring nodes, and *Global* uses a global server that has recent snapshots of the global state of the network. We show that both these algorithms reduce energy consumption by roughly 40% for settings that use data from real life IoT deployments (data from Barcelona city). We further show that if deadlines are tight, *Local* is preferable for smaller networks, and *Global* is preferable for larger networks. When deadlines are loose, *Global* is preferable if we need to follow hard real time semantics, otherwise *Local* is preferable.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; *Sensor networks*; Sensors and actuators; • **Networks** → Network components;

KEYWORDS

Internet of things, Power efficiency, Smart scheduling, DVFS

ACM Reference Format:
Smruti R. Sarangi, Sakshi Goel, and Bhumika Singh. 2018. Energy Efficient Scheduling in IoT Networks. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167213

1 INTRODUCTION

We are experiencing a new era of transformation from a physical world to the digitized world of the Internet of Things (IoT). The IoT paradigm [20] refers to the interconnection of various day-to-day physical devices around us through the internet, thus enabling them to interact with each other and exchange meaningful information. IoT networks are predicted to exponentially grow in the next few years in terms of the quantity and complexity of devices. Some projections indicate that we shall have 50 to 100 billion IoT devices by the end of this decade.

There are several popular IoT reference architectures produced by Intel [9] and Microsoft [15]. The common denominator in these architectures is a 3-layer structure (see Figure 1). The lowest layer consists of IoT devices, the middle layer consists of gateway nodes and aggregators, and the highest layer consists of cloud based data centers. Each of these layers can consist of various sub-layers. For example, we can have a hierarchy of gateway nodes, where some nodes act as simple routers, and some other nodes perform sophisticated analytics. Even the highest layer can have many sub-layers: client facing servers, cloud nodes, and storage nodes.

Energy consumption in such complex IoT networks is regarded as a very important problem as noted in prior work([1, 8]). The reasons are as follows. Most IoT sensors and actuators are small and very power constrained. They typically run on batteries or use intermittent sources of power such as solar energy. Even IoT hubs and gateways do not have reliable power supplies and are often placed in the field, where power supply by itself is an issue. Moreover, given the fact that IoT nodes typically deal with a lot of data, we require a lot of energy to process all this data, and filter out the relevant portions for subsequent processing. Given these requirements, we believe that reducing the energy consumption in IoT networks is a worthy goal to pursue.

In general, the literature in reducing energy consumption is very extensive. This area has been an active topic of research for the last 15 years. However, the area of energy optimization for IoT networks where the nodes co-operatively reduce power is very sparse, and there is very little work in this area to the best of our knowledge. Most of the work in managing power has focused on a single node that may consist of several multicore processors. The methods mostly include a combination of dynamic voltage-frequency scaling(DVFS) [23], throttling techniques [2], and heuristics to transition to low power states [3]. These solutions are at the level of a single node. We use such solutions as a baseline. However, we observe that a far greater potential exists if IoT nodes co-operate among themselves to reduce power. We develop two protocols in this paper that decrease power by 40% on an average.

Specifically, we aim at solving the problem of optimizing energy consumption at IoT nodes, gateways and servers, while processing streams of soft real-time tasks. We are primarily interested in the energy consumed by processing devices in this paper. This can be anywhere from 1-99% of the total energy [11, 13] depending on the

SAC 2018, April 9–13, 2018, Pau, France
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167213

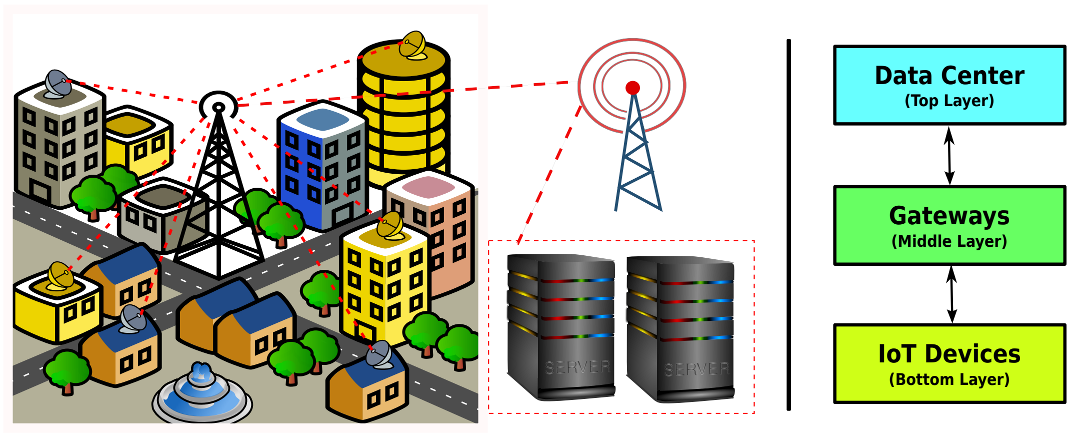


Figure 1: IoT architecture

type of the network. In general, there is a trend towards increasing compute energies as analytics moves to the edge. Each processing node regulates its operating voltage and frequency, based on an estimate of the slack time of the stream that it is processing. The novelty of our approach lies in the methods to exchange information between IoT nodes regarding power settings, deadlines, and slack times. We propose two methods in this paper: i) *Local*, where the information about slack times is piggybacked along with the tasks and is exchanged only within a small neighborhood, ii) *Global*, where a dedicated server (connected with high bandwidth links) maintains the state of all the nodes, and computes the configurations of each of the IoT nodes. We shall show that both the schemes are useful albeit in different situations.

