Creating and Evaluating a Software Power Model for Linux Single Board Computers

Luca Ardito
Politecnico di Torino
Turin, Italy
luca.ardito@polito.it

Marco Torchiano
Politecnico di Torino
Turin, Italy
marco.torchiano@polito.it

ABSTRACT

The number of Single Board Computers (SBCs) is increasing, and so is the cumulative energy consumed by this category of device. Moreover, such devices are often always-on or running on batteries. Therefore, it is worth investigating their energy consumption to provide software developers and users with indicators for understanding how much energy the device is consuming while running a software application. In this paper, we explain a procedure for the creation of an energy consumption model of SBCs based on the usage of its components. We apply the procedure on a Raspberry PI 2 model B to test the model with a set of real applications. The results demonstrate the practical feasibility of the approach and show that estimated consumption values on our device have an average error of 2.2%, which is a good approximation without using external and expensive measuring devices.

KEYWORDS

energy consumption; software energy consumption; software engineering; energy consumption data; computer engineering

ACM Reference Format:
Luca Ardito and Marco Torchiano. 2018. Creating and Evaluating a Software Power Model for Linux Single Board Computers. In *GREENS'18: GREENS'18:IEEE/ACM 6th International Workshop on Green And Sustainable Software , May 27–26, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3194078.3194079

1 INTRODUCTION

We are increasingly surrounded by electricity-driven devices (e.g., home gateways and servers), many of which are running 24/7 [3]. In 2017, the Raspberry Pi official magazine announced more than 12.5 million Raspberry Pi devices had been sold since 2012[1], which does not even include competitor sales. According to Franck Beasnard[2] a Raspberry Pi 2 consumes approximately 14.45 kWh per year, and a Raspberry Pi Zero consumes about 5.26 kWh, assuming a load with an average power of 50% for all cores on both devices. At an order of magnitude of a million devices, it is easy to measure dozens of TWH per year attributable to this category of device. With such figure, we ought to evaluate their energy efficiency, and to do so, we need to measure their consumption in different contexts. In Linux laptop computers it is possible to use PowerTOP, a terminal-based diagnosis tool developed by Intel for monitoring the power usage of software applications. However, PowerTOP does not work on SBC devices because it uses the Advanced Configuration and Power Interface (ACPI) to collect information on how much power the device is currently draining. So, the easiest way to measure the power consumption of SBC devices is with a power meter, but this solution requires to instrument the device under test. An energy model for SBC devices offers an alternative to PowerTOP. The contribution of our work is the development of a repeatable and verifiable methodology for creating energy models for Linux Single Board Computer devices. We validate this procedure with Raspberry PI 2 model B.

The remainder of this paper is organized as follows:

- Section 2 introduces the related work;
- Section 3 describes process to get energy consumption data from the devices, and shows the experiment setup;
- Section 4 applies the process creating a power model for the Raspberry Pi 2 model B;
- Section 5 shows our results;
- Section 6 analyzes the threats to validity;
- Section 7 provides our conclusions.

2 RELATED WORK

Bekaroo et al. in [1] measured Raspberry Pi 2 model B power consumption while running a set of 20 usage scenarios selected from [2] and [6], and compared results with a desktop computer, laptop, smartphone, and tablet. Power measurement on each device is performed differently. On the laptop, Raspberry Pi, and Desktop computer, power is measured through an external watt-meter, while for the smartphone and tablet, it is measured by a mobile application called PowerTutor[3]. Results show that Raspberry Pi power consumption ranges between 2.10 to 2.20 W in an idle state, and can ramp up to 4.00 W while streaming a video. On average, a laptop computer averaged 17.68 W, while the Raspberry Pi was 4 times lower. The smallest energy footprint was scored by mobile devices with 0.44 W for the smartphone and 0.33 W for the tablet. The authors concluded the Raspberry Pi is a low-power device, and

[1] http://theverge.com/circuitbreaker/2017/3/17/14962170/raspberry-pi-sales-12-5-million-five-years-beats-commodore-64 Last Visited: 2018/02/01

[2] http://besn0847.blogspot.it/2016/01/electrical-consumption-of-raspberry-pi-2.html Last Visited: 2018/02/01

