Apis: A Toolkit for Federated Power Management

Adam Prey, Jason O. Hallstrom, Jiannan Zhai, and Chancey Kelley

Institute for Sensing and Embedded Network Systems Engineering, Florida Atlantic University

aprey2016, jhallstrom, jzhai, ckelle22@fau.edu

ABSTRACT

Embedded systems and Internet of Things (IoT) devices have been limited in application by constraints posed by batteries. Batteries add size, weight, and upkeep costs, limiting the lifetime of devices preferred to be small, lightweight, and long lasting. We present Apis, a software and hardware toolkit for federated power management in energy harvesting applications. By replacing batteries with rapid charging storage capacitors, circuitry to control federated energy storage, and software support to make this architecture accessible to developers, embedded devices can potentially run indefinitely with limited maintenance. We present the Apis hardware for controlling federated energy storage, supporting software, and experiments performed to validate the Apis model. The system is named Apis, after the genus for the Honey Bee, a creature dedicated to the harvesting and federated storage of energy resources.

1. INTRODUCTION

These obstacles are inconsistent with most visions of the future and are far from what is envisioned for ubiquitous computing [**?**].

Fortunately, energy harvesting techniques continue to advance, as interest in alternative energy sources continues to grow. Solar, wind, and hydroelectric energy harvesting have been studied and improved upon for many years, as these sources of power are widely available.

Lesser used methods for harvesting energy from the environment include vibrations, radio waves, magnetic fields, and thermal sources. With the variety of options available, having access to energy is less of a problem than it first seems, but storing and using this energy efficiently remains a challenge.

An alternative method for storing energy is the use of a capacitor or super capacitor, which can rapidly store and discharge energy without adding a significant amount of weight or size to a device. Capacitors come in a variety of physical sizes and construction, with different energy storage capabilities. A super capacitor can store more energy than other capacitor types, and is also capable of rapidly charging and discharging without damage, unlike a battery [**?**, 4].

The trade-off is that a super capacitor has a lower voltage limit than other capacitors. For many small electronics, this trade-off is perfectly acceptable, as these systems tend to require only 5 volts or less to function. These qualities make capacitors and super capacitors strong candidates for replacing batteries, despite the inability to store energy for long periods of time. This drawback may be manageable with the support of intelligent software services that aid in charging, discharging, and forecasting of energy use. This software support can bring embedded systems one step closer to becoming independent from batteries.

Through intelligent control of capacitor charging and discharging, in combination with energy harvesting and charge state reporting, battery-less devices can become a reality. Without the lifetime, size, weight, and maintenance limitations imposed by batteries, we come closer to the ubiquitous computing vision.

We present an energy measurement unit called a `service unit` for management of available energy. Service units are used to determine if enough energy is available to complete a given task before attempting to run the task. An example application would be determining how many times a radio could transmit a data packet before available energy is depleted, i.e., how many services are needed?

We present our hardware architecture for charging federated power systems, a software layer for hardware control and service unit management, experimental results, and our conclusions.

2. RELATED WORK

The work of Hester et al. [3] has greatly influenced our own. They present UFoP , the United Federation of Peripherals system, which uses separate capacitors to store energy for each peripheral, and a control circuit to allow the use of the peripherals. By using a comparator to manage charging of capacitors, the circuit determines whether to charge the capacitor for the microcontroller (MCU), or to charge peripheral capacitors. Apis differs, in that we use ADC readings to measure charge and make decisions in software by converting these readings into service units. Instead of having to manage ADC values, voltage drops, or other factors, developers only need to be concerned with the number of service units available for each peripheral, providing greater runtime flexibility for the Apis system. UFoP uses fixed comparators in the hardware layer to determine

what to charge, thereby limiting system dynamism. Apis uses service units to determine when a capacitor needs to be charged, instead of relying on the hardware comparator. This difference enables Apis to make intelligent charging decisions and reduces the affect noise may have on the system when deciding when and what to charge.

Hibernus [1] is a software platform used for energy harvesting systems that saves the system state before power failure and enters a sleep state. All volatile variables are saved to FRAM when a voltage threshold is reached, and the system state is restored from FRAM once the voltage threshold for activation is reached. When the system enters the sleep state, core registers are saved first, followed by RAM contents, including the stack segment, local, and global variables. Next, the general registers, stack pointer, and program counter are stored and the system enters a sleep state. Once an upper voltage threshold is reached, a restore procedure copies the stored information from the non-volatile memory back to the original memory locations. Once the restore is complete, the system resumes operation. Similar systems have been developed, including Mementos [5] and HarvOS [2], which allow programmers to add checkpoints in their code to restore the system to well defined points in the event of power loss. When a checkpoint is reached, a function stores the system state in non-volatile memory, similar to Hibernus.