The organization of the rest of the paper is as follows. In Section 2 we discuss background and related work, we then formulate our problem in Section 3, and present our techniques in Section 4; we present our evaluation results in Section 5, and finally conclude in Section 6.

2 BACKGROUND AND RELATED WORK

2.1 Reference IoT Architecture

The schematic of a reference architecture is shown in Figure 1. The architecture in consideration is similar to that used in [7, 10, 17]. Specifically, the three-layer IoT architecture considered consists of the following layers:

(1) Bottom layer: The bottom layer comprises of wireless sensor networks (WSNs) with an assortment of sensor/actuator nodes that can be deployed in a multitude of entities ranging from vehicles, smart homes, buildings, street lights, and traffic lights to devices such as wearables, smart phones, shoes, and smart cards.
(2) Middle layer: Gateway nodes that collect and buffer data read from sensor nodes, perform various computations (typically data analytics operations) and forward it across multiple hops to a data center.
(3) Top layer: This layer consists of servers typically located in data centers. These servers process the data given to them by gateways, store data in storage nodes, and direct the actuators to change the state of their immediate environment. The messages to the actuators again traverse the gateways and hubs (take the reverse path).

2.2 Related Work

In the following three sub-sections we focus on three aspects of power reduction in modern computing systems. We briefly summarize some of the related work in the areas of reducing power for multicore processors, sensor networks, and data centers. Even though highly optimized solutions for each of these areas exist, to the best of our knowledge, we have not seen comprehensive solutions for a plurality of nodes such as IoT networks.

2.2.1 *Reducing energy consumption in multicore processors.*

Dynamic energy is broadly proportional to a square of the operating voltage, and the operating voltage is approximately proportional to the operating frequency. The quintessential equations that are used in this area are: $E \propto V^2, V \propto f$, where V is the voltage, and f is the frequency. Hence, most of the techniques try to reduce the voltage and frequency in unison (known as dynamic voltage-frequency scaling (DVFS). In addition, leakage power is one more component of the total power dissipation. It is a super-linear function of the temperature, which is again mostly proportional to the dynamic power.

Now, DVFS per se is a very old technique, and hundreds of papers have been published in this area. The survey by Mokarippor et al. [16] elaborates on most of the important techniques. Most of the heuristics are focused on reducing the frequency while executing non-critical parts of the code, or when we have enough slack. In addition, we can have other techniques that decrease the activity factor by throttling the activity (refer to [2]). Higher activity leads to higher temperature, which leads to higher leakage (the net effect can be super-linear). We shall mainly use one of the popular DVFS heuristics as a baseline technique.

2.2.2 *Reducing Energy Consumption in Sensor Networks.*

Reducing energy consumption in sensor networks is a very rich area of research (refer to the survey by Anastasi et al. [1]). There are three major techniques: duty-cycling, data-driven approaches, and mobility. Duty-cycling means that the sensor nodes (or just their transceivers) are intelligently powered down. This ensures that we do not waste power by keeping the nodes on, when it is not required. The sensor nodes coordinate their wakeup and sleep times between themselves, or as per the requirement of the application [5]. Data-driven approaches aim to reduce the volume of the sampled data, such that redundant information does not get communicated. The sensing accuracy is set at an acceptable level in order to reduce the amount of sampled data. Moreover, nodes can also modulate their frequency and amplitude while communicating with other nodes [21]. There is a tradeoff between the communication data rate, bit error rate, and performance. The mobility approach involves moving motes that move towards the source of the entity that they need to sense. If done properly, this can reduce the communication energy, and reduce the number of hops that a message needs to traverse.

2.2.3 Reducing Energy Consumption in Data Centers. The traditional energy consumption reduction methods at data centers [8] include using the right topology, virtualization, energy aware routing, dynamic voltage and frequency scaling (DVFS), and dynamic power management (DPM). In addition, in recent proposals researchers are applying DVFS techniques to network elements, focussing on energy aware scheduling of network traffic and using energy efficient cooling strategies.

3 IMPLEMENTATION

3.1 Problem Statement

In this paper, we consider an architecture where each sensor/actuator is connected to a single gateway. The gateways are connected to each other using a tree based topology as shown in Figure 2. The gateway that is connected to the servers is designated as the root gateway(s). Streams of tasks are generated by sensors, and sent towards the central server (can be a set of cloud based servers or a data center). The data reaches the central server(s) through a hierarchy of gateways (or hubs). Each gateway can also do some processing of its own, and aggregate/filter the data. Finally, the servers process the tasks and determine the actions that need to be taken. This information is communicated to the actuators, again, using the same network of gateways (reverse path). We are assuming that each node (sensor, gateway, server) has processing units that either just store and forward packets, or intelligently process them. Our aim is to minimize the total energy without violating deadlines.