[3] https://play.google.com/store/apps/details?id=edu.umich.PowerTutor&hl=en Last Visited: 2017/11/28

GREENS'18, May 27–26, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5732-6/18/05…$15.00
https://doi.org/10.1145/3194078.3194079

adopted best practices for possibly reducing power consumption. Other researchers [4] built a power model for a specific version of the Raspberry Pi 2 model B based on the consumption of the CPU, Ethernet network, and Wi-Fi module in idle and load states. The power modeling for these components is based on collecting the power measured through an ADC (Analog to Digital Converter) and system-resource usage by the operating system. The idea is to translate the resource usage values into power. The authors showed their model results in a maximum error of 3.3%. Finally, Rieger and Bokisch [7] analysed eleven approaches for assessing software energy consumption and divided them into best practice approaches for writing energy-efficient code and measuring approaches. These two classes can be further divided into measuring approaches without model derivation, measuring approaches with model derivation, and measuring approach with power analysis attacks. This survey describes the current state of the art in the field. Finally, in a recent technical paper, Kaup et al. [5] created and compared energy models for different SBCs.

3 PROCESS DEFINITION

The energy consumption model for a Linux Single Board Computer can be used for converting data about resource usage into energy (measured in Joules) or power consumption (measured in Joule/sec) information. To create the model, we need to measure with a physical gauge the instant power consumption of the SBC during use and record the resulting usage data provided by the operating system. These two values must be linked because a given instant power consumption corresponds to a specific usage level of the SBC. Many SBC components have different usage profiles and, based on these, the resource consumes more or less power. So, we need to establish specific scenarios that trigger each profile accordingly. The goal is to obtain, for each component of the board considered, the resource usage and the instantaneous power consumption simultaneously. We can apply a linear regression on this data where the explanatory variable is the resource usage, and the dependent variable is the power usage in Watts. By combining these individual component models, we obtain a mathematical model, which includes component utilizations as variables and predicts the overall device output. In the following subsections, we follow a process for the energy model creation for a Linux SBC. For simplicity, we include Idle Consumption as a constant value in the model with CPU and Ethernet usage as variables.

This process is composed of power consumption characterisation, and power consumption estimation. First, we need to know how the power consumption varies over usage. Then, we estimate the power consumption based on the device usage.

Assuming that the idle power consumption is a constant value, we need tools to stress the CPU and network device. We suggest using *Sysbench*[4] for the CPU and iperf *iperf*[5] for the network. With these, we create scenarios (such as CPU usage at 10%, 20%, 30% until 100%, and send or receive data through the network) to run while measuring the actual instant power consumed by the device. We use the data obtained for creating the power model.

[4]https://github.com/akopytov/sysbench Last Visited: 2018/02/01
[5]https://iperf.fr/iperf-download.php Last Visited: 2018/02/01

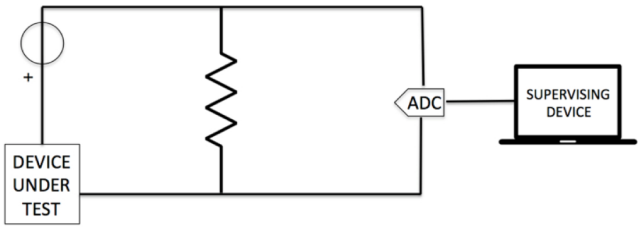

Figure 1: Basic circuit to measure instant power consumption

3.1 Procedure

The CPU is the first component we want to model. We stress it with Sysbench, so we need to measure the power consumption of the device while Sysbench is running. We identify two steps labelled S1 and S2.

S1: *Characterize the power consumption of the SBC CPU using the Sysbench software.*

We also model the network using *iPerf*, so measuring the device power consumption must also occur while *iPerf* is running

S2: *Characterize the power consumption of the SBC Ethernet adapter using the iPerf software*

3.2 Experiment Setup

The following outlines information about the experimental setup including hardware, software, and practical knowledge.

3.2.1 Hardware Instrumentation for power measurement. The physical power measurements require a standard setup, and for our execution, we selected the following set of devices:

- Voltage generator, (Metex MS-9150 1)
- Shunt resistor (0,05 Ω)
- Analog/digital converter (ADC) (NI USB-6210[6])
- Router/switch to allow time synchronization between the Device Under Test (DUT) and the supervising device
- Dupervising device[7] to run the software managing ADC sampling.

