WattEdge: A Holistic Approach for Empirical Energy Measurements in Edge Computing

Mohammad S. Aslanpour[1,2], Adel N. Toosi[1], Raj Gaire[2], and Muhammad Aamir Cheema[1]

[1] Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Clayton, Australia,
[2] CSIRO's DATA61, Canberra, Australia

Abstract. Of the main challenges to keep the edge computing dream alive is to efficiently manage the energy consumption of highly resource-limited nodes. Past studies have limited or often simplistic focus on energy consumption factors considering computation or communication-only solutions, questioned by either costly hardware instrumentation or inaccurate software-specific limitations. With this gap in mind and the wide adoption of single-board computers (SBCs) such as Raspberry Pis in edge, in this paper, we propose a novel holistic and accurate energy measurement approach in edge computing. Exploring a *Test and Learn* strategy, (1) we firstly perform a comprehensive analysis of identifying factors affecting energy consumption of edge nodes; (2) we develop and utilize WattEdge, a standard framework to evaluate the identified factors; (3) we conduct extensive empirical experiments on Raspberry Pis to thoroughly and uniformly assess the significance of each factor, thereby proposing an all-inclusive energy model. Wattedge is able to measure energy consumption factors such as CPU, memory, storage, a combination of them, connectivity, bandwidth usage, and communication protocols, as well as energy sources such as batteries. The results specifically warn us of the necessity of considering previously underestimated factors such as connectivity. A Smart Agriculture use case is implemented to validate the performance of the energy model, demonstrating a 95% accuracy.

Keywords: Edge Computing· Energy Consumption· Measurement· Raspberry Pi· Internet of Things (IoT)· Performance Evaluation

1 Introduction

With the ever-increasing growth of the Internet of Things (IoT), Cisco believes that "the number of connected devices will exceed three times the global population by 2023 [1]." Edge (or Fog) Computing can bring the compute, storage and network resources closer to IoT devices to address low latency requirements of IoT applications [6]. Low power and small sized devices are intended to bring those capabilities at the edge. Recently, Single-Board Computers (SBCs) have attracted special attention and are entitled to realize the presumed edge nodes [23]. SBCs such as Raspberry Pis (Pis) or Odroids are highly power-constraint [13].

The problem of efficient energy utilization for ultra low power edge nodes appears urgent [21, 5, 25, 22]. On the demand side, this is urgent because while edge nodes struggle with their energy management for running heavy tasks (e.g., AI tasks) [17, 16, 14], they are also expected to share their resources with peers [7, 20], known as task offloading. On the supply side also, the challenge of harvesting energy from the environment (solar, wind, or thermoelectric) became a challenging issue. This, however, does not exclude line-powered edge platforms from pressing environmental and economical side effects of high energy usage.

Given the exponential growth of IoT, a myriad of connected devices in industry, including agriculture, automobile, telecommunication, etc., will co-exist in the near future which will increase energy consumption and the demand for power supply. Such concerns warn the importance of intelligence about the energy consumption of IoT and its underlying platforms, so that optimization actions become feasible. Basically, this intelligence cannot be achieved without the knowledge of the major energy consumers and their impact on these platforms [4].

The key questions to optimise the energy consumption on the edge devices is *what are the factors contributing to energy consumption*? *How significant each factor could be*? And more essentially, *how to develop a practical holistic approach for accurate estimation and measurements of these factors*? With the current state of the art literature [15, 23, 24, 8], however, answering these questions appears difficult since each work only measures an in-comprehensive list of factors. Moreover, accumulating partial measurements from different studies such as [3, 9, 12] that employ dissimilar system under tests, cannot guarantee a reliable outcome. More critically, they either perform software-based measurements that present a restricted coverage to specific applications, or perform hardware-based measurements that require costly hardware instrumentation [17]. Rigorous countermeasures are required to first identify potential factors. Also, the significance of each factor needs to be assessed under a similar setting and for a reasonable duration so that accurate and reliable energy models can be built [13, 21, 5, 25].

Motivated by this gap in knowledge, we believe that a systematic and thorough study is required to identify energy consumption factors and the degree at which these factors affect energy consumption. To achieve this, the following key contributions are made:

- Identifying potential factors impacting energy consumption of edge nodes;
- Proposing *WattEdge*, a standard framework for measuring energy consumption of SBCs; *WattEdge* does not require costly hardware instrumentation such as sensors, shunt, analog-to-digital converter, etc.
- Empirically evaluating various edge-related energy consumption factors using *WattEdge*, supplemented by an all-inclusive energy model; and
- Validating the model's performance in Smart Agriculture domain using a practical application and demonstrating a 95% accuracy.

The remainder of this paper is structured as follows. Section 2 discusses our proposed holistic approach to identify major energy consumption factors for edge nodes. Then, we propose "*WattEdge*", a standard framework for our empirical experiments in Section 3. In Section 4, the factors are empirically studied under the same settings to provide the basis for (a) accurate comparison, (b) acquiring the significance thereof, and (c) an energy model. In Section 5, we validate the energy model generated by *WattEdge* using a practical application in Smart Agriculture domain on a cluster of Raspberry Pis Finally, we discuss the key findings of this research in Section 6 and conclude in Section 7.

2 Related Work and Energy Consumption Factors

Identifying potential factors impacting energy consumption of edge computing devices, the literature on measuring and modelling the energy consumption factors of edge computing is thoroughly reviewed. Our study uncovers nine factors impacting the energy consumption of the edge nodes. A summary of the degree at which those factors are considered in related work is provided in Fig. 1 (numbers in Fig.1 correspond to references).

To begin with, a baseline for measurements is considered as an essential factor. Hence, measuring energy consumption when the device is in idle state ① seems unavoidable as perceived by several studies [26, 23, 24, 11, 9]. Edge nodes host the IoT applications, demanding them to utilize computational resources. Among them, of course, CPU ② is fairly dominant [15], but other resources such as memory ③ and storage ④ are also worth considering for two reasons: firstly, edge nodes are highly power-constrained and hence sensitive to minor factors; secondly, certain IoT applications (e.g. AI applications) heavily rely on such resources. To the best of our knowledge, only one work considers the memory [21], and none paid attention to the storage in edge. Hence, we also included cloud-specific efforts [11, 26] in Fig. 1. Despite individual CPU, memory and storage, the energy usage due to a bundle of resources ⑤ is a matter of concern as well which is neglected to a large extent. However, calculating energy consumption for individual resources and then accumulating them may not give an accurate estimation, as their combination also alters the energy usage.