While Apis lacks a checkpointing system, such a system can be added, though may not be needed. Apis allows the MCU to prioritize itself over the peripherals to ensure that it will have enough energy to keep the system active.

Other techniques involve using an energy-aware task scheduler, such as DEOS [6]. The Dynamic Energy-Oriented Scheduler performs code optimizations to combine and simplify subroutines to save power. Once optimized, DEOS schedules tasks based on priority, energy availability, and energy use. The system may require greater programming effort and more intensive debugging to ensure that the program is properly optimized, without introducing undesired behaviors. Automated combining of subroutines may provide another vector to introduce unwanted behavior to a system.

Apis currently lacks a scheduling algorithm and does not provide automated code optimization. A scheduling algorithm could prove to be a valuable addition, allowing the system to make better decisions for task execution. Service units can be readily integrated into many scheduling algorithms.

Most similar to our work is Hester's UFoP, as this system uses federated energy storage and circuitry to manage capacitor charging. Apis differentiates itself from this work, as it provides a support structure for energy harvesting. By integrating hardware, software, and service unit translation for developers, long-lasting, low-maintenance, battery-free systems become easier to create. Collecting real-value energy measurements with an ADC, and converting these values into service units; a measurement that corresponds directly to task execution, developers will have greater flexibility in designing battery-less systems.

3. SYSTEM ARCHITECTURE

The Apis architecture comprises hardware and software. The hardware layer includes capacitors used for energy storage, energy harvesters, power distribution circuitry, peripherals, and a microcontroller. The power distribution circuitry uses an off the shelf, ultra-low power, switch mode power supply that has been configured as a voltage regulator for the Apis application.

The software layer includes drivers for the various circuits, as well as the charging algorithm used to control charge and discharge of the capacitors, when energy is available. Service units are a key aspect of the software, enabling Apis' unique decision-making strategy. The service unit drivers measure voltage across each capacitor, and convert the voltages to service units, while also allowing the capacitors to provide power when needed.

3.1 Hardware Layer

The Apis circuit board shown in figure 2, with the circuit schematic shown in figure 1, isolates a storage capacitor from a voltage bus and prevents the capacitor from discharging energy while isolated. The isolation is achieved by the three MOSFETs in zone 1 of the schematic. The storage capacitor is used to provide power to a TPS62737 switching regulator (Zone 4). The output voltage of the capacitor can be discharged to a minimum of 3.3 V through the switching regulator. The output voltage can be adjusted for application specific purposes. We chose 3.3 V, as it is a common operating voltage for many microcontrollers and peripherals. While other regulator choices are available, the TPS62737 is a high-efficiency switching regulator with a quiescent current in the hundreds of nano-ampere range, with the ability to achieve high efficiency with low current loads. With the ultra-low leakage current of the MOSFETs and low quiescent current of the switching regulator, the storage capacitor is effectively isolated from the system unless needed.

The Apis power distrubtion circuit is designed for multiple use cases. When the Apis circuit board is configured for the MCU, it will always be in the power enable state. The MCU running the Apis software will need consistent access to power to prevent the system from shutting down. The design difference is achieved with a solder jumper (JP2) in Zone 1 of the schematic.

Zone 1 of the schematic shows the VIN and EN_IN pins of the Apis circuit board. The solder jumper JP2 is used to set the input MOSFETs as either default on or default off, for use with an MCU or peripheral, respectively. These MOSFETS are configured to serve as a bi-directional load switch. JP2 is connected to VIN and ground at either end, with R10 and the gate of MOSFET Q5 connected to the center pin of the jumper. When R10 is connected to VIN, it serves as a pull up resistor, resulting in the default state of the board being on. Connecting the same resistor to ground provides a pull down resistor, setting the default state of the MOSFETs to off. By configuring the board in the on state, MOSFET Q3 (Zone 3) is removed from the circuit. As a result, the $\overline{EN1}$ and EN2 pins of the of the TPS62737 regulator (Zone 5) are tied to ground and VIN, respectively. This configuration defaults the regulator to the enabled state, and further reduces the quiescent current of the board, as current is no longer used to activate MOSFET Q3 to power the regulator.