The voltage generator supplies 5V and current flows across the shunt resistance where it is measured by the ADC. Power is computed according to: $P = V \times I$.

Figure 1 represents the circuit setup, and Figure 2 shows the setup used in the laboratory where all the instrumentations listed above are visible. The 50mΩ resistor is soldered on a perfboard to improve the durability of the setup. The router is positioned on top of the power supply. On top of the power supply we can see the router used.

Alternately, it is possible to use specific devices that provide a single package with all the above elements, or equivalent, with APIs for automating data retrieval. A widely-used device is the Monsoon[8].

[6]http://sine.ni.com/nips/cds/view/p/lang/it/nid/203223 Last Visited: 2018/02/01
[7]http://www.ni.com/it-it/shop/labview.html Last Visited: 2018/02/01
[8]https://www.msoon.com/online-store Last Visited: 2018/02/01

Figure 2: Laboratory setup for instantaneous power measurement

3.2.2 Time synchronisation. Since we collect physical power measures on the support laptop while the usage and bandwidth measures are collected directly on the SBC, we need to set up a synchronisation mechanism between the two devices. An accurate synchronisation is essential to building an effective consumption model. The synchronisation mechanism is based on a time reference common to both the support device and the DUT. This approach is based on the ad-hoc protocol NTP. Low latency is mandatory because accuracy and test reliability are otherwise compromised. This choice was driven by the need to control the SBC through an SSH connection. We set up a router to connect to both the support device and the SBC. Then, we installed a local NTP server on the support device and synchronised the time between both devices before each test.

3.2.3 Sampling Frequency. The choice of sampling frequency is an essential aspect of the tests as a higher sampling frequency means a more accurate instantaneous power measurement. However, output file size must also be considered. As Saborido et al. state, a suitable sampling frequency is around 125 KHz [8].

In our case, we ran multiple tests in a single measurement session, so 250KHz (maximum allowed by our ADC) was not feasible because the size of the output file was too large. Therefore, we chose a sampling frequency of 1KHz. On the DUT, usage and bandwidth sampling were set to 1 sample per second to limit the overhead generated by the sampling application caused by obtaining the resources utilisation. The overhead is crucial because a software process collects the resource usage data and, as with all other software processes, it affects the consumption of the device on which it is executed.

4 CREATING THE POWER MODEL

In this section, we introduce the procedure to create and validate the power model. As discussed in Section 3, our goal is to translate resource usage into instant power consumption. In order to do this, we will use some specific software to build the consumption model of each component of the SBC. Our goal is to translate the resource usage into instant power consumption. To do this, we use specific software to build the consumption model of each component on the SBC. We perform an estimation of the CPU and network power and energy consumption based on the model defined in steps S1 and S2 detailed in Section 3.1. We label this activity Step 3 defined as:

S3: *Estimate the power and energy consumption of a set of applications running on the SBC.*

It is also necessary to define sub-steps to specify the applications included in the application set.

S3.1: *Estimate the power and energy consumption of the Avconv software*[9] *running on the SBC.*

S3.2: *Estimate the power and energy consumption of the OpenSSL software*[10] *running on the SBC.*

S3.3: *Estimate the power and energy consumption of the Zip software*[11] *running on the SBC.*

S3.4: *Estimate the power and energy consumption of the Quicksort sorting algorithm software*[12] *running on the SBC.*

We used these applications to create our usage scenario because they provide video play, cryptography, compressing/decompressing, and sorting features, which are typical use cases. In the following subsection, we describe the software run on the device for carrying out the steps. The software executing on the SBC is referred to as the Software Under Test (SWUT).

However, before starting, we need to determine the instant power consumption of the SBC while it is idle (when only OS-related processes are running). We subtract this value from the consumption obtained during the testing of each component to appear in the model.

4.1 Handling idle consumption

Before beginning the physical measurement, we need to stop all applications not needed for the experiment. This action is required because some third-party applications may run in the background causing noise in the instant power consumption data. This will also help correctly assign the power consumption to the SWUT and not other processes running simultaneously. However, it is not possible to stop or remove all processes managed by the operating system, which can display unpredictable execution. These unanticipated processes may affect the measurements, so we repeat the measurement routine 30 times and average the results to mitigate the effect of these processes as they will likely not be executed during each run of the experiment.