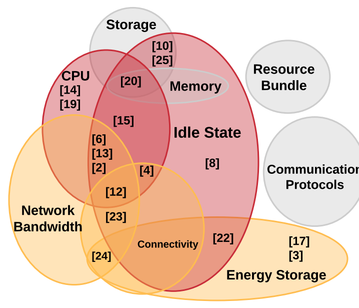

Fig. 1: Related work on energy measurement and modelling: red=major, yellow=moderate and gray=minor effort.

Connectivity is critical for edge nodes [19], where short-range connectivity means ⑥ such as WiFi, Bluetooth, ZigBee, USB, HDMI, VNC, etc. play essential role in Edge-IoT domain. Therefore, their effects on energy consumption have to be considered. Notably, WiFi as a widely-used means of connectivity has already gained a lot of attention [13, 24, 25, 5]. By connectivity, the communication comes into the play. The degree at which communication influences energy consumption may vary which makes it worthy of consideration [2]. Industrial IoT (IIoT) applications tend to send and receive continuous messages containing single-data points such as temperature and humidity, for instance, while images sent by Traffic Control Cameras in a Smart City application utilize much more network bandwidth [6]. Considering this, the network bandwidth utilization ⑦ and its impact on energy usage under different levels of data transmission must be investigated [13, 24, 25]. Communications rely on underlying protocols ⑧ that function in either a request/reply (e.g., HTTP or CoAP) or a publish/subscribe (e.g., MQTT or DDS) fashion, whose impact is missed in the related work.

On the supply side, however, it is essential to understand the energy supply limitations. Edge nodes have to rely on only limited capacity batteries or renewable energy sources harvested locally, other than or along with the grid. Given that, the estimation of energy consumption would be practical only when the behaviours of energy source and storage ⑨ such as battery are well understood.

Our proposed approach differs from the existing solutions as specified in the following. Firstly, the works mentioned above tend to consider a selective list of factors from only one [2, 7] to four [11, 18]. In contrast, our approach holistically covers the nine identified factors to provide a fine-grained measurement. Secondly, the accuracy in [26, 5, 25, 7] is doubted by not validating the proposal using real use cases or by merely relying on simulations; hence, in addition to extensive empirical studies, we validate our proposal in a realistic scenario. Thirdly, to achieve reasonable accuracy, cutting-edge hardware instrumentation used by [11, 25, 2, 3, 7, 14, 21] would not always be feasible due to its complexity and cost. We try to avoid this by encouraging a lightweight and low-cost Test and Learn strategy. Moreover, the lightness of the proposed framework for resource-limited edge nodes appears critical, which is compromised in [21, 16, 20, 23, 12]. Finally, our proposed approach is accompanied with a framework for reproducibility and extensibility, similar to [9] which provides an open-source framework, while [2, 20, 18, 12, 25, 22, 7, 21] lack such features.

3 WattEdge: The Evaluation Framework

The *WattEdge* framework, open-sourced on GitHub[3], is designed and implemented on a real SBC-based testbed to measure the significance of all identified factors (see Fig. 2). In brief (see Fig. 2), ① an SBC edge node is prepared. ② Simultaneously, a Stress Worker and System Monitor Agent on the main edge node and ③ a Power Monitor Agent on the secondary Pi is invoked. ④ The Stress Worker invokes the Stress Function ⑤ which triggers stress tools. ⑥ Supplementary services and scripts are executed on the edge nodes. Finally, ⑦ the Logger function collects and reports the monitored data.

Edge device: Emerging in 2012, Raspberry Pis have gained the momentum in the race of IoT devices [5]. They are recently employed as a perfect option for adopting edge computing, whether as standalone or even clusters of edge nodes [23]. In Pi family, we find Raspberry Pi 3 Model B+ to be one of the most utilized ones [7, 23, 3, 18]. This Pi features a 1.4Ghz Quad-Core Processor, a 1GB LPDDR2 SDRAM, a 40-pin GPIO header, 5v USB power adaptor

[3] https://github.com/aslanpour/wattedge

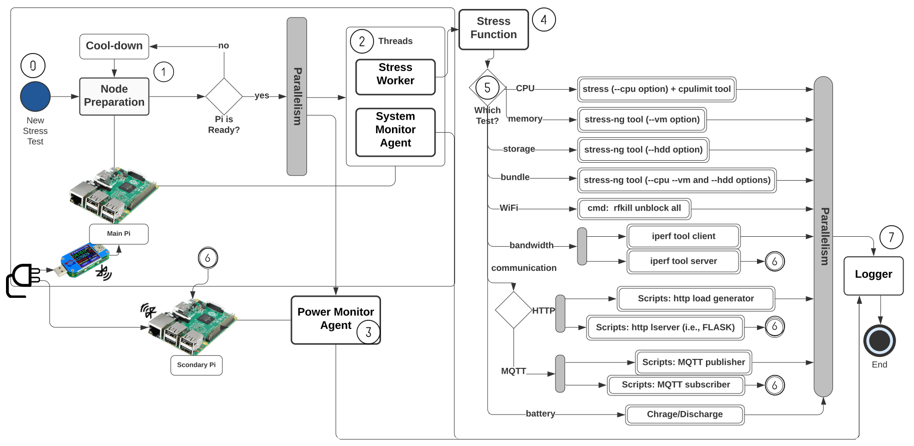

Fig. 2: *WattEdge*: The Proposed Edge Energy Measurement Framework.

with a 1200 mA current, and connectivity options such as WiFi, Bluetooth, HDMI, and USB, all in an ARM architecture standing on a Raspberry Pi OS platform. Comparative quad-core SBCs include: AML-S805X-AC (La Frite),[4] UDOO BOLT V8,[5] ASUS Tinker Board,[6] and Odroid-C2.[7] To obtain sufficiently accurate measurements: (a) Pis are configured headless; (b) connectivity ports such as USB and HDMI are disabled; (c) communication means such as WiFi and Bluetooth are turned off (unless specified otherwise); (d) Pis are not repositioned to avoid environmental conditions; (e) reasonable cool-down times are considered between each test; (f) tests last long enough and are repeated several times, and average and standard errors are reported for reliability; (g) certain tests are conducted at night to minimize network interference; and (h) the Raspberry Pi OS is updated with minimal installations.