Zone 2 of the schematic features the inputs for connecting the bulk energy storage bank; any power source attached to J5 will be the primary source of energy for the the Apis circuit. Zone 3 includes the control pins that enable or disable the switching regulator. Q3 and Q4 are used in the event that the MCU GPIO pins are not able to provide suf-

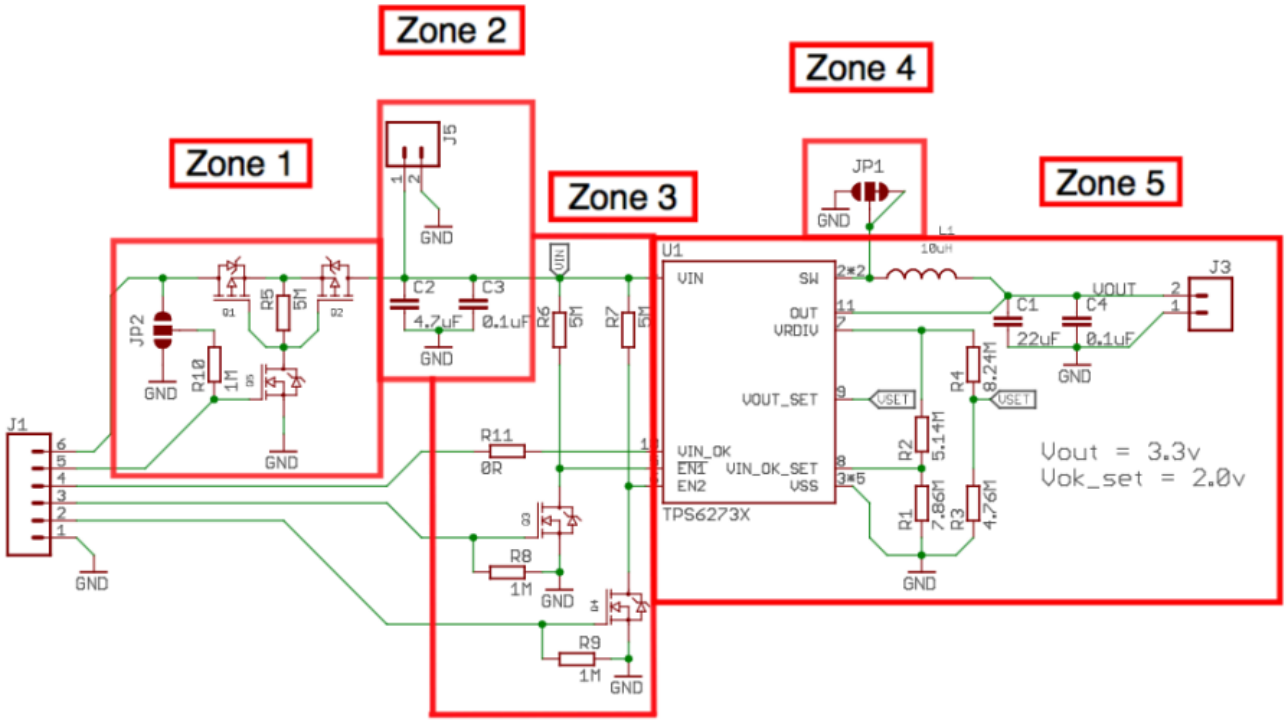

Figure 1: Schematic of the Apis Circuit Board

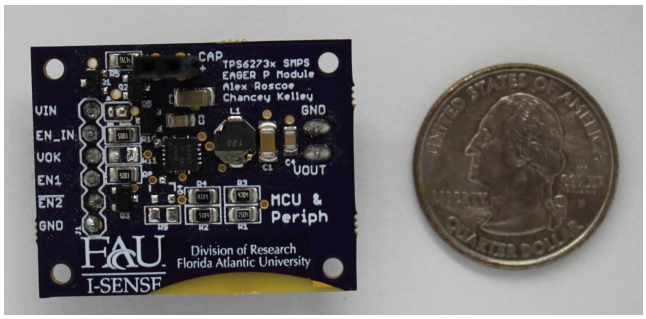

Figure 2: The Apis Circuit Board

ficient voltage to activate the enable pins of the switching regulator. R5 through R10 have high resistance values to further reduce the quiescent current used when activating the system, at the cost of an increased response time.