[9] https://libav.org Last Visited: 2018/02/01
[10] https://www.openssl.org Last Visited: 2018/02/01
[11] http://www.info-zip.org Last Visited: 2018/02/01
[12] https://bitbucket.org/softengrpi/loader-cpu Last Visited: 2018/02/01

4.2 CPU Model

Step 1 covers characterising the energy consumption of the CPU using Sysbench. We run the CPU at different speeds while measuring the instant power consumption of the device. However, Sysbench is designed to use the CPU at 100%. So, we use another program called CPULimit to limit the CPU load at 10%, 20%, 30%, until 100%. The pseudo code for extracting and storing the CPU information into a file is represented as the following:

Listing 1: CPU Model

```
run_cpu_test(maxCPUusage, sampling_time) {

  update_system_time()
  run_sysbench()
  run_cpu_limit(maxCPUusage)

  while (get_system_time() < MAX_TIME) {

    sys_time = get_system_time()
    //cpu_usage -> /sys/device/system/cpu...
    cpu_usage = get_cpu_usage()
    append_to_file(sys_time, cpu_usage)
    sleep(sampling_time)

  }

  stop_cpu_limit()
  stop_sysbench()

}
```

This procedure needs to be repeated for each CPU load we want to take into account.

4.3 Ethernet Model

Step 2 characterises the energy consumption of the CPU using iPerf. We run the Ethernet adapter at different speeds for inbound and outbound connections while measuring the instant power consumption of the device. The pseudo code for obtaining and storing the Ethernet information in a file is represented as follows:

Listing 2: Ethernet Model

```
run_cpu_test(bandwith,
  direction, sampling_time) {

  update_system_time()
  run_iperf(bandwithm, direction)

  while (get_system_time() < MAX_TIME) {

    sys_time = get_system_time()
    //eth_usage -> /proc/net/dev....
    eth_usage = get_cpu_usage()
    append_to_file(sys_time, eth_usage)
    sleep(sampling_time)

  }
  stop_iperf()
}
```

This procedure is repeated for each bandwidth and direction considered.

5 THE APPROACH IN PRACTICE

In this section, we apply the procedure described in Section 4 for creating a consumption model for a Raspberry Pi 2 model B.

5.1 Idle Measurements Results

Raspberry Pi idle data is listed in Table 1, which show an average idle value of 1.7694 W and an RMSE of 27 mW. The low RMSE suggests there exists little noise during the idle state and the values measured are reliable. Also, the CPU frequency of 633 MHz may be due to routine operations that are short-lasting and executed in quick bursts requiring a frequency scale to 900 MHz.

Table 1: Values for idle state

Quantity	Value	Unit
Usage	1.44	%
Power	1.7694	W
RMSE	27	mW
Frequency	633	MHz

5.2 CPU model generation results

The data acquired to generate the CPU power model is shown in Figure 3 (gray points). The coloured lines represent the linear regression of the 1st order generated based on the collected data. The x-axis represents the CPU usage from 0% to 100%, and the y-axis represents the power usage in Watts. The frequency scaling of the Raspberry Pi 2 is limited as the governors only allow a 1-step frequency change from 600 MHz to 900 MHz. The extracted mathematical function is:

$$P(u) = 0.01048u + 1.722 \quad (1)$$

where u is the % of CPU utilisation

5.3 Ethernet model generation results

The power model obtained for the network is based on the measurements of the power consumption of data upload and download. The collected data for the upload bandwidth (client-to-server) and its linear regressions are shown in Figure 4. The behaviour is very different from the CPU. Data is more grouped with less scattering. However, the data also spreads at higher bandwidths, but not at the 100 Mb/s mark, which may be due to the hardware not being optimised for medium-to-high transfers. Also, around the 50 Mb/s bandwidth marks a sudden change in power consumption with the values below this threshold around 1.6 W to 1.65 W, while above 50 Mb/s, the data fluctuates between 1.75 W to 1.8 W with smoothing