Testbed: Two Pis are needed to emulate edge nodes (see Fig. 2). The main Pi runs the stress test program, i.e., Stress Worker, and the System Monitor Agent in concurrent threads. While stressing the Pi, a fine-grained monitoring agent continuously monitors and logs the whole system under test (e.g., CPU usage). To obtain accurate measurements, a hardware-level approach is adopted by employing a USB power meter model UM25C, which is highly accurate as shown in [23]. The meter reports the power and energy data in millisecond granularity via Bluetooth connection. Obtaining these measurements demands Bluetooth connection which influences the actual energy consumed on the Pi under test. Hence, the Power Monitor Agent, collecting the power and energy data, lives on a secondary Pi and is invoked remotely. The agent is connected to the power meter and reads the measurements during the stress tests.

[4] https://libre.computer/products/boards/aml-s805x-ac/
[5] https://www.udoo.org/docs-bolt/Introduction/Introduction.html
[6] https://www.asus.com/au/Single-Board-Computer/Tinker-Board/
[7] https://wiki.odroid.com/odroid-c2/odroid-c2

Node Preparation: ① prepares the edge node for a new stress test wherein no interference exists. Actions include disabling services such as MQTT broker which may have been employed for certain tests, disconnecting the Pi's battery and freeing up the memory, cache and swap. Disabling interfaces such as WiFi and Bluetooth, USB chip, and HDMI output are confirmed as well. A hot CPU can significantly influence the results, so a reasonable cool-down time is imposed.

Stress Worker: ② is written in Python and lives on a thread on the main Pi. It can run the specified test, e.g., CPU stress, for specific levels depending on the test plan by invoking the Stress Function.

System Monitor Agent: ② is run on a concurrent thread. It collects the monitored data every single second and, at the end of test, saves it on the storage (a 32GB micro SanDisk SDHC UHS-I card). The lightness, i.e., low overhead, will be confirmed in our empirical studies. Measured metrics include: timestamp, battery charge, CPU (usage, temperature, frequency, context switching and interrupts), memory usage, disk (usage, I/O read/writes) and bandwidth (packet sent/received). The *psutils* python module is employed to measure those metrics, except for the battery charge level which is measured by the *pijuice* module. The data is kept in memory until the end of test to avoid disk operations.

Power Monitor Agent: ③ is remotely invoked on the secondary Pi by Stress Worker. It gets connected to the power meter through Bluetooth and reads the power and energy data such as the wattage, current, volts, watt-hours etc. The data is finally saved on a local file.

Stress Function: ④ executes the specified test to stress a resource by ⑤ evaluating the test plan. It interacts with the secondary Pi depending on the test plan to run required services as well ⑥. Such interactions happen for running *iperf* server/client, HTTP server/client or MQTT publisher/subscriber.

Logger: ⑦ collects, merges and stores the data monitored by the two monitors.

4 Empirical Study

We use *WattEdge* to empirically analyze the 9 identified factors with a *Test and Learn* strategy. Note that all tests are conducted for 15 minutes and repeated 3 times, and average and standard errors are reported. We believe that 15 minutes is large enough for the purpose of building energy model and provides stable results. In reporting energy consumption results, we present all the y-axes at the same range, i.e., 0-1000mWh for the sake of easy comparison. We firstly obtain energy usage in idle state as a baseline for drawing an analogy between the impact of different factors. Then, each factor is analyzed by first reporting the overall energy consumption of that stress test, then subtracting already identified factors to obtain the actual energy consumption of the investigated factor.

Idle State Stress. To establish a baseline, a series of non-stress tests are performed wherein the Pi is idle. The measurements are labeled as "idle" in figures. The energy consumption, denoted as E_{idle}, was measured as a total of 179.33mWh on average. The average CPU usage was observed at 1.56%, confirming the lightness of *WattEdge*.

CPU Stress. The Stress Function employs the widely used *stress* tool to stress test the CPU [10] at full capacity and meanwhile it runs the *cpulimit* tool to throttle the usage at certain percentages (see Fig. 3). Results show increasing energy usage, starting from 179 to 699mWh (see Fig. 3). The upward slope appears constant for CPU usages up to 50% while it gradually reaches a flat plateau for usages above 70%. The reason for such behavior was found in the CPU temperature. The temperature throttling for Pi 3 is capped at 60C° upon which the CPU frequency is reduced automatically to avoid overheating. CPU usage below 70% never reached this threshold. This also warns us that (a) the long-running benchmark tests are more reliable and the accuracy of performance evaluations that last for only a few seconds/minutes as in [5] is questionable; and (b) the CPU frequency tends to be driven by the temperature in certain IoT use cases such as Smart Farming wherein devices are exposed to the sunshine.

Energy Model: By fitting the collected data to a linear regression, we can model the energy consumption driven by CPU usage as follows: $E_{cpu}(u) = (22.9u + 107.6) \times t$ where E_{cpu} stands for energy consumption in mW due to the CPU usage percentage u and t is the duration of the experiment in hours (if 15 minutes, $t = 0.25$). Given the interference of power management mechanism on the device, for a pure CPU-dominant model, we use a sub-model as $\widehat{E_{cpu}}(u) = (26.9u + 24.6) \times t$, measured by only considering CPU usages below 70% that gives the R^2 value of 99.7%. With the pure CPU model, a total energy model (E) is modelled as $E = E_{idle} + \widehat{E_{cpu}}$.

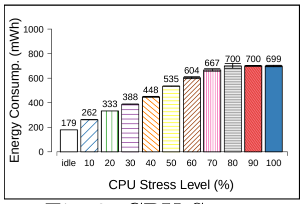

Fig. 3: CPU Stress

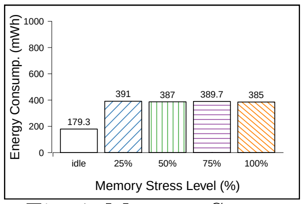

Fig. 4: Memory Stress

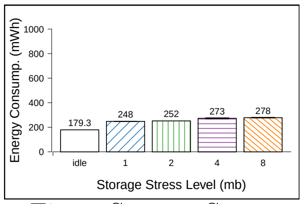

Fig. 5: Storage Stress

Memory Stress. To stress test the memory, the advanced version of *stress* tool, *stress-ng*, is employed. This allows the workers to stress on specific percentage of unused memory. A sample command is: *stress-ng* --*vm* 1 --*vm-bytes* 25% $-t$ 900s. That is, spawning 1 worker spinning on 25% of unused memory. Four different tests stressing on 25, 50, 75 and 100% of unused memory are performed.

