Energy Attack Mitigation for Intermittent Sensor Devices

Hannah Emily Elisabeth Atmer

June 19, 2022

Contents

List of Figures

Abstract

Energy attacks against energy-harvesting intermittent devices let a malicious agent shut down the device without physical or network access. Energy attacks can prevent a sensor device from collecting important sensor data due to being powered down. We present an attack mitigation system for energy harvesting devices that improves quality of service during energy attacks. The attack mitigation system helps prevent or delay the device from running out of energy and powering down, which maximizes the device's ability to provide sensor data despite energy attacks. We show that the attack mitigation system is effective by simulating an energy harvesting device in a range of energy conditions.

Chapter 1

Introduction

1.1 Problem Statement

Energy-harvesting intermittent sensor devices are a sustainable IoT solution in which devices harvest energy from the environment, e.g., light, vibrations, or electromagnetic waves, and operate intermittently due to variable levels of available ambient energy [5][12]. Intermittent computing devices do not necessarily require batteries or a direct power supply, which means that the devices are economically and environmentally efficient, especially for large-scale deployments. This creates new possibilities for massive-scale deployments of sensors such as smart dust [20] and long-term wildlife monitoring [21]. Without batteries, the devices have longer lifetimes and therefore can even be embedded in concrete to measure the structural integrity of a building or placed in a human body for preventative medicine [1].

Battery-less energy harvesting devices store small amounts of energy in capacitors [4] but will shut down if their ability to harvest energy is disrupted for long enough. An attacker may force a sensor device to power off by disrupting the energy-harvesting device's ambient energy source. Loss of energy supply due to this type of attack may be challenging to differentiate from a natural decrease in available energy. In a yet unpublished work, an attack detection system that provides information about whether or not an attack is ongoing has been developed. Once an attack is known to be ongoing, the device can attempt to minimize the effect of the attack. The sensor device's primary goal is to maximize the quality of service (QoS) that it provides. Quality of service is the sensor device's overall performance, i.e. the number of data packets received by the user during a given time period. The lowest QoS occurs when the device "thrashes", i.e. runs out of energy and then repeatedly shuts down due to attempting to perform application tasks before the capacitor has stored enough energy. The device can adapt its energy consumption to delay or prevent shutting down due to running out of energy. The most energy-intensive operations performed by sensor devices are peripheral operations such as collecting sensor data or sending data via the network. Restricting the amount of energy spent on peripheral operations significantly decreases the amount of energy used by the device.

1.2 Approach and Method

Our approach is inspired by methods for energy-conserving used in Android mobile phones: the device enters a low power "sleep" state with no peripheral hardware activity while the screen is off but wakes up periodically to provide a minimum guaranteed quality of service, e.g., fetching app data via the network [6]. We developed a C library for intermittent computing applications that provides attack-mitigation wrapper functions for peripheral API calls. During an energy attack, the wrapper functions prevent the application from accessing peripheral hardware and the device is put to sleep during a variable percentage of the application's runtime, i.e., the device enters "sleep mode." Periodically, the wrapper functions restore the application's access to hardware peripherals and execute pending API calls, i.e., the device enters "wake mode."

The intermittent device can often harvest some energy despite the ongoing attack. This means that the device can avoid running out of energy by spending more time in sleep mode to ensure that the device harvests sufficient energy before attempting to perform application tasks. The application developer specifies the minimum required QoS for his/her application during energy attacks. The attack mitigation system will initially reduce application QoS to this minimum required QoS, and then cautiously attempt to improve the QoS using an multiplicative increase, additive decrease (MIAD) policy to optimise the QoS while being careful to avoid using up all the energy and causing the device to shut down. The attack detection system may specify that the attack is periodically recurring. In the event of a periodic attack, the expected length of the attack can be used by the MIAD algorithm to maximize QoS.

We simulate the effect of using the attack mitigation library on QoS using Cooja, Contiki OS's network simulator [2]. We run an application with and without the attack mitigation library functions wrapping peripheral API calls and compare the application's QoS during different energy levels. Specifically, we create a simulation of the device in Cooja, measure the amount of time the CPU and peripherals are active with and without attack mitigation, and use these measurements and information about the hardware's energy consumption in a Python simulation to generate results. The results show that our attack mitigation system helps the device survive for longer and increases the total number of packets sent at low harvestable energy levels.

1.3 Limitations

We assume that there is no way to directly measure the amount of energy stored in the capacitor. This means that the attack mitigation system cannot simply sleep when the amount of stored energy is low and wake when the amount of stored energy is high. Additionally, the device should not wirelessly alert the base station about the attack. This decision is based on the likelihood that sending an alert packet to the base station is likely to exhaust the device's stored energy.

1.4 Alternative Approaches

Instead of implementing the attack-mitigation system as an application, we could have implemented it as part of the operating system's scheduler. This would not work on systems that run without an OS. We also considered limiting only the most energy-intensive peripheral API calls. Instead we chose to limit all peripheral API calls since this approach restricts, rather than modifying, the application's behaviour. And instead of implementing the attack-mitigation system as function wrappers, we could have provided a function to be called at the beginning of each iteration of the application's main loop. But the application's main loop iteration can take an arbitrarily long time, and since peripheral API calls are the application's most energy-intensive operations, we instead place the attack-mitigation system at the application's interface with peripheral hardware. As a result, the attack mitigation system's primary purpose is to limit peripheral operations while an attack is ongoing.

1.5 Scientific Contributions

Our system is the first energy attack mitigation system for intermittent devices that do not necessarily have an operating system. We evaluate the MIAD algorithm for adjusting the amount of time that the attack mitigation system forces the system to stay in low power mode.

1.6 Report Structure

Chapter 2 is background information relevant to understanding this thesis. Chapter 3 is an overview of the attack mitigation system design, and Chapter 4 contains implementation details. Chapter 5 evaluates the AIMD algorithm's ability to save energy during energy attacks, and Chapter 6 is the conclusion and directions for future work.