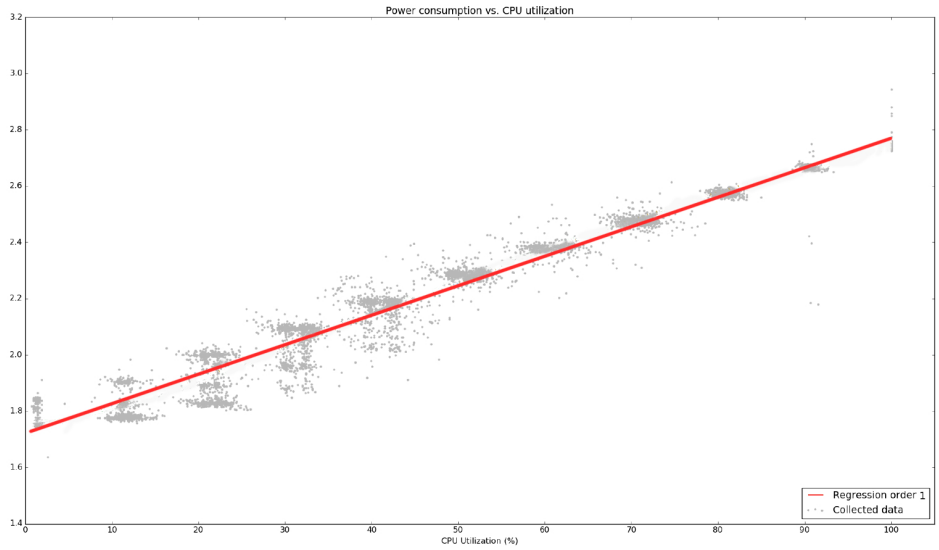

Figure 3: CPU Power Model

at the highest rates. Due to this sudden break, we decided to fit a linear split model where the split bandwidth was computed as the level of the maximum cumulative squared residuals (41.2 Mb/s). The bottom part of Figure 4 reports the squared residuals with those in dark red being relative to the full model and those in light blue for the split model. We observe a clear improvement with the latter. In fact, the RMSE for the full model (0.267 mW) is 50% higher than that for the split model (0.178 mW).

Figure 5 shows the power usage vs. the download bandwidth component of the network power model. We can see that download is more power-consuming than upload with the download power usage starting around 1.65 W and topping just above 1.9 W. On the other hand, the upload bandwidth starts above 1.6 W and reaches a maximum of 1.9 W. The sudden change in power consumption occurring at 50 Mb/s is more pronounced. Again, we apply a split linear model with the split bandwidth level at 42.5 Mb/s. The bottom part of Figure 5 reports the squared residuals where those in dark red are relative to the full model and those in light blue for the split model. We observe a clear improvement with the latter. In fact, the RMSE for the full model (0.273 mW) is approximately double that of the split model (0.145 mW). The full linear regression models computed in the analysis for the upload are shown in Equation 2 (full model) and Equation 3 (split model).

The models for the download are reported in Equation 2 (full model), and Equation 3 (split model)

$$P(r) = 0.00296r + 1.592 \tag{2}$$

$$P(r) = \begin{cases} 0.00164r + 1.613 & \text{for} \quad r \leq 41.2 \\ 0.00182r + 1.685 & \text{for} \quad r > 41.2 \end{cases} \tag{3}$$

The models for the download are reported in eq. 4 (full model) and 5 (split model).

$$P(r) = 0.003373r + 1.625 \tag{4}$$

$$P(r) = \begin{cases} 0.00162r + 1.654 & \text{for} \quad r \leq 42.5 \\ 0.00210r + 1.728 & \text{for} \quad r > 42.5 \end{cases} \tag{5}$$

5.4 Model testing

The results of the linear regression are shown in Table 2. The set of applications tested is composed of these applications Avconv, Quicksort, OpenSSL, and Zip as requested by Step 3. The set of applications tested include Avconv, Quick sort, OpenSSL, and Zip as suggested in Step 3. The first two results with Avconv show the conversion from an original mp4 file format to mkv and vice-versa of the file berry.mp4 [13] lasting 33 seconds. The third result with Avconv uses a longer file called burgers.mp4 [14], which lasts 59 seconds. For the Quicksort algorithm, the dataset size was chosen based on the timing for completing a run. We selected 21000 entries for the reverse order (integer numbers from 20999 to 0) and two 5000000 pseudo-randomised entries based on a fixed seed. Each dataset was sorted 30 times to provide adequate averaging on the time results. OpenSSL offers symmetric cryptography with a variety of algorithms (e.g., AES, DES, RC2, and RC4), public-key cryptography (RSA, DSA), secure hashing (MD, SHA), and many other functionalities. We tested the double-hashing capabilities of the Raspberry Pi 2 model B, so we used OpenSSL to perform a double SHA-256 hashing on a 400 MB input text file. Finally, we tested Zip program with a 400MB input text file.