The true impact of memory load on energy consumption appears rather similar for all memory loads (Fig. 4). Noticeably, a slight increase in energy consumption for lower memory loads is observed. This is due to the overhead on CPU context switching and interrupts. Technically, the *stress-ng* tool is continuously calling *mmap(2)/munamp(2)* and writing to the allocated memory. If the allocated memory is smaller (e.g., the 25% stress), the writing process is finished sooner and since the experiment lasts for minutes, this happens more often. Such context switching and interrupts appeared to be less for memory stress on higher loads. Further analysis shows an average 26% CPU usage in all memory experiments. The sole CPU stress at 26% constitutes for the energy consumption of 369mWh, which is 5% lower than the average energy consumption of 388mWh obtained in memory experiment. In other words, the memory can impose 11% more energy consumption (added to the idle) when under load which appears serious for highly power-constrained edge devices, while neglected in related works.

Energy Model: A memory-bound energy usage, $E_{memory}(m)$, here is equivalent to $E_{idle} \times \frac{m}{100}$, where m is the memory impact and for a Pi 3 B+ $m = 11$. Memory impact should be involved in the $\widehat{E_{cpu}}$ sub-model when the edge node is executing both CPU- and memory-bound IoT applications. This leads to an aggregated formula as: $E(u, m) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m)$.

Storage Stress. The aim of this stress is to evaluate if there exists any difference in energy consumption of (a) read and write, as well as (b) combined operations on storage and to what extent. This is evaluated using *stress-ng* tool. Observations for storage stress in terms of individual read and write operations for 15 minutes confirm that write operations (264mWh) consume more energy than reads (235mWh). A 11% difference in energy usage between read and write operations is seen. The main dichotomy in their performance can be attributed to the 79% more context switching occurrences by write operations. The question, however, is whether the increased energy usage is only due to the disk operations or the impact of memory and CPU, i.e., over-fitting? If so, how much? Given the 5% observed CPU usage, we ran a CPU stress at 5% to measure the net energy usage. The *WattEdge* framework reported 218mWh energy usage that means individual read and write operations can impose an extra energy usage of 9% and 26%. This gives an energy model as: $E_{storage^r} = E_{idle} \times 9\%$ and $E_{storage^w} = E_{idle} \times 26\%$, respectively. Having that, the $E_{total}(u, m)$ can be updated to $E_{total}(u, m) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage^r} + E_{storage^w}$.

In practice, the read and write operations are highly likely to exist simultaneously, whose energy usage pattern may be different. We ran the storage stress by continuously writing, reading and removing files of different sizes of 1, 2, 4, and 8 MB (the experiments at KB scale are done by [11]). Simplistically, such file sizes can resemble media files, ranging from image, to voice and video streams. Fig. 5 confirms that combined operations' energy usage will always be higher than that of read-only operations (i.e., 235mWh) and also higher (file sizes > 2M) than write-only operations (i.e, 264mWh). The CPU usage again remains similar to individual operations ($u = 5\%$, equivalent to 218mWh energy usage). At maximum, combined operations showed 278mWh energy usage. Excluding the impact of CPU ($218mWh$) and memory ($20mWh$), the net value increases due to storage is $278 - 218 - 20 = 40mWh$, equivalent to 22% imposed energy usage (added to the idle state) only due to combined storage operations which is considerable. We also evaluated larger file sizes, but the usage would not increase much further due to the SD card and CPU performance used in our testbed.

Energy Model: Involving CPU usage along with combined read and write operations of storage, i.e., $E_{storage^{rw}}$ (at high intensity), we can estimate that $E_{storage^{rw}} = E_{idle} \times 22\%$. More precisely, the energy usage observed in Fig. 5 shows a linear pattern that can be formulated as: $E_{storage^{rw}(l)} = (42.2l - 17.9) \times t$, where l stands for stress level: $l = \{1, 2, 4, 8\}$, and the accuracy is approximated at $R^2 = 92\%$. This insight leads the total energy model to $E(u, m, l) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + \big[[E_{storage^r} + E_{storage^w}] \vee [E_{storage^{rw}(l)}]\big]$.

Resources Bundle Stress. The *stress-ng* tool is able to stress all resources simultaneously. Four different levels of stress are imposed to the edge devices. The stress levels for CPU and memory are considered at 25%, 50%, 75% and 100% while the storage is undergoing simultaneous read and write operations at the size of 1, 2, 4, and 8 MB. This also could be deemed a realistic application which is not necessarily single-resource-bound.

Observations are shown in Fig. 6. The energy usage presents a considerable increase over the idle mode. The slight decrease for the fourth level, compared to the third level, once again has the root in the energy management mechanism on Pi devices. In this series, the mechanism is automatically activated for all levels, but at different points. It is also important to note that this mechanism was not activated for CPU usages below 70% in CPU-only stress while in combined resources this happened for even 25% stress. The exact impact of resource bundle needs further investigations. Take 25% stress as an example. According to the obtained energy model, we expect a total energy usage of $E(53.85, m, 1) = E_{idle} + \widehat{E_{cpu}}(53.85) + E_{memory}(m) + E_{storage^{rw}}(1) = 179.33 + 368.6 + 20 + 6.83 = 574.76$. This estimation is less than the observed energy usage in Fig. 6 (i.e., 627mWh). Analysing all four stress levels, an average extra usage of 8.4% for a resource bundle energy usage is obtained which may have the root in increased context switching due to resource (CPU, memory and storage operation) interference which causes this overhead.

Energy Model: The revised energy model, considering the impact of resource bundle energy usage, i.e., E_{bundle}, can be equal to a constant (i.e., β, here $\beta = 8.4$) value which is added to the total expected energy usage. This gives $E_{bundle}(u, m, l, \beta) = \big(\widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage}(l)\big) \times (\frac{\beta}{100})$. The following aggregated formula including the impact of resource bundle is hence gained: $E(u, m, l, \beta) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage}(l) + E_{bundle}(u, m, l, \beta)$.