Chapter 2

Background

In this chapter we provide an overview of energy harvesting devices, intermittent computing, energy attacks, and the simulation software Cooja. We also describe related works in throughput-optimising variable adjustment and energy-aware operations.

2.1 Energy Harvesting Devices

Energy Harvesting devices use ambient environmental energy to charge their batteries or capacitors. Types of harvestable ambient energy include solar, kinetic, and RF energy. Harvesting ambient energy is a zero-emissions alternative to wired power, and energy harvesting devices can be deployed on a massive scale since they use cheap hardware and do not require manual recharging.

Relying on harvested energy means that devices can operate without a connection to a man-made energy supply, without large batteries, and can be placed in locations that would be impossible for devices that require a connection to a power supply or battery replacement [1]. Energy harvesting devices are used in many domains, including as sensors for industrial monitoring, security systems, smart sports devices [32], structural integrity monitors [33], and as satellites.

Energy harvesting devices that do not have a battery are more environmentally friendly than devices with batteries or wired power: ambient energy is a valuable fossil-free energy source, conventional devices require more material and maintenance, and batteries are often not recycled properly [25]. Battery-less devices are also cheaper, smaller, and last longer since batteries are expensive, bulky, and usually the first component to wear out [24]. These intermittent IoT devices are self-sustaining [23] and can be deployed for many years without requiring maintenance.

2.1.1 Intermittent Computing

Intermittent devices are a subset of energy harvesting devices. Specifically, intermittent devices power on and off depending on the available ambient energy. Figure 2.1 shows

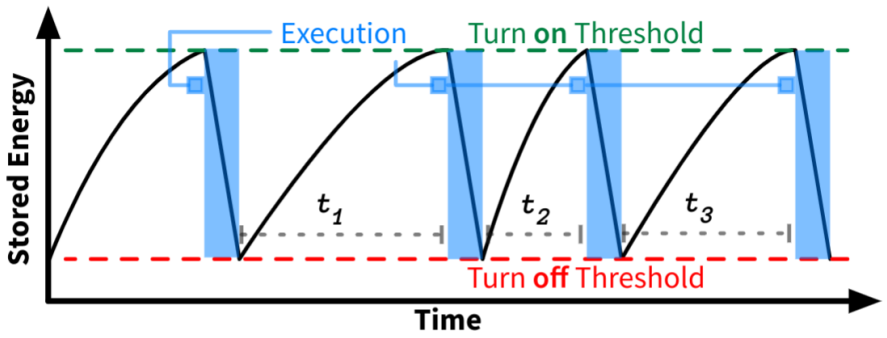


Figure 2.1: Intermittent Device Energy Levels over Time [37]

how device energy levels change during device operation. These devices often do not have batteries and instead rely on a capacitor to store a small amount of energy.

Intermittent devices often have explicit on/off logic to ensure that the device has enough time to harvest energy before attempting to run its application [26]. But, the programs that run on intermittent devices must execute correctly even when the device reboots [22]. Intermittent devices typically use checkpointing, i.e., they save execution state to non-volatile memory, to avoid losing state when the device powers off. Special attention must be paid to peripheral operations to avoid leaving the peripheral hardware in a state incompatible with the software [13][14]. For example, HarvOS implements code instrumentation for automatically making checkpoints if an energy-intensive operation is about to happen [16]. Approximate Intermittent Computing is another way for devices to cope with intermittent operation. Stateful computations are limited according to how much energy is available within a single power cycle, which allows for using the entire energy budget for useful computation instead of for saving state [27]. Approximate computation is only possible for specific applications such as image classification where intermediary results are viable outputs.

2.1.2 Energy Attacks

Energy harvesting intermittent devices are vulnerable to energy attacks, where an attacker prevents the device from harvesting energy. An attack mitigation system must be specialised for the type of attack [7][8] that is being protected against. In an energy attack, a malicious agent attempts to block the device's energy source to prevent it from performing its normal functions. Mitigating an energy attack involves maximising the quality of service provided by the device for the duration of the attack.

2.2 Cooja

We use Cooja, Contiki's network simulator, to develop and test the attack mitigation system. Cooja can execute the program binary, speed up execution to arbitrary speeds,

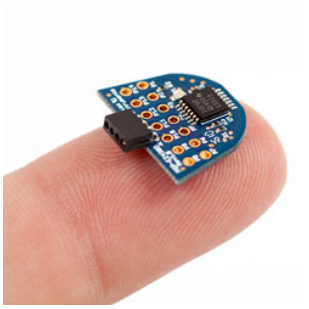


Figure 2.2: Tiny MSP430 MCU on human finger for scale [39]

show the output from the devices, and show when radio data is transmitted. Cooja runs a Contiki application with and without the attack mitigation system, and the amount of time the CPU and radio device were active is recorded.

2.3 Contiki OS

Contiki OS is a lightweight operating system for tiny networked sensors [11]. Contiki OS implements protothreads: stackless threads that minimize per-thread RAM usage [30]. Contiki OS also implements event timers that allow threads to generate timed events. The device can enter low power mode while all threads are waiting for a timer to expire [31].

2.4 Hardware

In this section we describe the hardware used in the simulation to measure the effectiveness of the attack mitigation system for a realistic intermittent device.

2.4.1 MSP430 MCU

The MSP430 is a microcontroller unit that is specialised to have low energy consumption. The MSP430 is often used for sensing devices in domains including grid infrastructure, factory automation, and medical devices [28]. Figure 2.2 shows a tiny MSP430 MCU on a PCB.

2.4.2 CC2420 RF Transceiver

The CC2420 is a single-chip, 2.4 GHz, low energy consumption radio frequency transceiver that is often used to transmit data from sensor devices [29]. Figure 2.3 shows the CC2420 Transceiver's circuitry.