The results show the consumption (energy and power) estimated by our model have a maximum error of 4.5% and, on average, the error is around 2.5%. Furthermore, the difference between real and predicted consumption values is not related to the experiment execution time. The estimations related to Avconv have a lower error than Quicksort and Zip. When we run the estimation test, the sampling rate is 10Hz. If there were many variations in the CPU

[13] http://softeng.polito.it/ardito/berry.mp4 Last Visited: 2018/02/01
[14] http://softeng.polito.it/ardito/burgers.mp4 Last Visited: 2018/02/01

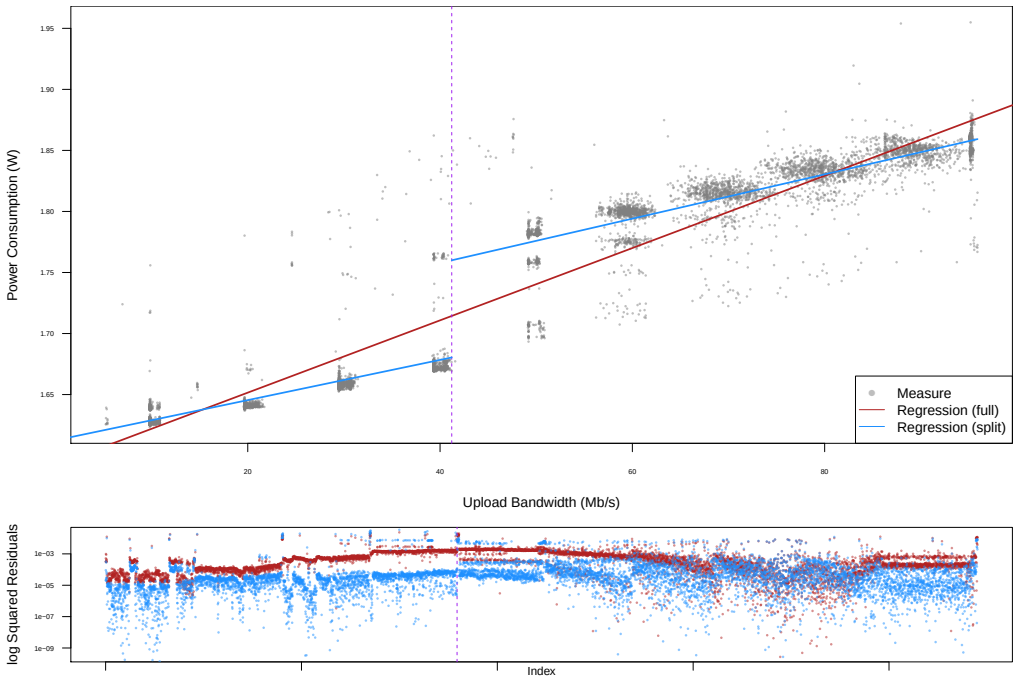

Figure 4: Ethernet Power Model (Upload)