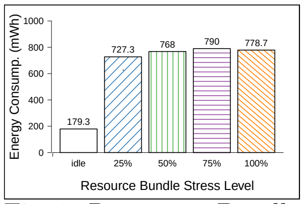

Fig. 6: Resources Bundle Stress

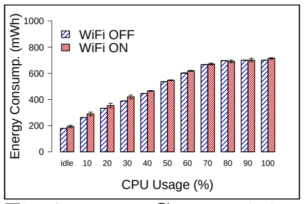

Fig. 7: Connectivity Stress (no comm.)

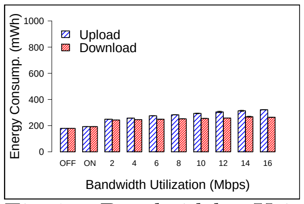

Fig. 8: Bandwidth Utilization Stress

Connectivity Stress. In the literature [13, 14], WiFi and Ethernet connectivity have gained more attraction due to highly adaptability to IoT domain. However, the limitations of employing the Ethernet in many wide-area IoT use cases makes it less worthy of consideration [19]. We redo experiments on CPU tests wherein

WiFi is enabled, but no data transmission is occurred. The results for when WiFi is off is used as a baseline for comparison. This study is essential since idle state for edge nodes is highly likely, yet the consequences are left unattended, particularly for connectivity impacts [14, 15, 18, 20].

Results in Fig. 7 reveal that the WiFi connectivity impact exists most of the times even though insignificant overall. In details, the lower the resource utilization, the higher the impact of the idle WiFi activities will be. Activities can be seen as responding to the beacon signals sent by a router or peers. A slight CPU temperature increase due to such activities was also reported by the System Monitor Agent. Precisely, an extra energy usage of between 0 and 17.38% for idle state is observed. Not a linear trend is seen, hence the average impact of WiFi enabled is considered here (7.82%). Further investigation is conducted for connectivity means such as Bluetooth, USB, HDMI and VNC which unexpectedly showed 18, 128, 6 and 18% extra energy usage, respectively. This and WiFi observations means that connectivity, at least for investigated means, can impose a sum of 178% extra energy usage.

Energy Model: A constant c representing the influence of connectivity, i.e., sum of WiFi-enabled (but idle), Bluetooth,USB, HDMI and VNC, can be involved as $E_{connectivity}(c) = E_{idle} \times (\frac{c}{100})$. This finding leads the aggregated energy model to the following: $E(u, m, l, \beta, c) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage}(l) + E_{bundle}(u, m, l, \beta) + E_{connectivity}(c)$.

Network Bandwidth Stress. Communications between edge nodes can be categorized in upload and download actions, regardless of the data type. However, one cannot ignore the importance of data transfer rate. To study it, an *iperf3* client is invoked on the main Pi and an *iperf3* server is invoked on the secondary Pi. Then, the client sends data in TCP mode at different rates to the server: {2, 4, 6, 8, 10, 12, 14, and 16 Mbps}. *WattEdge* measures the upload impact on energy usage. Similarly, the opposite roles are given to the Pis also to measure the impact of download operations.

Results, in Fig. 8, show the energy consumption due to upload and download at particular transmission rates. It is obvious that the more the bandwidth is utilized, the more the energy is consumed. Moreover, the upload (generating and sending data) appears more influential than the download (receiving data). There exists certain CPU usage, however, which needs to be taken into consideration to discover the real impact of bandwidth utilization. The CPU usage grows from 1.53% in idle state to 5% for the highest bandwidth utilization, i.e., 16Mbps. Excluding the idle state and CPU usage, a maximum of 58% and 26% increase in energy usage due to bandwidth utilisation for upload and download operations, respectively, is observed. This understanding will help making a reasonable decision for establishing or preventing communication in edge.

Energy Model: The bandwidth net effect, excluding CPU, is involved in the E_{total}, as the CPU effect is independently considered by E_{cpu}. It presents a linear pattern which in Pi 3 B+ is measured as: $E_{bandwidth^u}(r) = (14.8r + 173) \times t$ for upload with $R^2 = 99\%$ and $E_{bandwidth^d}(r) = (-0.9r + 172.2) \times t$ for download with $R^2 = 92\%$, respectively, where $r = \{2, 4, 6, 8, 10, 12, 14, 16\}$ stands for the rate of data transmission in Mbps. This eliminates the need for calculating the connectivity solely, i.e., $E_{connectivity}(c)$ as WiFi is under use. Involving bandwidth in E, we have: $E(u, m, l, \beta, r) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage}(l) + E_{bundle}(u, m, l, \beta) + \left[E_{bandwidth^u}(r) \vee E_{bandwidth^d}(r)\right]$.

Communication Protocols Stress. In practice, as we observed in storage analyses, the communication operations as upload and download are expected to co-exist. Hence, this study evaluates the energy consumption due to communication protocol families: request/reply and publish/subscribe in a full cycle of transmission. The *WattEdge* picks up the most popular ones from each category in IoT domain, i.e., *HTTP* and *MQTT*, respectively. The test scenario for HTTP is to send simple HTTP GET requests from a client to a server which is a Python Flask HTTP server echoing the message. For MQTT, a publisher sends messages to a subscriber on another node through a Mosquitto MQTT broker. The subscriber receives messages and publishes its response to the original publisher, similar to the HTTP study design for consistency. Note that in the request/reply family only client and server live and consume the energy while in the publish/subscribe there are publisher, subscriber and broker. The *WattEdge* framework comprehends such differences and assigns each role to the main Pi and the auxiliaries on the secondary Pi, depending on the test plan.

Tools such as *Jmeter* can be used as a client generating the load. Jmeter, however, is unreasonably heavy for SBCs such as Pi 3 B+. Since we aim at both considering the client and server impact of protocols, the *WattEdge* framework benefits from a lightweight customized python script for load generations. For the MQTT load test, the *paho* Python module is employed which efficiently generates and publishes messages with imposing negligible overhead. The load generator will concurrently send 10 to 90 requests/messages per second (in 9 tests) to the server in HTTP tests and to the subscriber in MQTT tests. This is the maximum load a Pi 3B+ could generate according to our configurations.