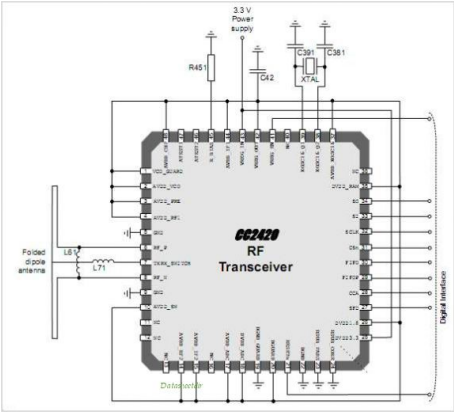


Figure 2.3: CC2420 Application Circuits [40]

Figure 2.4: Sensirion SHT31 Temperature Humidity sensor [41]

2.4.3 Sensirion SHT31 Temperature & Humidity Sensor

The Sensirion SHT31 Temperature and Humidity Sensor is a tiny sensing device that provides accurate temperature and humidity data despite minimal energy usage [36]. Figure 2.4 shows the SHT31 sensor hardware.

2.5 Related Work

TCP uses an additive increase, multiplicative decrease (AIMD) policy for controlling packet buffer congestion [18]. The additive increase of packet send rate is a cautious way to approach optimal QoS. The multiplicative decrease in packet send rate when congestion is detected enables an immediate back-off for quick adaptation. Figure 2.5 shows the ”sawtooth” pattern of TCP packet send rates that results from using the AIMD algorithm for congestion control. We expect a similar sawtooth pattern to emerge from using the MIAD algorithm to adjust the amount of time the attack mitigation system makes the application sleep.

Many intermittent computing systems implement energy-aware operations [9][10][15][16]

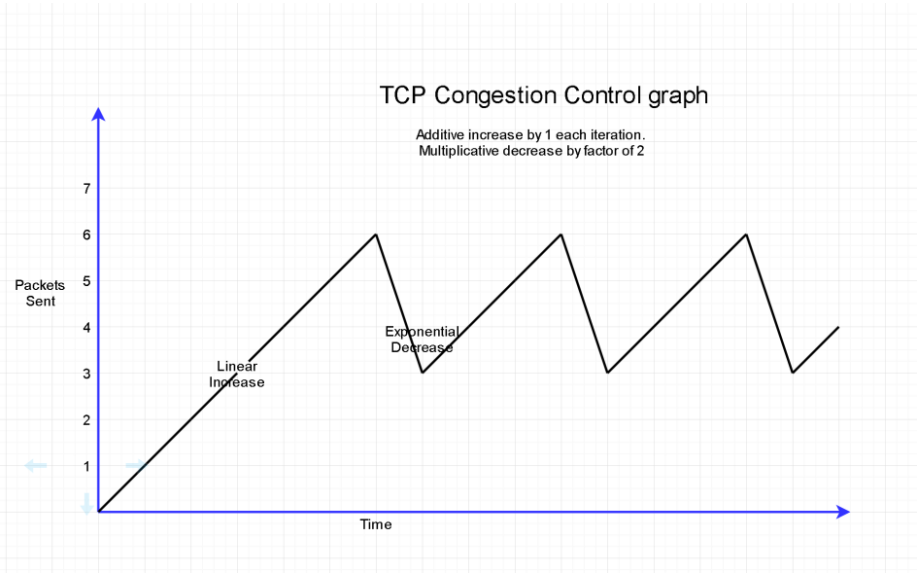


Figure 2.5: TCP AIMD Congestion Control

for adapting the system's behaviour to varying amounts of available energy for checkpointing, ensuring security, and scheduling. AsTAR [10] implements an energy-aware task scheduler to achieve optimal task scheduling rates despite varying amounts of available energy. AsTAR uses an AIMD or MIAD policy depending on whether the amount of available energy is increasing or decreasing. AsTAR is built for battery-free satellites. Unlike AsTAR, our solution can run as either as a Contiki process or on a system without an operating system. The GomX-3 Case [9] describes another scheduler-based solution for conserving energy that applies only to a specific battery-driven satellite application.

Chapter 3

Design and Implementation

3.1 Attack Mitigation

Once the device knows that an attack is ongoing, it can strategically alternate between a low power state and normal operations in order to conserve energy, increase the amount of time spent harvesting energy, and prevent the worst-case scenario of powering off for a long time during which the attacker can take advantage of the energy harvesting device being powered off to do malicious actions. The attack mitigation system knows that an attack is ongoing when a boolean variable is set by the attack detection system.

Our attack mitigation system puts the device in LPM for more time so that it can harvest more energy before attempting to use a peripheral device. This strategy is based on the assumption that the energy source is probably not entirely blocked by the malicious agent during the energy attack. Therefore it is still possible to harvest enough energy for the device to avoid running out of energy, as long as it spends more time waiting for its capacitor's energy buffer to fill up.

We implemented the attack mitigation system as a library that provides function wrappers for peripheral API calls. The wrapper functions accept a pointer to the peripheral API call, its arguments, and a pointer to a struct specifying whether or not an attack is ongoing and the application's minimum required QoS. In the case of an ongoing attack, the attack mitigation system puts the device to sleep for an amount of time determined by the minimum required QoS. Otherwise, the peripheral API function call executes immediately. Figure 3.1 shows a high-level overview of how the attack mitigation system is placed between the application and the peripherals.