3.2 Task Model

We consider a stream of tasks (t_1, t_2, t_3, . . .) generated at sensors. A task t_i arriving at a node can be modelled as a 4-tuple (g_i,d_i,l_i,c_i), where g_i is the task generation time, d_i is the deadline, l_i is the amount of network traffic associated with a task (in bytes), and c_i is the number of execution cycles required to execute the task (worst case).

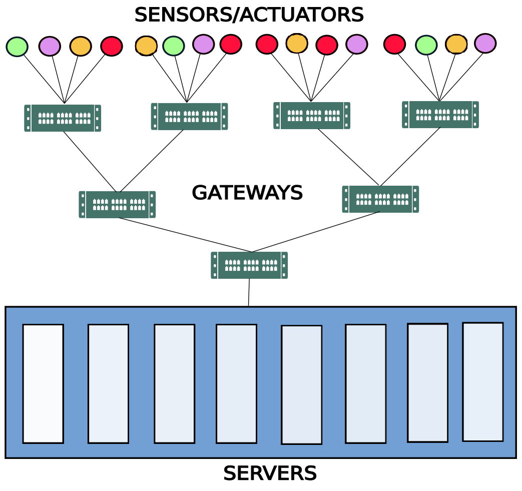


Figure 2: Architecture of an IoT system

3.3 Energy Consumption Model

We consider each processing node in the IoT network to be an m-core computing system. Each core supports r frequencies (f_1, f_2, ... , f_r), where $f_1 < f_2 < \ldots < f_r$. The energy consumption at a node is determined by the CPU and memory. This is a standard assumption for such systems [24]. The energy consumed by the CPU has two components: dynamic energy and static energy, which is also known as leakage energy [25]. As the dynamic component is the prime contributor of the energy consumption in our class of systems and temperature fluctuations are limited, our energy consumption model focusses mainly on the dynamic energy consumption (similar to [26]). We assume the static energy to be a constant [26].

Now, let n_j be the j^{th} node on task t_i's path in the network. The energy consumption, e_{ij}, at node n_j is given by:

$$e_{ij} = \kappa_j \times n_{ij} \times f_j^2 \tag{1}$$

where κ_j is a constant of proportionality (remains constant for a given node), n_{ij} is the number of cycles the task takes to execute at node j, and f_j is the execution frequency. The total energy is given by:

$$\begin{aligned} E_{total} &= \sum_{i=1}^{n} \sum_{j=1}^{l} e_{ij} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{l} \kappa_j \times n_{ij} \times f_j^2 \end{aligned} \tag{2}$$

where n is the total number of tasks that have been generated, and l is the number of nodes in the execution path of the task.

4 ENERGY EFFICIENT SCHEDULING ALGORITHMS

In this section, we discuss in detail the two algorithms that we have proposed for energy efficient scheduling of tasks in an IoT network, while trying to ensure that we violate as few deadlines as possible. In both the algorithms each processing node uses t_{est}, the estimated time that the task will take to reach the actuator once it leaves the current node, to calculate t_{rem}. This is the maximum available time the node has to execute the current task, and this is done by taking the deadline into consideration. Subsequently, the multicore node performs DVFS based scaling (up-scaling or down-scaling the frequency of some core).

Algorithm 1 Algorithm for applying DVFS using t_{est}

```
1:  procedure EXECUTETASK($task, t_{est}$)
2:      $t_{current} \leftarrow$ getCurrentTime();
3:      $t_{rem} \leftarrow (task.finishTime$ - $t_{current} - t_{est})$;
4:      if $t_{rem} \leq 0$ then
5:          $frequency \leftarrow$ max operating frequency;
6:      end if
7:      if there is an idle core then
8:          run on idle core with frequency $\frac{c}{t_{rem}}$;
9:      else
10:         run on a core having minimum frequency satisfying $w_i + \frac{c}{f_i} \leq t_{rem}$ ;
11:         if no such core found then
12:             increase frequency of core having maximum frequency to max operating frequency;
13:         end if
14:     end if
15: end procedure
```

Let us now describe the details (refer to Figure 1). Every node tracks the average waiting time at its constituent cores and frequencies at which different cores are running. Whenever a task needs to be scheduled, the availability of an idle core is checked first. If an idle core is available, then, the task is scheduled on it and the core's frequency is set to the minimum available frequency that is greater than $\frac{c}{t_{rem}}$.

However, if no core is idle, an optimal frequency is calculated to run the task such that the least possible amount of energy is consumed, while remaining within the deadline. Since we consider a discrete set of frequencies, we consider the smallest such frequency (f) that is greater than or equal to the optimal frequency. Then, we consider the frequencies at which all the cores are running. We narrow down our search to those cores that have frequencies greater than or equal to f, and find the core with the least frequency in this set. We then send the task to the EDF (earliest deadline first, priority queue) of this core. Then based on its deadline, the task executes.