Zone 5 presents the switching regulator and associated output circuitry. The regulator reduces input voltage from the storage capacitor to a regulated 3.3 V. This output voltage can be changed to accommodate lower voltage devices by adjusting the voltage divider ratio on the VOUT_SET pin. The section allows the MCU to control the switching regulator, allowing peripheral power control. The VIN_OK pin of the regulator provides quasi-digital tracking of VIN for the MCU until a minimum voltage threshold is met, where tracking is turned off. When VIN_OK reaches a minimum voltage threshold, set by VIN_OK_SET, the tracking will activate. When VIN drops below the VIN_OK_SET threshold, VIN_OK becomes a ground signal. These additional features and circuitry enable the storage capacitors to be used efficiently with the TPS62737 regulator's excellent operating efficiency.[1]

Capacitor selection is used to tune the Apis system for specific applications. Selecting a large capacitor provides greater energy storage, at the cost of charging speed. The inverse is also true. The storage capacitors used for testing were 470 μF. Individual application needs will vary, and it is crucial for programmers to be aware of the selected capacitors, as features of the software layer require this information. The need for this information is discussed in Section 3.2.3.

Apis software services are capable of working across a variety of MCUs, enabling greater design flexibility. For development and experimentation, the MSP430 and ATMega328 MCUs were used. These MCUs are common choices among embedded developers.

3.2 Software Layer

We next describe the software layer and its constituent components. The main features of the Apis software layer include the software drivers for Apis circuit boards, service unit monitoring services, check functions for service unit availability, and a charging algorithm for storage capacitors.

[1]Zone 5 provides another jumper for the regulator. This jumper connects the SW pin of the TPS62737 to ground, as certain versions of this regulator require this connection, while others do not.

For software developers, Apis provides an easy to use framework to support the development of battery-less systems.

3.2.1 Service Units

The backbone of the software layer is a calculated value known as a service unit. Service units represent the energy cost to execute a single task. The use of service units can be compared to a bank ledger for task execution. Each execution costs one service unit, and the software keeps the ledger up-to-date. This ledger enables developers to be fully aware of what functions can be executed. For instance, the service unit balance for radio transmissions informs the developer (or application) how many times a data packet can be transmitted by the radio before transmissions are no longer available. As the service units are used, the ledger is updated, and the system can determine if it is possible to run a task.

The number of available service units specifies how many times a task can be executed before the associated storage capacitor can no longer supply the required energy. Service unit availability is the basis for most decisions within Apis. The relationship between the service unit for a task and the energy cost is the average voltage drop of the storage capacitor when the task is run. For example, suppose the radio used in the previous example causes voltage drops between 0.01V and 0.07V for each transmission. By collecting enough data samples when transmitting, an average voltage drop can be calculated for use as service unit's costs Typically, the standard deviation across voltage drops is small to be insignificant in determining service unit costs. A caveat, is that this is only true when the parameters(e.g. power levels, packet length) of a task's execution are held constant.

The average voltage drop for each task must be determined experimentally. The relationship is used to build a predictive model of available service unit availability based on remaining storage capacitor voltage. The average voltage drop of a task is then used to determine how much voltage the service unit will cost a storage capacitor. The energy costs remain consistent, despite being sourced from a capacitor due to the linear discharge from the switching regulator used in the Apis circuit.

The experimental method used to determine the relationships between service units and voltage drop requires several trials of repeated task execution with the Apis circuit and storage capacitors. The task is configured to be powered only by the storage capacitor (not the harvester), which is fully charged at the beginning of each trial. Tasks are run until the storage capacitor discharges to 3.3 V, the cutoff voltage of the switching regulator. Other thresholds may be used if selecting a different switching regulator for the circuit. In some cases, a higher voltage threshold is needed to prevent a peripheral from failing. Such circumstances can arise when using a peripheral with a minimum voltage rating higher than the selected regulator's voltage threshold. Using the MCU's ADC, the storage capacitor voltage is logged before and after each task execution. The collected readings are then used to calculate the energy cost. Thirty trials were run for each task and averaged to calculate energy cost.