In the simulation, the attack mitigation code runs immediately before the call to each peripheral API call, i.e. collecting sensor data or sending a packet, due to constraints of the simulation environment. We assume that we have information about whether or not an attack is ongoing in the form of a boolean. The device is put to sleep using a Contiki timer, which lets the device enter LPM until the timer expires. During this time, the device will still consume some energy because the operating system must repeatedly check if the time has expired. But, the device also harvests energy during LPM, so the attack mitigation will be effective as long as the amount of energy

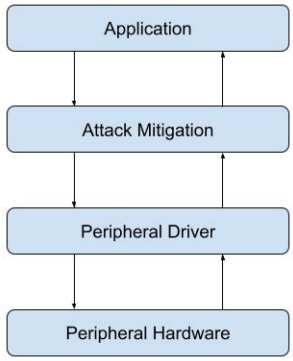


Figure 3.1: Design Overview

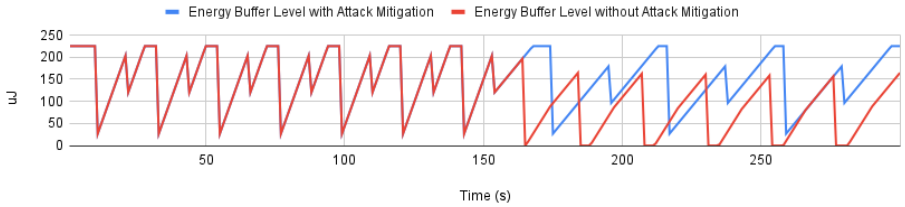


Figure 3.2: Example device energy buffer levels, with and without attack mitigation. The energy attack starts at time 150. The red line shows the device thrashing after the attack starts

harvested is more than the amount of energy used in LPM.

For example, consider an intermittent device that runs without shutting down as long as it harvests at least 17 microjoules of energy per second. Figure 3.2 shows this example device's energy buffer levels with and without the activation of the attack mitigation system soon after the attack starts. Under normal conditions, the device can harvest at least 17 microjoules of energy from the environment. Also, consider an adversary who performs an energy attack that limits available ambient energy so that the device can only harvest 10 microjoules of energy per second. The device may run out of energy, causing the device to "thrash" between being out of energy and attempting to run application tasks. During thrashing, the device may run out of energy during peripheral operations such that it makes no progress before shutting down. However, if the device uses the attack detection and mitigation system, it will continue to make progress even though it can only harvest 10 microjoules of energy per second. Once the attack detection system notices the ongoing energy attack, the attack mitigation system causes the device to spend more time in LPM harvesting energy and less time performing application tasks. The device makes slower progress but avoids thrashing, i.e. repeatedly running out of energy and failing to make progress.

3.2 Constraints

The attack detection system provides information about whether or not an attack is happening at any given time. The attack mitigation system relies on the attack detection system for information about any ongoing attack, and simply limits the application's use of energy-demanding peripherals such as radio and sensors. Additionally, we do not add energy-saving logic to how the application uses peripheral devices during the attack: we assume that the application developer has implemented the optimally energy-conserving version of the application. The attack mitigation system is fully separate from the application.

3.3 MIAD

We use MIAD to maximise quality of service during the energy attack. If information about the expected duration of the attack is provided, this information can also be used in deciding the amount of time spent in sleep mode. The minimal quality of service required by the application is provided by the application developer and determines the upper bound on the amount of time spent in sleep mode.

The amount of time to be spent in sleep mode during attacks is stored as the attack mitigation system's internal state. We assume that we know when the capacitor's energy level is critically low. Initially, the attack mitigation sleeps for an amount of time corresponding to the minimum quality of service required by the application. Every time the attack mitigation system runs, it checks if the energy level is critically low. If the energy level is critically low, time spent in sleep mode is doubled, as long as the amount of time in sleep mode does not cause the device to spend less time running the application than required by the minimum QoS. Otherwise, time spent in sleep mode is decreased by a constant δ. This causes the system to sleep for approximately the minimal amount of time required to mitigate the effects of the attack. It is important to set the value of delta to a small enough value so that the device will have a chance to detect that the energy level is critically low. If the value of delta is large, the device may run out of energy because the amount of time spent harvesting energy decreased too much.

Chapter 4

Evaluation

We evaluate the attack mitigation system's effectiveness in helping the intermittent device survive an energy attack. We measure the amount of time the device stays on during an energy attack, with and without the attack mitigation system. We also measure the amount of data sent from the device, so that QoS can be visualised in terms of device lifespan and number of sensor readings.

4.1 Experimental Setup

We measure the effectiveness of the attack mitigation system using Cooja. First, we implemented the attack mitigation system as a C library that provides function wrappers for peripheral API function calls. Then, we implemented a Contiki application that uses the function wrapper library, albeit modified to be compatible with Energest and Contiki processes.

Next, we use Cooja to measure how much energy is consumed by the application, with and without attack mitigation, specifically measuring how much energy the application uses in LPM and during tasks. Finally, we run a Python script that simulates the sensor device operating under different energy conditions to generate result graphs.

4.1.1 Demo Application

We built a simple demo application to measure the effects of the mitigation system. The demo application runs with and without the mitigation system to provide metrics about how long it takes before the device runs out of energy and the quality of service. The demo application runs as a Contiki process. The application waits 60 seconds to harvest energy, senses temperature and humidity, harvests energy for another minute, and sends the sensor reading to the base station via the RF transceiver.

The device's energy level is tracked in the software. We use the Contiki library Energest to estimate how many milliseconds the device's CPU, RF transceiver, and sensor were active, as well as how long the device spent in low power mode (LPM). The hardware datasheets were used to determine the energy consumption rates for the

Critical Energy Levels by Capacitor Size				
Capacitor Size (uF)	20	30	40	50
Maximum Energy (uJ)	90	135	180	225
Critical Energy (uJ)	14	21	29	36

Table 4.1: Critical Energy Level for each Capacitor Size

CPU, RF transceiver, and sensor. Using the amount of time spent in each activity and the energy consumption rate for each activity, we can simulate realistic device energy levels. We set the capacitor size and amount of energy that is still harvestable during the attack as variables in the simulation. When the demo application runs out of energy, it outputs the amount of time it was awake and the number of packets it sent.