Now, if we cannot find such a core, then we consider the core with the largest frequency f_{max} (note that $f_{max} < f$). We enter the task in its EDF queue. Before the task begins to execute we upscale the frequency of that core to f.

If we naively continue this process, then after some time all the cores will start executing at the maximum frequency. However, to stop this, we have two mechanisms. If a core becomes idle we power it down. After this, its execution frequency is determined by the task that is assigned to it. For a core that has been in continuous operation for C cycles, we enter it into set $\mathcal{S}$. We choose a core at random from $\mathcal{S}$ (once every $50\mu s$) and downscale its frequency to that required by the currently executing task. The assumption is that it takes $20\mu s$ [6] to change to a new DVFS setting. We choose C as 10 times the worst case execution cycles of the task. In our method, the most important parameter for performing DVFS is the information about the estimated time, t_{est}. We propose two methods by which this information can be made available to a node.

4.1 Global Algorithm

In the *Global* algorithm (see Algorithms 2 and 3), we maintain a central high bandwidth server, *CS*, that is accessible to all the nodes, such that all the nodes can communicate with the CS and vice-versa. This algorithm is more useful in a system having a lot of gateways, which have some form of direct internet connectivity. We assume that all the communication to the CS has the highest priority. Similar settings have been used in [14].

Each node locally calculates t_{avg}, the average execution time taken by a task to execute on that node by computing the sum of $(t_d - t_a)$, where t_a is the arrival time of the task at the node and t_d is the departure time of the task from the node, over all tasks on the node and dividing it by the number of tasks. This average execution time includes the time the task waits in the input queues of the node and the time the task takes to execute at some core of the node. This information about the average execution time is sent to the CS periodically by each node (once every 100 micro-seconds).

The CS stores this information for each (parent,child) pair in a table. This table basically stores how long a task will take to execute at the parent node, if it comes from a given child node. This is because different children might be sending different types of tasks to the parent. This is the value of t_{avg} that the CS will use to calculate t_{est} for any node. Since we consider steady streams of tasks, we expect t_{est} to be a stable value across small durations of time.

Note that alternative renditions of this idea are possible. Each node can also send a mean and variance (assuming a normal distribution). In this case, the CS needs to store the mean and variance of t_{avg}. Since the sum of normally distributed random variables is also a normally distributed random variable, t_{est} will also be normally distributed. We can then provide a value that is three standard deviations above the mean to the node.

In another rendition of this idea, it is possible that the same group of sensors might generate different types of tasks. Here, we can label each type with a number. CS will now maintain the t_{avg} information for the 3-tuple (parent,child,task_type).

4.2 Local Algorithm

In the *Local* algorithm (Algorithm 4), each node updates its t_{est} value using the information piggybacked with the task that it receives from its adjacent nodes. Each node calculates the average time, t_{avg}, that a task spends at the node by averaging the time interval (t_d - t_a), where t_a is the arrival time of the task at the node and t_d is the departure time of the task from the node, for all the tasks getting executed at the node. This time interval includes the waiting time of the task at the input queue of the node and the EDF queue of the core it gets assigned to, as well as the execution time of the task at the assigned core. Further, each node maintains information about estimated time (t_{est}) for each of its neighboring

Algorithm 2 t_{est} calculations at a node with the *Global* Algorithm

```
1:  procedure GlobalAlgoAtNode
2:      receive($request$);
3:      $task, source \leftarrow request.task, source$
4:      Update $t_{avg}$ corresponding to tasks received from source node
5:      $t_{current} \leftarrow$ getCurrentTime();
6:      if $t_{current}$ % $update\_t_{est}\_interval = 0$ then
7:          $req.reqCode = get\_t_{est}\_value$;
8:          $req.requestingNode \leftarrow self$;
9:          send($req$,$CS$);
10:         Wait for a response from $CS$;
11:         receive($response$);
12:         $t_{est} \leftarrow response.message$;
13:     end if
14:     if $t_{current}$ % $send\_t_{avg}\_interval = 0$ then
15:         $req.reqCode = send\_t_{avg}\_value$;
16:         $req.message \leftarrow t_{avg}$ for all child nodes
17:         send($req$, $CS$)
18:     end if
19:     if $execute\_task()$ then
20:         executeTask(task,$t_{est}$);
21:     end if
22: end procedure
```