The following equation provides a simple model for service unit availability for a given capacitor voltage.

$$SU = \left\lceil \frac{CapVoltage - 3.3}{AverageVCost} \right\rceil$$

SU is the number of available service units, $CapVoltage$ is the voltage of the storage capacitor, 3.3 is the voltage cutoff for the regulator used, and $AverageVCost$ is the average ongoing cost voltage for the task. This average varies by capacitance; an advanced model could be parameterized by capacitance; however, this simplified approach suffices.

By subtracting the cutoff voltage of the regulator from the storage capacitor's voltage, we calculate the voltage available for use. Dividing by the average voltage drop from a service unit and rounding up to the nearest whole number, an approximate number of available service units can be found for a given voltage.

The equation below is a variation of the first, but subtracts an offset.

$$SU = \left\lceil \frac{CapVoltage - 3.3}{AverageSUCost} - Offset \right\rceil$$

In some cases, it may be necessary to include the offset when modeling service unit availability due to electrical noise or other factors in the system. For example, if a peripheral has a large difference between minimum and maximum voltage drops, or a large standard deviation, the average may not be accurate enough to be used by itself. The offset allows for more tuning to ensure that predicted service unit availability is never greater than actual service unit availability.

3.2.2 Peripheral Board Software Driver

The Apis software driver provides a set of functions to control the charging of storage capacitors and discharge to external peripherals. All that is required to control the board is to send a high signal to enable, and a low signal to disable ()EN_IN, EN1). The former, which enables charging a capacitor, and the later for discharging of a capacitor. To use the software driver, the developer must specify the MCU pins connected to the enable and charge pins on each peripheral board.

These portion of the header uses `#defines` and structs to assign pins and ADCs for Apis to control capacitor discharge. The `#defines` provided allow developers to specify what parts of the system to check on by passing their assigned values to an ADC read function. This provides flexibility allowing for individual capacitors, the voltage bus, or the entire system's energy availability be read by the ADC when needed. The driver functions include device enable (for individual circuits), charging enable ()for individual circuits), charging enable for all boards and the respective disable functions.

3.2.3 Charging Cycle

The Apis charging cycle uses one of MCU's comparators and as many ADCs as needed, to determine when to begin charging, which capacitors need to be charged, and when to stop charging. To begin the charging cycle, there must be a voltage higher than the comparator's voltage threshold on the power bus. Once the threshold is met, the comparator triggers an interrupt to wake the Apis system, and a flag is set to check if charging can be done. Capacitor voltage is converted to joules ($Joules = \frac{1}{2} Capacitance * Voltage^2$).

The conversion is used to determine if there is enough energy on the power bus to increase the energy stored in each capacitor. To stabilize voltage coming from the harvester to the power bus, a capacitor included between the harvester and ground. This added capacitance must be considered when determining if enough energy is available before

charging begins. In some cases, the power bus may have a sufficiently large voltage to activate the charging cycle, but not have enough energy capacity to charge any capacitors.

If the bulk charge on the power bus is greater than the bulk charge of a given capacitor, the capacitor is added to the charging cycle. If the bus does not have enough bulk charge for a given capacitor, the capacitor will discharge itself until it is equal to the power bus. Developers must provide the total bulk capacitance of the power bus and storage capacitors so the software can perform the necessary calculations and prevent unintended discharging of the storage capacitors.

When ready to charge a capacitor, the corresponding device is disabled (except in the case of the MCU) to allow faster charging. Charging then occurs until a usage defined minimum service unit threshold is reached. The cycle then switches to the next capacitor and repeats until all minimum thresholds are met, or until bus energy is no longer available. Next, the service unit threshold is increased by a user-defined value, and the cycle repeats until all capacitors are fully charged, or bus power is no longer usable. The process continues iteratively. If no capacitor can be charged, the charging cycle ends, and the system goes to sleep. During the charging cycle, a priority order is established in the event of multiple capacitors needing charge. The MCU's storage capacitor always has the highest priority, and it is recommended to have a higher minimum service unit threshold to keep the system running. For other devices, charging priority should be based off task priorities.

3.2.4 Service Unit Monitoring

When the the Apis software is not in low power mode, continuous ADC readings are collected from the storage capacitors to track the number of available service units. This allows Apis to track which tasks can be completed, and how many times they can be completed before running out of energy. These balances are regularly updated to account for leakage current. While the Apis circuit is designed to mitigate leakage current, some leakage is unavoidable.