4.1.2 Hardware Energy Consumption

We assume a constant 3V across the capacitor. The MSP430 MCU draws 400 microamps when the CPU is active, which equals 1.2 microjoules per millisecond [34]. We assume that we are using the p112 MSP430 MCU and that the device is operating at 25 degrees Celcius. We also assume that the MCU enters LPM3 when the application sleeps. The only way for the MCU to understand the energy content of the capacitor is to check the voltage across the capacitor through an ADC call. Anything below 2.1V is considered critical. Table 4.1 shows the critical energy level for each capacitor size.

The CC2420 RF Transceiver draws 8.5 milliamps, which equals 275 microjoules per millisecond [35]. We assume that the radio is configured with an output power of 25 decibel milliwatts. The transceiver uses approximately the same amount of energy regardless of whether it is transmitting or listening.

4.2 Data Set

A Python script generates the final result data. We measure the time it takes for the device to die, and the number of packets sent before dying, for each harvestable energy level between 0.0 and 20.0 microjoules, with a granularity of .01 microjoules. We compare the effects of the attack mitigation system given different QoS settings and different starting capacitor energy levels.

All measurements use a 225 microjoule capacitor since this is the minimal size required for the demo application. We set the value of delta to 1 unless otherwise specified.

4.3 Evaluation

The blue lines show performance with attack mitigation and the red lines show performance without attack mitigation.

Figures 4.1 and 4.3 show device lifetime and number of packets sent given starting capacitor levels of 100% and 25%. When the capacitor starts at 25% full, there is worse performance for harvestable energy levels that cause the capacitor to drain slowly to zero. At these energy levels, the device dies sooner because it does not have an energy reserve stored in the capacitor.

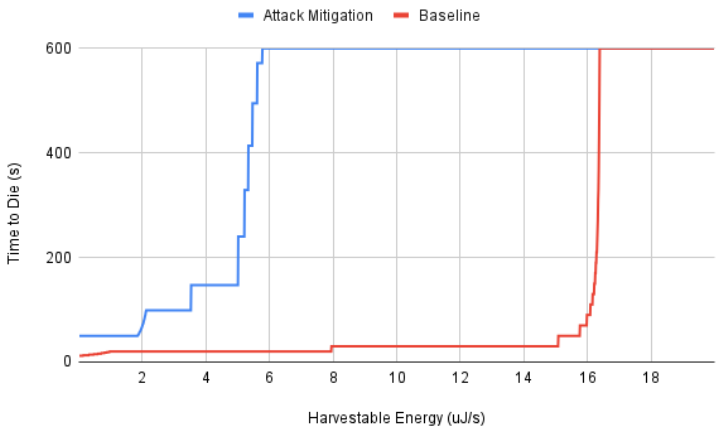
Figure 4.2 shows device lifetime and number of packets sent given different values of delta in the MIAD algorithm. We measure the effect of the MIAD adjustments to the percentage of time the device sleeps by running the simulation with the value of delta set to 0, 1, or 2. Adjusting the sleep time can only be done in very small steps because the device can only detect that energy levels are critical if the energy level is between 0 and the critical energy level during the MIAD adjustment. We see that when delta=0, i.e. no MIAD adjustment takes place, the device lives the longest but sends fewer packets. When delta=2, the device lives for a shorter time but sends more packets. This is an intuitive trade-off and the application developer can tweak the value of delta according to individual application QoS requirements.

Figures 4.4 and 4.5 show device lifetime and number of packets sent given a 10% QoS requirement and a 50% QoS requirement, to illustrate the effects of the minimum QoS setting. A required QoS level Q is set by the application developer and ensures that during a time period of T seconds, the application makes progress for at least Q * T seconds. The capacitor starts with a full energy buffer. The device lifetime and packets sent both increase as the minimum required quality of service decreases, which is intuitive since the lower the required quality of service, the more time the device spends in LPM. As the minimum quality of service increases, the lines for device lifetime and packets sent become more similar to the baseline. This makes sense because the higher the quality of service requirement, the less difference there will be between the system with attack mitigation and without.

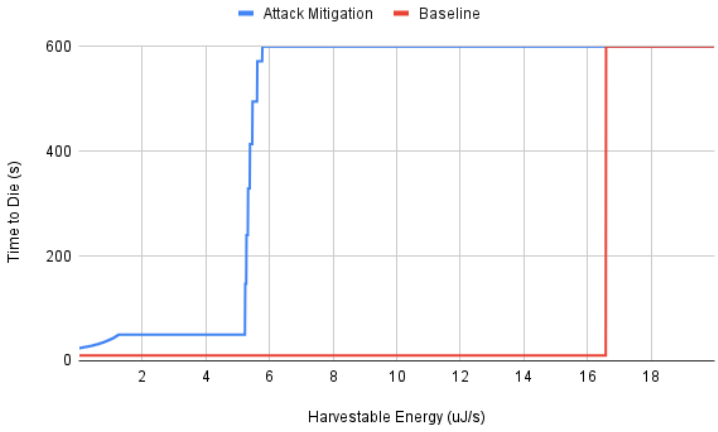
The attack mitigation system decreases the amount of energy required for the intermittent device to run for an hour without powering down. Without attack mitigation, the device runs for one hour when it can harvest more than 16.37 uJ per second. With attack mitigation and a 20% minimum quality of service, the device runs for an hour when it can harvest more than 5.77 uJ per second. When the attack mitigation system uses a 10% minimum QoS, the device runs for an hour when it can harvest more than 3.76 uJ per second.