Algorithm 3 t_{est} calculations at the Central Server with the *Global* Algorithm

```
1:  procedure GlobalAlgoAtCS
2:      receive($request$);
3:      $n \leftarrow request.requestingNode$;
4:      $t_{avg} \leftarrow request.message$;
5:      if $request.reqCode = get\_t_{est}\_value$ then
6:          $curnode, prevnode \leftarrow n$
7:          while actuator node is not reached do
8:              $curnode \leftarrow n.nextNode$
9:              add $t_{avg}$ for tuple ($curnode, prevnode$) to $t_{send}$
10:             add propagation delay for link between $curnode$ and $prevnode$
11:             $prevnode \leftarrow curnode$
12:         end while
13:         $res.message = t_{send}$;
14:         send($res$, $n$);
15:     end if
16:     if $request.reqCode = send\_t_{avg}\_value$ then
17:         Update the $t_{avg}$ for all the child nodes of node $n$
18:     end if
19: end procedure
```

nodes. The estimated time for a node N signifies the amount of time a task will take, once it leaves the current node, to reach the actuator if the next node where the task is headed is N. When a task is to be scheduled to a core, the t_{est} information that is used will be that corresponding to the next node where the task is headed. Before leaving a node, a message is piggybacked with the task. The message comprises of the value $t_{avg} + t_{est}$ where t_{est} is the value of estimated time corresponding to the node from where the task came.

Algorithm 4 t_{est} calculations in the *Local* Algorithm

```
1:  procedure Local
2:      receive($request$);
3:      $task \leftarrow request.task$;
4:      $(info_{recv}, source) \leftarrow (request.message,\ request.source)$;
5:      $t_{est}$ for source $\leftarrow$ ($t_{est}$ for source + $info_{recv}$ ) / 2;
6:      $info_{send} \leftarrow t_{avg} + t_{est}$ for source;
7:      $t_{est}$ for self $\leftarrow t_{est}$ for next node
8:      executeTask($task$, $t_{est}$);
9:      ($req.task$, $req.message$) = ($task$, $info_{send}$);
10:     send($req$, $self.nextNode$);
11: end procedure
```

5 EVALUATION

In this section, we first describe our simulation setup, and then proceed to describe our results in detail.

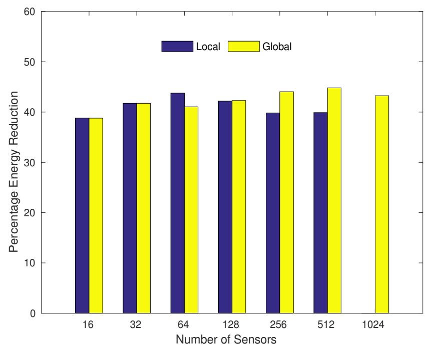
5.1 Simulation Setup

We developed an IoT simulator in Java to evaluate the potential energy savings in a real-time IoT network. The sensor network part of the simulator has been validated against the popular network simulator, NS3 [19] (error limited to 0.1%). The data center part of this simulator has been validated against the popular cloud simulator, CloudSim [4] (error limited to 2%). To the best of our knowledge there is no publicly available full scale IoT simulator that we could use to validate our entire simulation setup; hence, we had to validate it in parts.

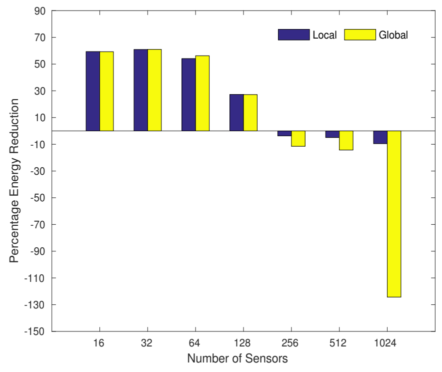
We assume that sensors generate data, the data flows up to the servers via a tree of gateways, the servers compute an appropriate response and this flows back to actuators via the gateways. A *task* comprises of all the actions that are taken by different nodes in the network to process a set of messages generated by a sensor (or a group of co-located sensors). Our IoT network processes many such streams of tasks simultaneously.

The simulator takes as input a task set, specified with the deadlines and computation requirements for each task. Other parameters provided to the simulator include the number of sensors/actuators, number of servers, frequency of sensing, a list of available operating frequencies, and the number of cores for each type of node. The number of execution cycles for a task at a node is a fraction of its worst case execution cycles [18]. This fraction is a uniformly-distributed random number. The output of this simulator contains detailed statistics about the execution of the tasks such as the time it took to execute them on each node, the energy consumed, and the number of deadlines missed.

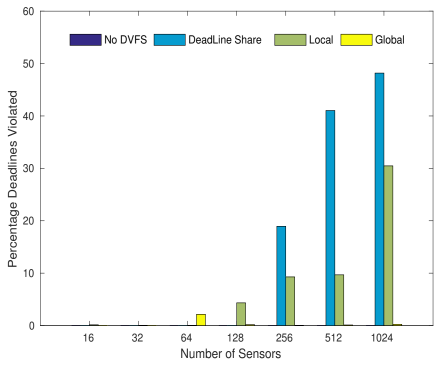
In the simulations, the frequency of sensing, rate of task generation, and the sizes of the tasks (network usage, and duration) are as per the real time data collected by Sinaeepourfard et al. [22] for the Barcelona city. As per this data, task deadlines are 10-100% more than the maximum execution time of the task (through the entire network), which is calculated by considering the minimum



(a) Energy consumption minimization (baseline: No-DVFS)



(b) Energy consumption minimization (baseline: DeadLine-Share)



(c) Task drop rates

Figure 3: Performance with tight deadlines

operating frequency of the cores, the worst case execution schedules at all the nodes, and the worst case propagation time over the network. This methodology is also in line with that described by Kim et al. [12].