4. EVALUATION

We validated our model by conducting a series of experiments with the Apis system. The experiments focus on calculating the actual energy costs of peripheral functions compared to modeled costs. The purpose of the experimentation is to demonstrate how service units can accurately create a model for peripheral function costs.

4.1 Dummy Loads

The first series of tests used three simulated loads. Loads consisting of LEDs and resistors were used in series to create current draws of approximately 2mA, 4mA, and 6mA, respectively. An MSP430 microcontroller was used. For each experiment, the MSP430 was used to enable a LED load through the Apis peripheral board for 250ms, and then disable the LED load. This cycle repeats until the capacitor reaches a threshold voltage of 3.3 V. The threshold is dictated by the Apis board's TPS62737 regulator, which has a dropout voltage of 3.3 V. The test loads were connected to the peripheral boards, which were in turn connected to the MSP430. The MSP430's ADC inputs were connected to each capacitor, as required by the Apis architecture. The capacitors were 470 μF each and the energy source a bench

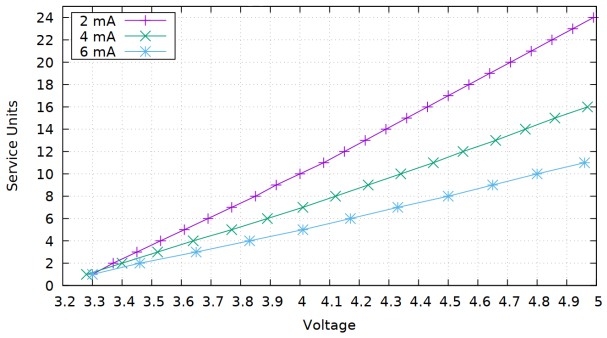

Figure 3: Measured Service Units / Voltage

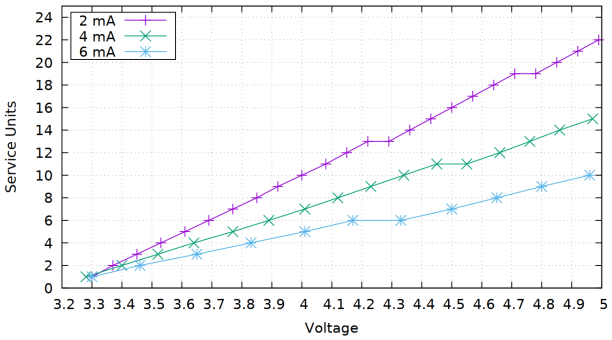

Figure 4: Estimated Service Units / Voltage

power supply set to 5 V. Charging was controlled manually by connecting the power supply to the power rail, and disconnecting once all capacitors were charged. Each trial was run 30 times; measurements were averaged.

Figure 3 presents the measured results of the dummy loads with 470 μF capacitors. The horizontal axis represents voltage, and the vertical axis represents measured available service units.

Average voltage drops of 0.08 V, 0.12 V, and 0.19 V were observed for the 2mA, 4ma, and 6ma loads, respectively. Using these averages, we are able to plot the graph shown in figure 4 using the service unit model. The horizontal axis represents voltage, and vertical axis represents estimated service units.

Figure 5 compares the measured and estimated service unit graphs. The horizontal axis represents voltage, and vertical axis represents service units. Apis provides an accurate model of service unit availability. It should be noted that at certain voltages the estimated service units do not increase, resulting in a plateau. This is due to rounding. The service unit model can be tuned to more precisely reflect the measured service unit values, however this creates the risk of overextending the model. When this occurs, possible system failures become more likely due to Apis making decisions with service units that are not available. By allowing greater deviation between measured service units and modeled service units, a buffer is formed that prevents the system from over expending available resources.

4.2 CC1101 Radio Module

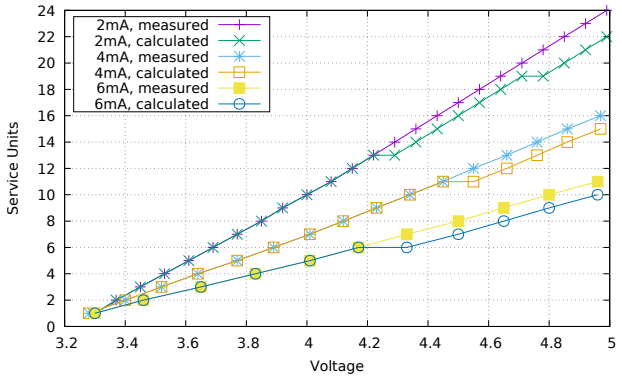

Figure 5: Measured and Estimated Service Units

The next experiment focused on a real peripheral device to further test and validate the model. An ATMega328 microcontroller was used for these experiments. As wireless communication is a key aspect of many embedded platforms, the CC1101 radio module was selected as the test device.