HTTP: The difference between client and server's impact appears insignificant (Fig. 9). Energy consumption increases from 179mWh in idle mode to 192mWh in WiFi-enabled and to 443 and 430mWh for maximum client and server stress, respectively. To reveal the net value for energy usage, we exclude the idle state and CPU usage obtained by the System Monitor Agent. More CPU usage was seen for the client than server. The net value is as 22% and 35% additional energy usage due to the HTTP client (E_{http^c}) and server (E_{http^s}), respectively, at maximum. The range of energy usage for different rates of concurrently, e.g., 10–90, is narrow and no linear pattern with a reasonable accuracy is observed. Hence, an average energy usage satisfies the inclusion of this factor which is observed at 18% (cl) and 30% (se) energy usage for client, $E_{http^c}(cl) = E_{idle} \times \frac{cl}{100}$, and server, $E_{http^s}(se) = E_{idle} \times \frac{se}{100}$, respectively. The cl and se stands for client and server's impact which for Pi 3 B+ will be 18 and 30, respectively.

MQTT: The energy usage depends on three entities: publisher, broker and subscriber. Fig. 10 shows that the MQTT mechanism adds to the energy us-

age, but insignificant differences exist between entities. The slight difference is seen, mostly for heavy loads (e.g., 80 and 90 messages), where the subscriber was dominant and the broker consumed relatively less energy than others. Excluding the idle state and CPU usage (observed at $< 5\%$ for entities), the net energy usage for publisher (E_{mqtt^p}), subscriber (E_{mqtt^s}) and broker (E_{mqtt^b}) at maximum load is measured as 28, 32 and 23% additional energy usage, respectively. Similar to HTTP, the range of energy usage is narrow, and the interest of simplicity, an average energy usage of 23 (pu), 26 (su) and 22% (br) for publisher, subscriber and broker is considered. This gives the following formulas: $E_{mqtt^p}(pu) = E_{idle} \times \frac{pu}{100}$, $E_{mqtt^s}(su) = E_{idle} \times \frac{su}{100}$ and $E_{mqtt^b}(br) = E_{idle} \times \frac{br}{100}$.
HTTP vs. MQTT: Overall, entities in HTTP consume much more CPU and energy than in MQTT. However, excluding CPU usage, the MQTT is imposing further energy usage. It should not be neglected that a third-party entity as broker exists in MQTT scenario whose energy usage must be considered. With this in mind, if we exclude the broker, the energy usage for both HTTP and MQTT becomes comparable. Moreover, this considerable usage due to communication raises the following question: "Is the task or data offloading, which requires communication between nodes, in edge computing always affordable?" With this insight, the total energy usage can consider finer-grained measurements based on entities performance in each protocol as follows: $E(u, m, l, \beta, cl, se, pu, su, br) = E_{idle} + \widehat{E_{cpu}}(u) + E_{memory}(m) + E_{storage}(l) + E_{bundle}(u, m, l, \beta) + \Big[\big[E_{http^c}(cl) + E_{http^s}(se)\big] \vee \big[E_{mqtt^p}(pu) + E_{mqtt^s}(su) + E_{mqtt^b}(br)\big]\Big]$.

Energy Sources Stress. With the widespread usage of the Lithium-ion batteries as energy storage, *WattEdge* employs a PiJuice HAT (i.e., Hardware Attached on Top–HAT) installed on the Pi to supply battery power. The PiJuice HAT features an on-board 1820mAh battery, original battery from Motorola Droid 2 (A955), and communicates with the Pi through GPIO Pins. A remotely controlled 5V Single Channel Relay Module handles the connection and disconnection of the charger.

The studies on the battery are to find out two behaviors: charging and discharging. For the former, we keep charging the battery from 10% to 98% and babysit the powering behavior. Fig. 11 shows energy usage during three hours. Starting from 10% charge, the PiJuice software asked the battery to get higher wattage. The wattage is reduced by reaching at the moderate charge level around 30-50%, increased at charge levels between 50-80% and then gradually decreased the powering until fully charged (163 minutes). After that, the incoming wattage is significantly reduced. This is evidencing that the battery software system considerably influences the powering which is worth considering.

On the discharge side, we are concerned about the efficiency of batteries. Hence, we drained a certain amount of the battery storage and measured how much energy it needs to obtain same amount of energy again. This revealed that the battery is returning 20% less energy. This is due to the internal resistance of the batteries. Also, the aging issue in Lithium-ion batteries deteriorates performance and increases internal resistance, all warning us of the energy sources considerations as well as energy consumers.

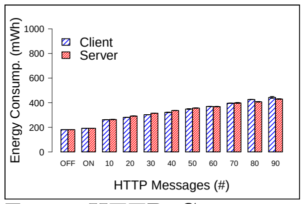

Fig. 9: HTTP Communication Stress

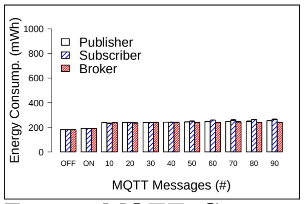

Fig. 10: MQTT Communication Stress

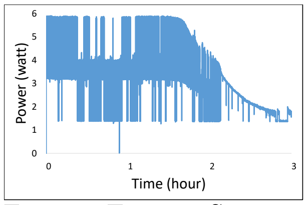

Fig. 11: Energy Sources: Battery Charging

The Test and Learn strategy provides us with an understanding of energy consumption factors in a practical measurement including: (1) *idle* state; (2) *computation* such as CPU, memory, storage and resource bundle (the storage is ignored for non-data-intensive applications); (3) *connectivity* if connectivity means are enabled; and (4) *communication* if data need to be transferred between the edge nodes using connectivity technology e.g. Wifi. For communication, if merely data transmission is of interest, bandwidth usage is included, otherwise only communication factors are included where specific protocols such as http or mqtt are used. If one wants to use the proposed model in practice, parameters of the model such as u, m, l, β, c, r, cl, se, pu, su, and br should be set based on the specifications of the edge nodes and running applications. In the next section, we validate the proposed energy model for an edge platform hosting a real-world application from agricultural domain.