On the line of the references, we set the deadlines of our tasks as follows (refer to Table 1). For different settings with different numbers of sensors, we consider two configurations: tasks with tight deadlines and loose deadlines. Table 1 shows the difference between the deadline and the worst case execution time of the task as a fraction of the worst case execution time (values in %). For example, for a system with 256 sensors, we have a minimum slack of 22-26% for systems with tight deadlines, and a minimum slack of 69-76% for systems with loose deadlines. Values are chosen uniformly within the ranges.

Table 1: Deadlines in our System

Number of Sensors	Tight Deadlines	Loose Deadlines
16	3-4%	9-12%
32	4-6%	14-18%
64	7-10%	23-28%
128	13-15%	40-45%
256	22-26%	69-76%
512	40-44%	80-90%
1024	72-78%	95-100%

Propagation delays across the network are generated randomly in the range of 1-10 milliseconds. For gateways, the range of operating frequencies is 0.5 GHz-1 GHz whereas, for servers, the range of operating frequencies is 1.5 GHz-2 GHz with a DVFS step size of 100 MHz (for both). The results reported are for simulations carried out for a duration of 12 hours.

Since till date, there is no state of the art method that caters to our said problem, we have compared our algorithms with the following two methods to demonstrate the effectiveness of our algorithms in minimizing energy consumption: i) No-DVFS: Each node runs without using DVFS by simply running the task using the average operating frequency of its available frequency set. ii) DeadLineShare-DVFS: The deadline of the task is divided equally among all the processing nodes and each node then scales the frequency by applying DVFS according to the allotted time to execute the task. In both the methods, the nodes are oblivious to the state of the IoT network.

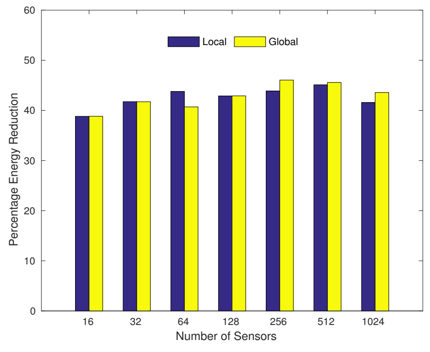
All our experiments were performed on a 64-bit Ubuntu Linux system (Version 14.04), with an Intel Core i7-3770s CPU running at 3.10 GHz with 4 GB RAM.

5.2 Results

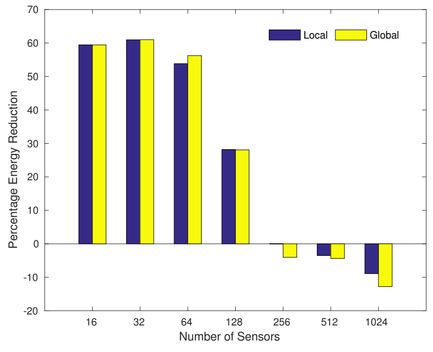
5.2.1 Comparison of Proposed Algorithms with Tight Deadlines. Figure 3a shows the results for experiments with tight deadlines, as compared to No-DVFS. The y-axis shows the mean energy minimized. It is between 40-45% vis-a-vis No-DVFS. This is because No-DVFS runs tasks at a much higher frequency than what is required. Now, for smaller networks, the energy consumption reduction by *Local* is more than *Global* because maintaining the CS and sending messages to it are additional overheads. Whereas, for larger networks, *Global* works better because of the global view that it creates.

Now, let us take a look at the percentage of tasks dropped (deadlines violated) in Figure 3c. It is negligible in the case of No-DVFS because it runs tasks at a higher frequency than what is required. The task drop rate is greater in the case of *Local* for larger networks as compared to *Global*. This is because *Local* gets its t_{est} information from its immediate neighborhood. If their is any congestion downstream, the algorithm is not very responsive. Whereas, for *Global*, having a global view helps us eliminate deadline violations almost completely.

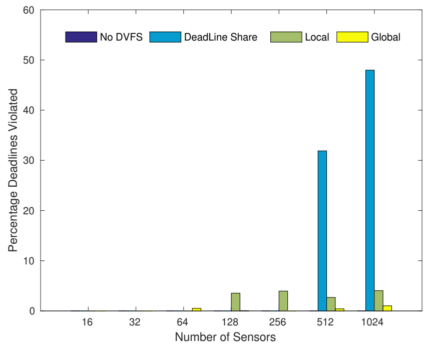
Let us now compare with DeadLine-Share in Figure 3b. For smaller networks, *Local* and *Global* have low task drop rates (0-1)% and their energy consumption is minimized by 20-60% vis-a-vis Deadline-Share. For larger networks (> 256 sensors), DeadLine-Share is more power efficient (20-100%). However, the concomitant



(a) **Energy consumption minimization (baseline: No-DVFS)**



(b) **Energy consumption minimization (baseline: DeadLine-Share)**



(c) **Task drop rates**

Figure 4: Performance with loose deadlines

task drop rate is more than 20%. This is prohibitive. For such networks, *Global* is the best choice. The reason for the high task drop rate for larger networks is because Deadline-Share is trying to run the tasks at a lower frequency than what is required. Since the deadline is distributed amongst all the nodes equally, the nodes remain unaware of any bottlenecks in the network and continue running the tasks at their predicted frequencies unbeknownst to any congestion ahead.