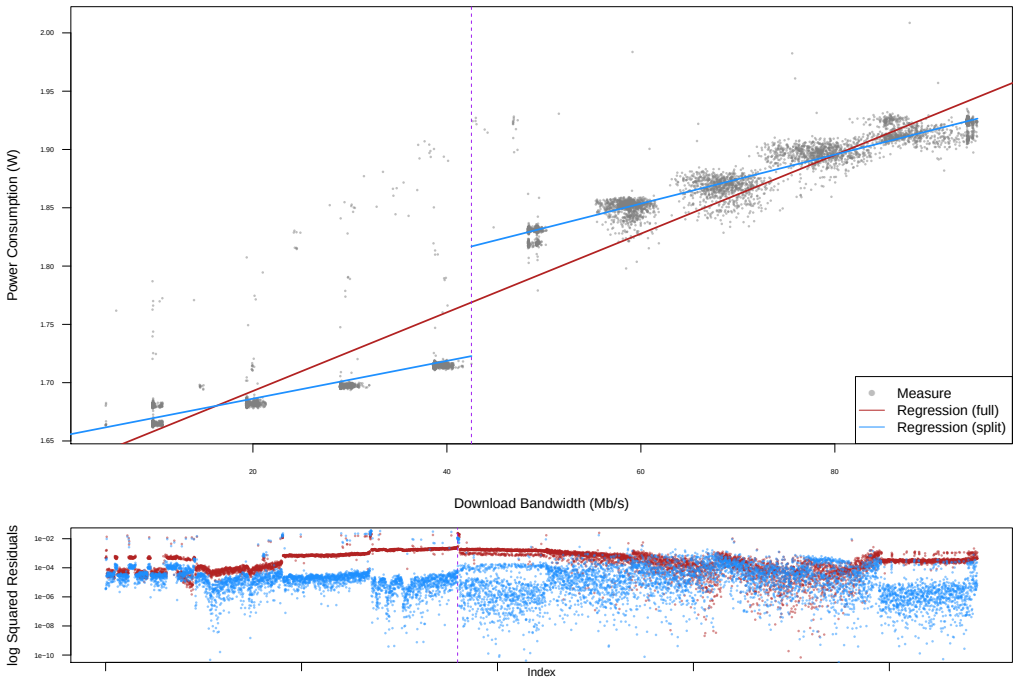

Figure 5: Ethernet Power Model (Download)

utilisation within the sampling interval, it would cause an error in the estimated consumption value. It is possible to reduce this error by increasing the sampling frequency.

However, we must consider the overhead caused by the larger number of readings from the CPU usage.

Table 2: Model Testing Results

	Power (W)			Energy (J)			
Test	Real	Predicted	Difference	Real	Predicted	Difference	Execution Time
Avconv berry mkv to mp4	2.355	2.317	-1.64%	2694.123	2650.48	-1.64%	1144
Avconv berry mp4 to mkv	2.495	2.482	-0.53%	3438.263	3419.757	-0.53%	1378
Avconv burgers	2.584	2.573	-0.43%	13244.292	13185.562	-0.43%	5125
Quick Reverse	2.653	2.586	-2.59%	106.102	103.431	-2.59%	40
Quick Random 1	2.568	2.635	2.55%	207.988	213.459	2.55%	81
Quick Random 2	2.523	2.642	4.50%	211.904	221.89	4.50%	84
Openssl	1.974	2.028	2.73%	943.619	969.387	2.73%	478
Zip	2.07	2.016	-2.68%	3326.233	3239.512	-2.68%	1607

6 THREATS TO VALIDITY

According to Wholin et al. [9] threats to validity can be classified as:

- Internal;
- Construct;
- External;
- Conclusion.;

Each is analyzed in the following subsections.

Threats to internal validity. The multi-tasking nature of the OS makes the attribution of power consumption to a specific SWUT difficult. We used a newly installed version of the OS with the minimum software to minimise the noise due to background processes. An idle power measurement was also performed to determine how much power is used by the background processes and OS maintenance operations.

Instantaneous Power Measurement Threats. The reliability of the power measurements was ensured by tracking the idle power consumption, which decreases the noise of the background processes in the system. The sampling frequency was chosen carefully, but some error should be taken into account as we could not use the maximum sampling frequency, as discussed in Section 3.2.3.

Model Prediction Threats. A significant issue of this research was handling the multi-core CPU of the RPi. For the single-core model testing, we needed to find a way to target which core was being used by the SWUT accurately. After some testing, we found an approach that worked providing we could enforce the SWUT to use a single core. This was possible only for Sysbench and the sorting algorithm application as the former includes a flag allowing for the specification of the number of threads to use, and the latter is guaranteed to be single-core based on its physical design. All other SWUTs (i.e., Zip, OpenSSL, and Avconv) did not allow the enforcing of a single-core behaviour, so we could not test them reliably on the single-core model prediction. Another threat is that our model may not consider other factors such as heating or power consumption by other components.