5 Validation

An edge computing platform for Smart Agriculture—A Bird Deterrent System—is practically implemented, that under the hood is a cluster of Pis. In this use case, a bird deterrent device utilizing motion and camera sensors is equipped with a Pi to act as an edge node (see Fig. 12). The edge nodes reside in a local network and are connected to each other using a wireless router. The IoT application works as follows (Fig. 12): (1) a motion sensor continuously senses the environment. (2) If a motion is perceived, the camera sensor is activated to take a photo. A trigger is pulled to call an object detection application on the device, for processing. (3) We utilized a YOLO[8] (Real-Time Object Detection) function running as a web service using Flask (Python web framework) deployed on Docker containers for

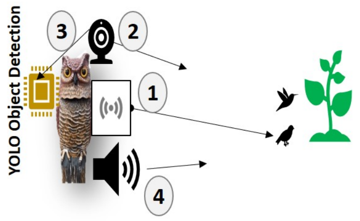

Fig. 12: A Pest Bird Deterrent Application's Workflow

[8] https://pjreddie.com/darknet/yolo/

such processing. (4) If a bird is detected in the image, the deterrent device is activated for a certain time duration.

In a cluster of nodes, we assume nodes can share resources for computation offloading (scheduling the YOLO object detection container on peers) to save energy since fake owls are intended to be powered by batteries and solar panels. The edge nodes form a Kubernetes (K3s) cluster (see Fig. 13). Given the event-driven nature of the application, a Serverless platform, OpenFaaS, is employed for deploying the core YOLO object detection function. Upon a photo taken by the camera, the function's endpoint is triggered and a request is sent to OpenFaaS gateway on the master node. The gateway invokes the function, and requests the photo from the Pi, generating the task.

Experiments are conducted in (A) local execution and (B) computation offloading scenarios to validate the energy model. We deploy the System Monitor Agent of *WattEdge* on each Pi to monitor the actual energy consumption. Using profiling, we obtain parameters that the energy model requires, i.e., u and l for estimations. Then we compare the energy consumption estimated by our energy model to the actual usage measured by the *WattEdge*. A Poisson distribution is used to generate task. Two Pis are involved in experiments: Worker 1 (task generator in both scenarios and task executor in scenario A) and Worker 2 (idle in scenario A and task executor in scenario B). A Master node (OpenFaaS gateway) also exists that is not involved in task generation and executions and only performs orchestration. Thus we do not discuss its energy consumption. The monitor reads energy consumption from the USB Meter locally. Hence, for accuracy, we include the Bluetooth-related energy usage in the model. Also, for consistency, our experiments last for 15 minutes and are repeated 3 times.

(A) For local execution, the actual energy usage of Worker 1 for the duration of test is reported at 573 by the USB Meter (see Fig. 13). This scenario involves computation: CPU (u), memory (m), storage (l) and resource bundle (β). Since the energy data is read locally, the Bluetooth connection energy usage (c) is considered in connectivity. In terms of communication, although Worker 1 is both task generator and executor, it still needs communication with Master node. This communication in Kubernetes is based on a request/reply protocol, so the client (cl) and server (se) roles must be considered for Worker 1. Hence, the total energy usage will be $E(u, m, l, c, cl, se)$. The variables such as $u = 41.43$ and $l = 1$ are obtained through profiling and constants are already known for Pi 3 (see Section 4). These two variables and other constants are used to estimate energy consumption using our proposed energy model. The estimation using the energy model results in total of 612.64mWh which demonstrates 93% accuracy com-

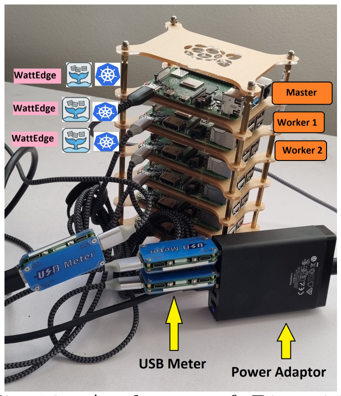

Fig. 13: A cluster of Pis with Kubernetes, OpenFaaS and WattEdge.

pared to the actual energy consumption. Using the same method for Worker 2, which was almost idle in this scenario, an accuracy of 91% is observed.

(B) For offloading execution, the actual energy consumption of Worker 1 is reported at 316mWh (less than previous scenario). Applying the energy model on the observed CPU utilization, storage and constant values, and comparing the estimated value with the actual one, a 97% and 98% accuracy in energy consumption estimation is obtained for Worker 1 and 2, respectively.

6 Discussion

Our findings show the significance of various factors in energy consumption of power constrained edge devices.

- Major factors: Connectivity prompts to be a major factor, neglected to a large extent by the literature. Confirming findings from previous studies (see Fig. 1), our findings pinpoint that the CPU and idle state are also major energy consumption factors.
- Moderate factors: Communication protocols and resource bundle are found to be moderate factors. The request/reply protocols are shown to be more power hungry compared to publish/subscribe models. The network bandwidth utilization and energy sources factors have moderate impact on the energy consumption.
- Minor factors: Impact of the memory and storage utilization appeared to be less significant.

We believe that the novel Test and Learn strategy in *WattEdge* significantly contributes to the literature by providing an accurate, low-cost, lightweight, fine-grained/holistic, and extensible framework. The *WattEdge* approach provides accurate enough measurements missed in software-based approaches while avoiding high-cost hardware instrumentation in hardware-based solutions. The high accuracy was ensured by running a diverse workflow (CPU-, memory- and storage-intensive as well as communication). The energy model developed based on *WattEdge* framework measurements. Two use cases were evaluated to validate the accuracy of energy models. An average accuracy of 95% are obtained in validation tests. The *WattEdge* framework is designed to be sufficiently lightweight as in practice it would not consume CPU usage of more than 1% as observed in our empirical studies. It is sufficiently fine-grained to allow a realistic and accurate measurement of a wide range of energy consumption factors: CPU, memory, storage etc. Finally, while the obtained power model is dependent on Pis, the *WattEdge* framework can be applied to other SCBs such as NVIDIA Jetson or Odroid [7] to obtain hardware-specific power models. In other words, SBCs considerably feature the same potential power factors but in different capacities. The modular design of *WattEdge* allows simple extensions for such resources. Besides, edge candidates other than SBCs can extrapolate the *WattEdge* idea.