5.2.2 *Comparison of Proposed Algorithms with Loose Deadlines.* In the case of loose deadlines, the energy consumption by the proposed algorithms is around 40% lower than the corresponding values for No-DVFS (for both small and large networks). The results are shown in Figure 4a. However, the trade-off, in the case of *Local* is that a small percentage (3-5%) of tasks get dropped (deadlines violated) in larger networks, as can be seen in Figure 4c. *Global* however has negligible task drop rates for even larger networks. The energy consumption by both *Local* and *Global* is almost the same. If we compare the energy consumption of the proposed algorithms against DeadLine-Share (see Figure 4b), we observe that for smaller networks, the energy consumed by both of them is far lower (30-60%). The task drop rate is in the range of 0-4% for *Local* (same reasoning as that for tight deadlines).

5.3 Sensitivity

In this section, we vary the sensors' frequency of sensing in order to evaluate the performance of the proposed algorithms in the case of medium and heavy traffic (loose deadlines). Figure 5 shows the task drop rate for a mean sensing duration of 1 sec, and 10-60 sec in increments of 10 seconds. From the figure, we can infer that *Global* always performs better (in line with Figures 3c and 4c). *Local* is relatively better when the frequency of sensing is high. This is because nodes get updated very frequently about the state of the network. In terms of energy consumption *Local* is marginally better (by 3-4%). Further, the energy consumption minimization done by

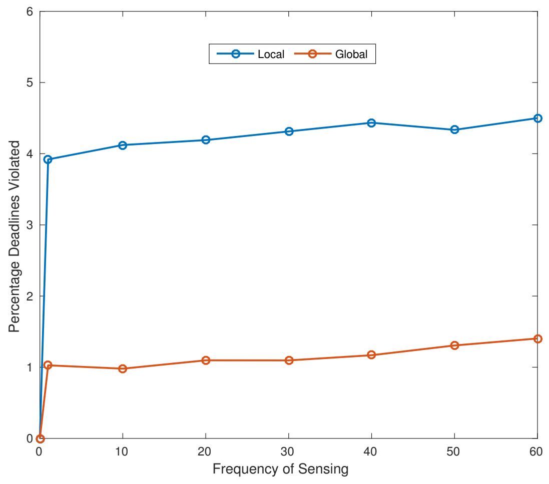


Figure 5: Task drop rate v/s frequency of sensing

both *Local* and *Global* against No-DVFS, for a mean sensing duration of 1 sec, and 10-60 sec in increments of 10 seconds, is shown in Figure 6. It is observed that both the proposed algorithms perform significant energy consumption minimization even in scenarios of heavy traffic. The performance of the algorithms in terms of energy consumption reduction is almost constant with respect to the frequency of sensing.

6 CONCLUSION

In this paper, we proposed two methods: global and local, for network aware energy efficient scheduling of tasks with deadline constraints in an IoT network. Both methods involve exchanging of information between nodes in order to give an estimate of the

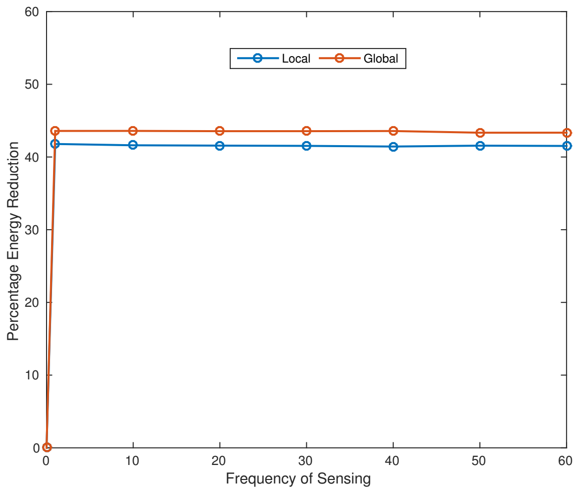


Figure 6: Energy consumption minimization (baseline: No-DVFS) v/s frequency of sensing

remaining time required for task execution and to perform voltage-frequency scaling. The global method uses a dedicated central server that computes the best configuration for the entire network. The local method gets this information from piggybacked data that comes along with regular messages from its neighbors. Simulation results show that both the proposed schemes achieve a significant energy consumption reduction (around 40%) with respect to simple DVFS based techniques, with little or no degradation in performance, or in terms of deadlines missed. For tasks with tight deadlines, the local algorithm works better in the case of small networks as it avoids the overhead of sending messages to a separate server, while the global algorithm works better in the case of large networks. For tasks with loose deadlines, the local algorithm can be preferred over global if the penalty in terms of 3-4% task drops is acceptable, otherwise the global algorithm can be used if task drops need to be avoided completely.