Threats to construct validity. Construct validity considers how consumption is measured such as the precision of the following measurements:

- Io Instant power consumption precision is impacted most by the precision of the measurement of the current and the noise produced by processes executed in parallel with the SWUT (see discussion above on internal validity)
- o Model estimation precision is key as the model may or may not consider relevant factors, such as heating or other device components, and, therefore, produce poor estimates.

Threats to external validity. The results obtained are valid only for the SBC tested (in our case, the results are valid only for Raspberry Pi version 2 model B), although the procedure described in this paper may be applied to other devices. While the results for a specific DUT cannot be generalised to another, it is possible to build a family of models that cover most SBC devices.

Threats to conclusion validity. Other threats that could render useless our tests include those from a low number of repetitions or a too short testing run. We handled these issues by:

- using long test times (10-15 minutes) for SWUTs where a single long run was better suited, and
- running the same test multiple times (30 times in our case) for the other SWUTs. Single repetitions were chosen to be long enough (a couple of seconds) for a reliable sampling.

7 CONCLUSIONS

In this paper, we show how to create an energy consumption model for a Single Board Computer. The described procedure explains how to create the model by converting the usage values of SBC components into energy consumption. It is necessary to exercise each component that appears in the model and measure its consumption with an external device during the test. A linear regression on the result produces an equation to convert the usage of the component into energy consumption.

We applied the procedure on a real case by creating a CPU and network power consumption model for a Raspberry PI 2 model B. We validated the model with an experiment to estimate consumption from the Avconv, Quicksort, OpenSSL, and Zip applications. The results obtained with the model, when compared to a reference physical measure, show an average error of 2.2%. Such a value suggests the proposed approach allows for a precise estimation of the energy consumption without requiring the use of external devices.

For future developments, we intend to create a tool to provide real-time energy consumption estimations, similar to PowerTop, that is currently not available for SBC devices

ACKNOWLEDGMENTS

The authors thank Marco Ardizzone for his practical contribution in the empirical experiment.

REFERENCES

[1] G. Bekaroo and A. Santokhee. 2016. 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)*. 361–366. https://doi.org/10.1109/EmergiTech.2016.7737367

[2] A. Carroll and G. Heiser. 2010. An analysis of power consumption in a smartphone. In *USENIX ATC*.

[3] Sarthak Grover, Mi Seon Park, Srikanth Sundaresan, Sam Burnett, Hyojoon Kim, Bharath Ravi, and Nick Feamster. 2013. Peeking Behind the NAT: An Empirical Study of Home Networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC '13)*. ACM, New York, NY, USA, 377–390. https://doi.org/10.1145/2504730.2504736

[4] F. Kaup, P. Gottschling, and D. Hausheer. 2014. PowerPi: Measuring and modeling the power consumption of the Raspberry Pi. In *39th Annual IEEE Conference on Local Computer Networks*. 236–243. https://doi.org/10.1109/LCN.2014.6925777

[5] Fabian Kaup, Stefan Hacker, Eike Mentzendorff, Christian Meurisch, and David Hausheer. 2018. *The Progress of the Energy-Efficiency of Single-board Computers*. Technical Report. Otto-von-Guericke-University, Institute for Intelligent Cooperative Systems. http://www.netsys.ovgu.de/netsys_media/publications/NetSys_TR_2018_01.pdf

[6] Aqeel Mahesri and Vibhore Vardhan. 2005. *Power Consumption Breakdown on a Modern Laptop*. Springer Berlin Heidelberg, Berlin, Heidelberg, 165–180. https://doi.org/10.1007/11574859_12

[7] Felix Rieger and Christoph Bockisch. 2017. Survey of Approaches for Assessing Software Energy Consumption. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Comprehension of Complex Systems (CoCoS 2017)*. ACM, New York, NY, USA, 19–24. https://doi.org/10.1145/3141842.3141846

[8] Rubén Saborido, Venera Arnaoudova, Giovanni Beltrame, Foutse Khomh, and Giuliano Antoniol. 2015. On the impact of sampling frequency on software energy measurements. *PeerJ PrePrints* 3 (2015), e1219. https://doi.org/10.7287/peerj.preprints.1219v2

[9] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.