7 Conclusions and Future Work

In this work, we conducted a comprehensive review to identify factors impacting energy consumption of edge devices first. Then, a framework called *WattEdge* was proposed to evaluate energy consumption of edge devices through a 9-step assessment using identified factors. These factors include: node's idle state, CPU, memory, storage, resource bundle, connectivity, network bandwidth, communication protocols and energy storage. *WattEdge* was implemented on a real SBC-based edge computing testbed while several empirical experiments were conducted. Based on the empirical analysis, an evolutionary all-inclusive energy model was developed. Our findings confirms that, in addition to major energy consumption factors such as CPU and idle state, connectivity uses significant energy in edge devices. This highlights the need for low power connection technologies and energy efficient communication protocols for the edge. Using real-world application in the smart agricultural domain, we validated our proposed energy model demonstrating a 95% accuracy of the model. In future, we will extend *WattEdge* to support a wider range of edge computing's requirements. This involves the study of: (a) renewable energy sources such as solar, (b) connectivity means such as Lora, and (c) communications protocols such as DDS.

References

1. Cisco Annual Internet Report (2018–2023) White Paper. Tech. rep. (2020), https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
2. Ardito, L., Torchiano, M.: Creating and Evaluating a Software Power Model for Linux Single Board Computers. In: Proceedings of the 6th International Workshop on Green and Sustainable Software. pp. 1–8. GREENS '18, Association for Computing Machinery, New York, NY, USA (2018)
3. Asaad, M., Ahmad, F., Alam, M.S., Rafat, Y.: IoT Enabled Monitoring of an Optimized Electric Vehicle's Battery System. Mobile Networks and Applications **23**(4), 994–1005 (2018)
4. Aslanpour, M.S., Gill, S.S., Toosi, A.N.: Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. Internet of Things **12**, 100273 (2020)
5. Bekaroo, G., Santokhee, A.: Power consumption of the Raspberry Pi: A comparative analysis. In: 2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech). pp. 361–366 (aug 2016)
6. Bouguettaya, A., Sheng, Q.Z., Benatallah, B., Ghari Neiat, A., Mistry, S., Ghose, A., Nepal, S., Yao, L.: An Internet of Things Service Roadmap. Communications of the ACM (2021)
7. Cabaccan, C.N., Reidj, F., Cruz, G.: Power Characterization of Raspberry Pi Agricultural Sensor Nodes Using Arduino Based Voltmeter. In: 3rd International Conference on Computer and Communication Systems. pp. 349–352 (apr 2018)
8. Dizdarević, J., Carpio, F., Jukan, A., Masip-Bruin, X.: A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. ACM Computing Surveys (CSUR) **51**(6), 1–29 (2019)
9. Fieni, G., Rouvoy, R., Seinturier, L.: SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers. arXiv preprint arXiv:2001.02505 (2020)
10. Hoque, S., De Brito, M.S., Willner, A., Keil, O., Magedanz, T.: Towards Container Orchestration in Fog Computing Infrastructures. In: 2017 IEEE 41st Annual Computer Software and Applications Conference. vol. 2, pp. 294–299 (jul 2017)

11. Hylick, A., Sohan, R., Rice, A., Jones, B.: An Analysis of Hard Drive Energy Consumption. In: 2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems. pp. 1–10 (2008)
12. Jiang, Q., Lee, Y.C., Zomaya, A.Y.: The Power of ARM64 in Public Clouds. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). pp. 459–468 (may 2020)
13. Kaup, F., Gottschling, P., Hausheer, D.: PowerPi: Measuring and modeling the power consumption of the Raspberry Pi. In: 39th Annual IEEE Conference on Local Computer Networks. pp. 236–243 (2014)
14. Kaup, F., Hacker, S., Mentzendorff, E., Meurisch, C., Hausheer, D.: Energy models for NFV and service provisioning on fog nodes. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. pp. 1–7 (apr 2018)
15. Kecskemeti, G., Hajji, W., Tso, F.P.: Modelling Low Power Compute Clusters for Cloud Simulation. In: 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). pp. 39–45 (mar 2017)
16. LeBeane, M., Ryoo, J.H., Panda, R., John, L.K.: Watt Watcher: Fine-Grained Power Estimation for Emerging Workloads. In: 2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 106–113 (oct 2015)
17. McCullough, J.C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A.C., Gupta, R.K.: Evaluating the effectiveness of model-based power characterization. In: USENIX Annual Technical Conf. vol. 20 (2011)
18. Mudaliar, M.D., Sivakumar, N.: IoT based real time energy monitoring system using Raspberry Pi. Internet of Things **12**, 100292 (2020)
19. Orsini, G., Posdorfer, W., Lamersdorf, W.: Saving bandwidth and energy of mobile and IoT devices with link predictions. Journal of Ambient Intelligence and Humanized Computing (2020)
20. Paniego, J.M., Libutti, L., Puig, M.P., Chichizola, F., De Giusti, L., Naiouf, M., De Giusti, A.: Unified Power Modeling Design for Various Raspberry Pi Generations Analyzing Different Statistical Methods. In: Pesado, P., Arroyo, M. (eds.) Computer Science – CACIC 2019. pp. 53–65. Springer, Cham (2020)
21. Rashti, M., Sabin, G., Vansickle, D., Norris, B.: WattProf: A Flexible Platform for Fine-Grained HPC Power Profiling. In: 2015 IEEE International Conference on Cluster Computing. pp. 698–705 (2015)
22. Rieger, F., Bockisch, C.: Survey of Approaches for Assessing Software Energy Consumption. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems. pp. 19–24. CoCoS 2017, Association for Computing Machinery, New York, NY, USA (2017)
23. Sagkriotis, S., Anagnostopoulos, C., Pezaros, D.P.: Energy Usage Profiling for Virtualized Single Board Computer Clusters. In: 2019 IEEE Symposium on Computers and Communications (ISCC). pp. 1–6 (jun 2019)
24. Serrano, P., Garcia-Saavedra, A., Bianchi, G., Banchs, A., Azcorra, A.: Per-Frame Energy Consumption in 802.11 Devices and Its Implication on Modeling and Design. IEEE/ACM Transactions on Networking **23**(4), 1243–1256 (aug 2015)
25. Toldov, V., Igual-Pérez, R., Vyas, R., Boé, A., Clavier, L., Mitton, N.: Experimental evaluation of interference impact on the energy consumption in Wireless Sensor Networks. In: 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM). pp. 1–6 (jun 2016)
26. Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., Wang, R.Y.: Modeling hard-disk power consumption. In: FAST. vol. 3, pp. 217–230 (2003)