The experimental setup is analogous to the dummy load experiments. The power bus was again connected to a bench power supply set to 5 V. The SPI pins of the MCU are only enabled before a radio transmission to prevent parasitic power.

In this experiment, the software initializes the ADC and the CC1101, and then turns off the C1101. The test program transmits messages with a single byte, 125 bytes, and 254 bytes, respectively, to examine the effects of message size at each of the CC1101's transmit power settings. The power settings used range from -30dBm to 7dBm. The result is 42 sets of measured voltage drops. The service unit formula is implicitly parametrized by device settings, such as transmission power and packet size.

The software runs 30 trials for each power setting and message length configuration. During each trial, the software enables capacitor charging until an ADC reading of 5 V is reached. Charging is then disabled. Next, the CC1101 is enabled. Once ready to send a message, the MCU enables SPI and a message is transmitted. Once transmitted, SPI and power are disabled. ADC readings of capacitor voltage are logged via UART. The process repeats until the ADC reads a voltage below 3.4V, the observed cutoff for the CC1101.

The data samples from each set of trials were averaged and processed in the same manner as before. The results are presented in figures 6, 7, and 8. Each plot covers the three packet lengths, with the seven different transmission power levels. The horizontal axis represents voltage, and the vertical axis represents measured service units. Similar to the dummy load case, linear discharge across the capacitor is observed.

The estimated service unit plots for the CC1101 are presented in figures 9, 10, and 11. The horizontal axis again represents voltage, and the vertical axis represents the estimated service units. Using conservative offset values for each setting, the estimated number of service units accurately reflects the number of measured service units. The data demonstrates the validity of the service unit model and is explored in greater depth in the Quantitative Summary