REFERENCES

[1] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. 2009. Energy conservation in wireless sensor networks: A survey. *Ad hoc networks* 7, 3 (2009), 537–568.

[2] Amirali Baniasadi and Andreas Moshovos. 2001. Instruction flow-based front-end throttling for power-aware high-performance processors. In *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM, 16–21.

[3] Luca Benini and Giovanni De Micheli. 1995. State assignment for low power dissipation. *IEEE Journal of Solid-State Circuits* 30, 3 (1995), 258–268.

[4] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience* 41, 1 (2011), 23–50.

[5] Christian C Enz, Amre El-Hoiydi, J-D Decotignie, and Vincent Peiris. 2004. WiseNET: an ultralow-power wireless sensor network solution. *Computer* 37, 8 (2004), 62–70.

[6] Andreas Genser, Christian Bachmann, Christian Steger, Reinhold Weiss, and Josef Haid. 2010. Power emulation based DVFS efficiency investigations for embedded systems. In *System on Chip (SoC), 2010 International Symposium on*. IEEE, 173–178.

[7] Jayavardhana Gubbi, Slaven Marusic, Aravinda S Rao, Yee Wei Law, and Marimuthu Palaniswami. 2013. A pilot study of urban noise monitoring architecture using wireless sensor networks. In *ICACCI*.

[8] Ali Hammadi and Lotfi Mhamdi. 2014. A survey on architectures and energy efficiency in data center networks. *Computer Communications* 40 (2014), 1–21.

[9] Intel. 2015. Intel IoT Platform Reference Architecture. https://www.intel.in/content/www/in/en/internet-of-things/white-papers/iot-platform-reference-architecture-paper.html. (2015). Accessed on 8^{th} December, 2017.

[10] Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. 2014. An information framework for creating a smart city through internet of things. *IEEE Internet of Things Journal* 1, 2 (2014), 112–121.

[11] Zeeshan Ali Khan and Mustafa Shakir. 2012. Interplay of communication and computation energy consumption for low power sensor network design. *International Journal of Ad Hoc, Sensor & Ubiquitous Computing* 3, 4 (2012), 65.

[12] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. 2007. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters.. In *CCGrid*, Vol. 7.

[13] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. 2010. DENS: data center energy-efficient network-aware scheduling. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. IEEE, 69–75.

[14] Nikolai KN Leung and Raymond T Hsu. 2005. Method and apparatus for out-of-band transmission of broadcast service option in a wireless communication system. (June 21 2005). US Patent 6,909,702.

[15] Microsoft. 2016. Microsoft Azure IoT Reference Architecture. http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft_Azure_IoT_Reference_Architecture.pdf. (2016). Accessed on 8^{th} December, 2017.

[16] P Mokarippor and Mirsaeid Hosseini Shirvani. 2016. A State Of The Art Survey On DVFS Techniques In Cloud Computing Environment. *Journal of Multidisciplinary Engineering Science and Technology (JMEST)* 3 (2016), 50.

[17] Congduc Pham and Philippe Cousin. 2013. Streaming the sound of smart cities: experimentations on the SmartSantander test-bed. In *GreenCom*.

[18] Padmanabhan Pillai and Kang G Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, Vol. 35. 89–102.

[19] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*.

[20] Pallavi Sethi and Smruti R Sarangi. 2017. Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering* 2017 (2017).

[21] Li Shang, Li-Shiuan Peh, and Niraj K Jha. 2003. Dynamic voltage scaling with links for power optimization of interconnection networks. In *High Performance Computer Architecture (HPCA)*.

[22] Amir Sinaeepourfard, Jordi Garcia, Xavier Masip-Bruin, Eva Marín-Tordera, Jordi Cirera, Glòria Grau, and Francesc Casaus. 2016. Estimating Smart City sensors data generation. In *Ad Hoc Networking Workshop (Med-Hoc-Net)*.

[23] Ali Taha, Sani Gosali, Zhijun Gong, Nelson Sollenberger, and John Wright. 2008. Method and System for Dynamic Voltage and Frequency Scaling (DVFS). (2008). US Patent App. 12/190,029.

[24] Jinhai Wang, Chuanhe Huang, Kai He, Xiaomao Wang, Xi Chen, and Kuangyu Qin. 2013. An energy-aware resource allocation heuristics for VM scheduling in cloud. In *HPCC_EUC*. 587–594.

[25] Le Yan, Jiong Luo, and Niraj K Jha. 2005. Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 7 (2005), 1030–1041.

[26] Xiaomin Zhu, Laurence T Yang, Huangke Chen, Ji Wang, Shu Yin, and Xiaocheng Liu. 2014. Real-time tasks oriented energy-aware scheduling in virtualized clouds. *IEEE Transactions on Cloud Computing* 2, 2 (2014).