If the amount of energy still harvestable during the attack is less than the amount of energy required for the device in LPM, then the device will inevitably shut down. If the amount of energy harvestable during the attack is greater than the amount of energy required for LPM, then the device can refill its energy buffer during sleep. Furthermore, it is clear from the results that as the minimum QoS requirement during energy attacks increases, the attack mitigation system is less able to conserve energy and extend the device lifetime.

The curve is smoother when the capacitor starts at 100% energy since if the capacitor has a full energy buffer when the attack starts, then there is improved performance in cases where the harvestable energy per second is close to the amount of energy expended by the device per second.

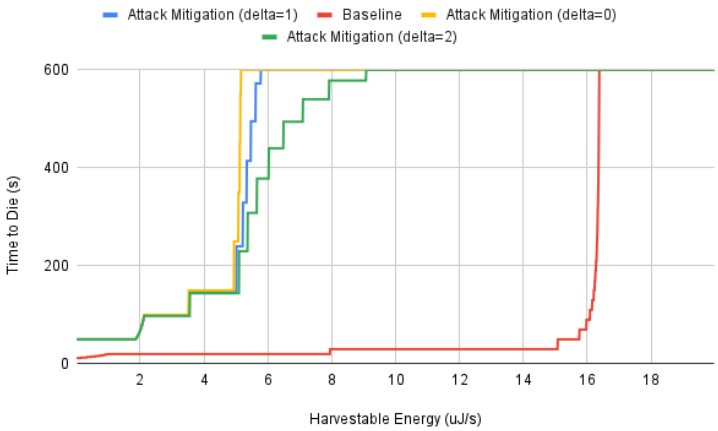


(a) Capacitor 100% Full when Attack Starts

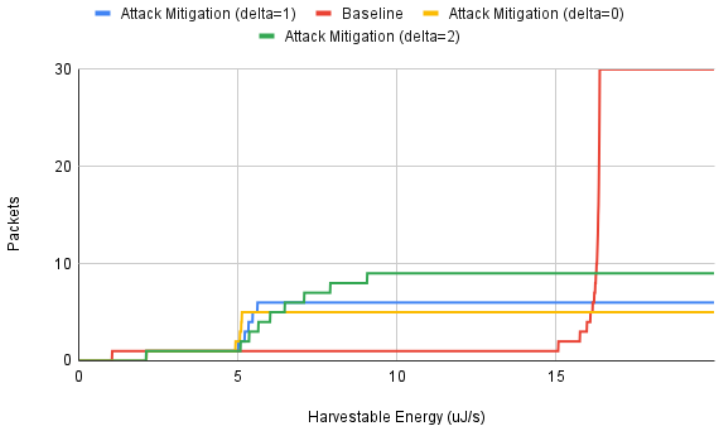


(b) Capacitor 25% Full when Attack Starts

Figure 4.1: 225 μJ Capacitor, 20% minimum QoS, time for device to die with full versus quarter full capacitor when attack starts

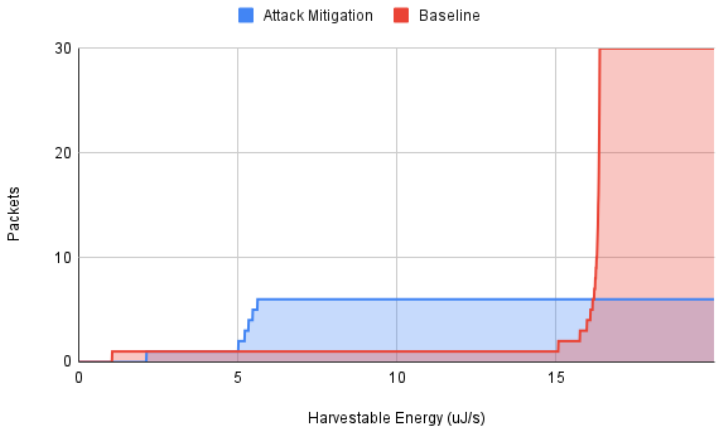


(a) Device lifespan for different values of delta

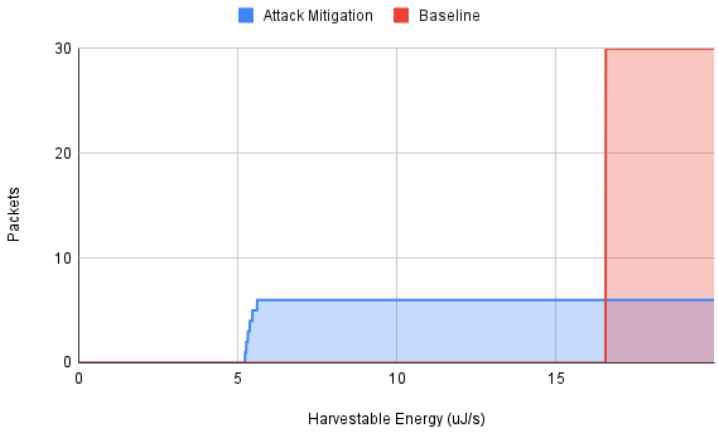


(b) Packets sent for different values of delta

Figure 4.2: 225 μJ Capacitor, 20% minimum QoS, time for device to die and packets sent for different values of delta (δ)