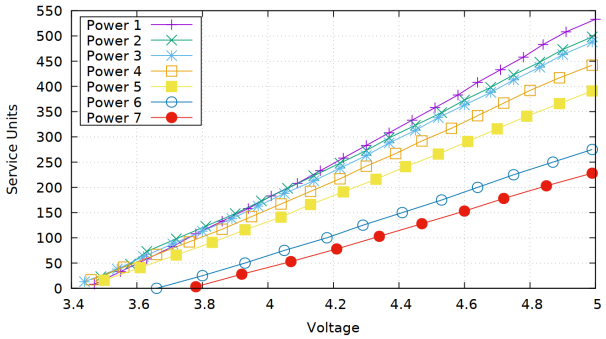

Figure 6: Measured Single Byte TX

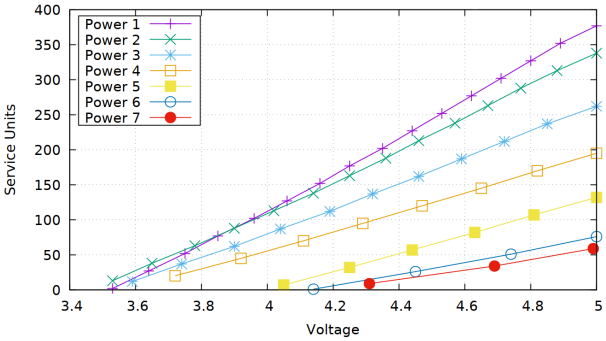

Figure 7: Measured 125 Byte TX

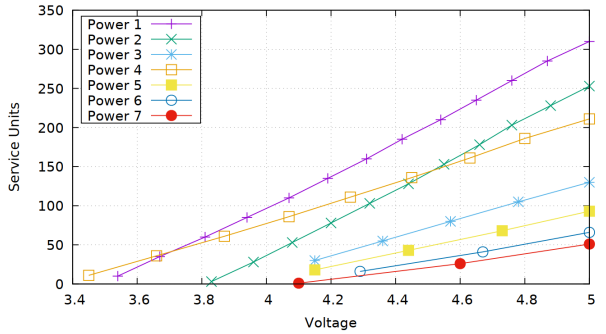

Figure 8: Measured 254 Byte TX

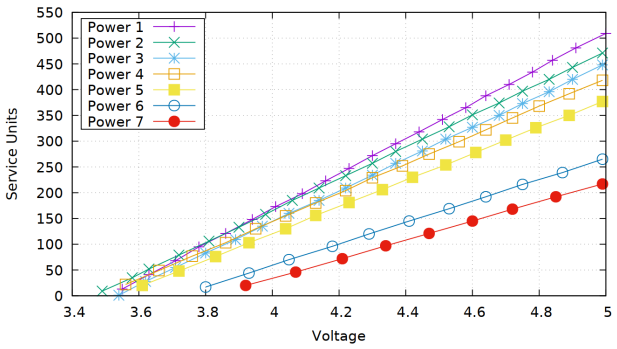

Figure 9: Estimated Single Byte TX

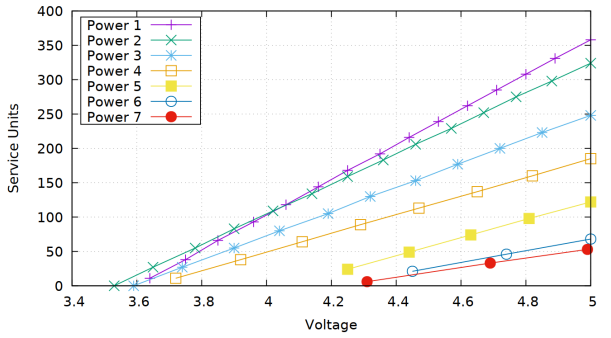

Figure 10: Estimated 125 Byte TX

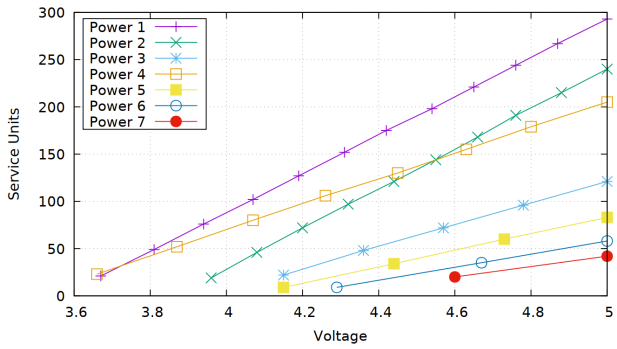

Figure 11: Estimated 254 Byte TX

section of this paper.

An interesting note is that certain configurations of the CC1101 result in intersecting service unit availability curves. This can be used in system optimization decisions, highlighting where trade-offs in packet length and transmission power can be made. The importance of collecting experimental data for the service unit model can not be overstated for developing Apis applications.

5. CONCLUSION

Our work focuses on the development of a battery free architecture for use in embedded system and IoT applications. We presented the following contributions, (1) first, the hardware architecture for controlling charge and discharge in federated power systems. (2) Second, a software layer for hardware control and service unit management. (3) Third, our experimental results for the accuracy of service unit availability report. (4) Finally, the conclusions and plans for future work, based from what has been achieved through this work.

This work brings us closer to a battery-free future for the Internet of Things, in line with the ubiquitous computing vision, where billions of smart devices are seamlessly interconnected, enhancing our lives. By overcoming the limitations imposed by batteries, IoT can expand to more application domains, solving even more challenging problems. With expanded IoT capabilities, the vision for the future of computing becomes clearer, and closer to actualization.

6. ACKNOWLEDGMENT

This work was supported by funding from NSF award number CNS-1644789.

7. REFERENCES

[1] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, March 2015.

[2] N. A. Bhatti and L. Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 209–220, April 2017.

[3] Josiah Hester and Jacob Sorber. The future of sensing is batteryless, intermittent, and awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, pages 21:1–21:6, New York, NY, USA, 2017. ACM.

[4] Paul Horowitz and Winfield Hill. *The Art of Electronics*. Cambridge University Press, 2015.

[5] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on rfid-scale devices. *SIGARCH Comput. Archit. News*, 39(1):159–170, March 2011.

[6] T. Zhu, A. Mohaisen, Yi Ping, and D. Towsley. Deos: Dynamic energy-oriented scheduling for sustainable wireless sensor networks. In *2012 Proceedings IEEE INFOCOM*, pages 2363–2371, March 2012.