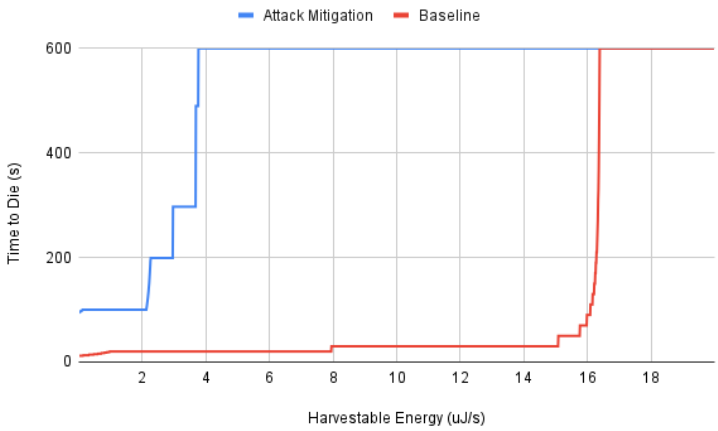


(a) Capacitor 100% Full when Attack Starts

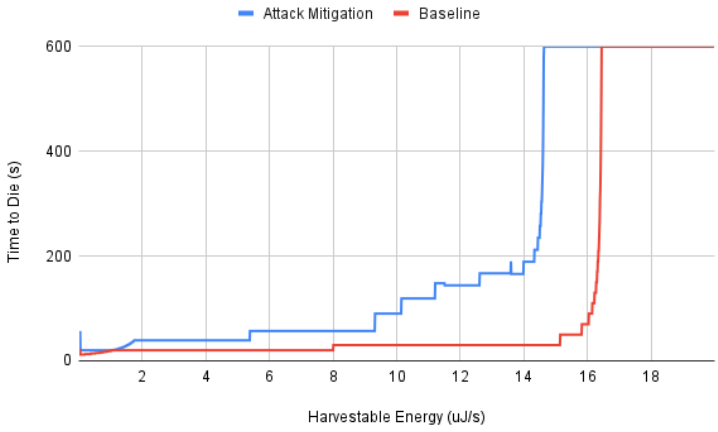


(b) Capacitor 25% Full when Attack Starts

Figure 4.3: 225 μJ Capacitor, 20% minimum QoS, packets sent with full versus quarter full capacitor when attack starts

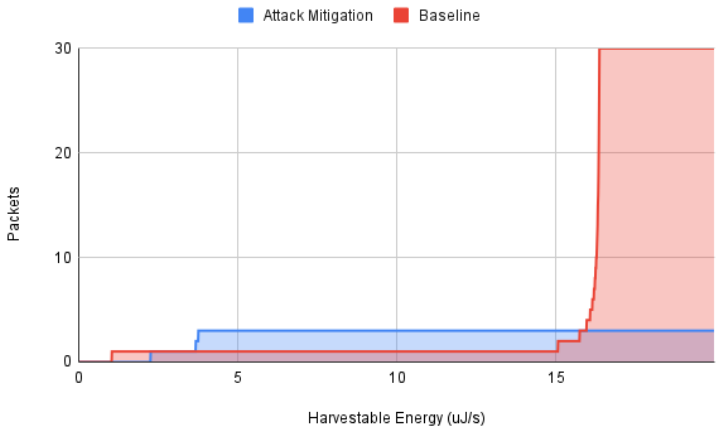


(a) 10% Minimum QoS

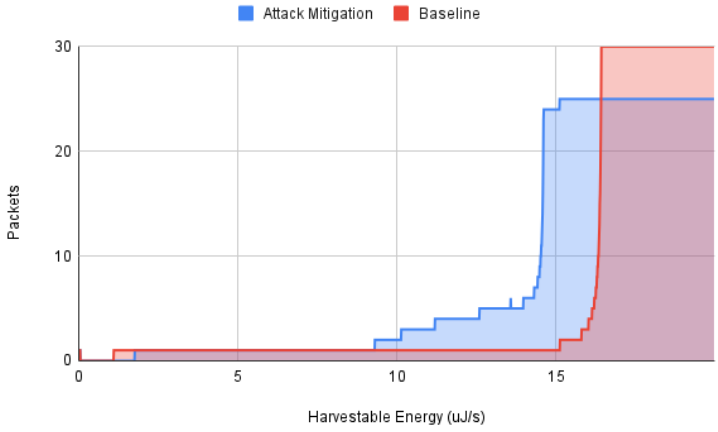


(b) 50% Minimum QoS

Figure 4.4: 225 μJ Capacitor, time for device to die with 10% minimum QoS versus 50% minimum QoS



(a) 10% Minimum QoS



(b) 50% Minimum QoS

Figure 4.5: 225 μJ Capacitor, packets sent with 10% minimum QoS versus 50% minimum QoS

Chapter 5

Conclusions and Future work

5.1 Conclusion

The attack mitigation system successfully decreases the device's energy consumption, delays device shutdown, and increases the number of packets sent during energy attacks. The sensor device's QoS is necessarily decreased in order to conserve energy, since collecting sensor data and sending it to the base station require a lot of energy. Decreasing quality of service in order to increase device lifespan is strategic since the most important goal during energy attacks is to prevent the attacker from disabling the sensor device for as long as possible. Modifying the value of delta for the MIAD adjustments to the amount of time the device sleeps is not useful for tweaking the behavior of the attack mitigation system.

5.2 Future Work

The attack detection system can provide additional information about the ongoing attack. Future work on this project will involve using this additional information to improve the attack mitigation system.

Chapter 6

Bibliography

[1] “Intermittent Computing to Replace Trillions of Batteries.” TU Delft, https://www.tudelft.nl/en/stories/articles/intermittent-computing-to-replace-trillions-of-batteries. Accessed 24 Sept. 2021.

[2] “An Introduction to Cooja - Contiki-Os/Contiki Wiki.” GitHub, https://github.com/contiki-os/contiki. Accessed 24 Sept. 2021.

[3] Branco, Adriano, et al. “Intermittent Asynchronous Peripheral Operations.” Proceedings of the 17th Conference on Embedded Networked Sensor Systems, ACM, 2019, pp. 55â67. DOI.org (Crossref), https://doi.org/10.1145/3356250.3360033.

[4] Sabovic, Adnan, et al. “Energy-Aware Sensing on Battery-Less LoRaWAN Devices with Energy Harvesting.” Electronics, vol. 9, no. 6, May 2020, p. 904. DOI.org (Crossref), https://doi.org/10.3390/electronics9060904.

[5] “Intermittent Computing.” Getting Started with Intermittent Computing, https://cmuabstract.github.io/intermittence_tutorial/. Accessed 24 Sept. 2021.

[6] “Platform Power Management.” Android Open Source Project, https://source.android.com/devices/tech/power/platform_mgmt. Accessed 24 Sept. 2021.

[7] “Comparing Master...Brute_v2 - Johwood/Linux.” GitHub, https://github.com/johwood/linux. Accessed 24 Sept. 2021.

[8] S. Ranjan, R. Swaminathan, M. Uysal and E. Knightly, “DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection,” Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, 2006, pp. 1-13, doi: 10.1109/INFOCOM.2006.127.

[9] Bisgaard, Morten Gerhardt, David Hermanns, Holger KrcÃ¡l, Jan Nies, Gilles Stenger, Marvin. (2018). “Battery-aware scheduling in low orbit: the GomXâ3 case.” Formal Aspects of Computing. 31. 10.1007/s00165-018-0458-2.

[10] Yang, Fan, et al. “AsTAR: Sustainable Battery Free Energy Harvesting for Heterogeneous Platforms and Dynamic Environments.” Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks, Junction Publishing, 2019, pp. 71â82.

[11] A. Dunkels, B. Gronvall and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455-462, doi: 10.1109/LCN.2004.38.

[12] Bhatti, Naveed Anwar, et al. “Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences.” ACM Transactions on Sensor Networks, vol. 12, no. 3, Aug. 2016, p. 24:1-24:40. August 2016, https://doi.org/10.1145/2915918.

[13] Asad, Hafiz Areeb, et al. “On Securing Persistent State in Intermittent Computing.” Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems, Association for Computing Machinery, 2020, pp. 8â14. ACM Digital Library, https://doi.org/10.1145/3417308.3430267.

[14] Ahmed, Saad, et al. “Fast and Energy-Efficient State Checkpointing for Intermittent Computing.” ACM Transactions on Embedded Computing Systems, vol. 19, no. 6, Sept. 2020, p. 45:1-45:27. November 2020, https://doi.org/10.1145/3391903.

[15] Hylamia, Abdullah, et al. “Security on Harvested Power.” Proceedings of the 11th ACM Conference on Security Privacy in Wireless and Mobile Networks, Association for Computing Machinery, 2018, pp. 296â98. ACM Digital Library, https://doi.org/10.1145/3212480.3226105.

[16] Bhatti, Naveed Anwar, and Luca Mottola. “HarvOS: Efficient Code Instrumentation for Transiently-Powered Embedded Sensing.” 2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2017, pp. 209â20.

[17] Zimmerling, Marco, et al. “Adaptive Real-Time Communication for Wireless Cyber-Physical Systems.” ACM Transactions on Cyber-Physical Systems, vol. 1, no. 2, Feb. 2017, pp. 1â29. DOI.org (Crossref), https://doi.org/10.1145/3012005.

[18] Blanton, Ethan, et al. TCP Congestion Control. Request for Comments, RFC 5681, Internet Engineering Task Force, Sept. 2009. IETF, https://datatracker.ietf.org/doc/rfc5681/.

[19] 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,” in IEEE Embedded Systems Letters, vol. 7, no. 1, pp. 15-18, March 2015, doi: 10.1109/LES.2014.2371494.

[20] Shwe, Yee Liang, Yung. (2010). Smart dust sensor network with piezoelectric energy harvesting. IJISTA. 9. 253-261. 10.1504/IJISTA.2010.036580.

[21] Loreti, Pierpaolo et al. âThe Design of an Energy Harvesting Wireless Sensor Node for Tracking Pink Iguanas.â Sensors (Basel, Switzerland) vol. 19,5 985. 26 Feb. 2019, doi:10.3390/s19050985

[22] Umesh, Sumanth, and Sparsh Mittal. âA Survey of Techniques for Intermittent Computing.â Journal of Systems Architecture, vol. 112, Jan. 2021, p. 101859. ScienceDirect, https://doi.org/10.1016/j.sysarc.2020.101859.

[23] Akin-Ponnle, Ajibike Eunice, and Nuno Borges Carvalho. âEnergy Harvesting Mechanisms in a Smart CityâA Review.â Smart Cities, vol. 4, no. 2, Apr. 2021, pp. 476â98. DOI.org (Crossref), https://doi.org/10.3390/smartcities4020025.

[24] Hester, Josiah. âIntermittent Computing: Batteries Not Included.â KA MOAMOA, 27 Apr. 2016, http://kamoamoa.eecs.northwestern.edu/project/intermittent-computing/.

[25] Kristina Dervojeda, Diederik Verzijl, Elco Rouwmaat, Laurent Probst, Laurent Frideres. "Energy harvesting" Clean Technologies, Jun. 2014, https://ec.europa.eu/docsroom/documents/13396/attachments/5/translations/en/renditions/native

[26] https://brandonlucia.com/pubs/snapl2017_preprint.pdf

[27] Luca's not-yet published IPSN paper The Case for Approximate Intermittent Computing

[28] https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-microcontrollers/applications.html

[29] https://www.ti.com/product/CC2420features

[30] http://contiki.sourceforge.net/docs/2.6/a01802.html_details

[31] http://contiki.sourceforge.net/docs/2.6/a01667.html_details

[32] âHomeâ. Energy Harvesting Solutions, https://enervibe.co/. Accessed 20 Mar. 2022.

[33] Sutton, Felix Buchli, Bernhard Beutel, Jan Thiele, Lothar. (2015). Zippy: On-Demand Network Flooding. 45-58. 10.1145/2809695.2809705.

[34] MSP430 Pdf, MSP430 Description, MSP430 Datasheet, MSP430 View::: ALLDATASHEET::: https://pdf1.alldatasheet.com/datasheet-pdf/view/27250/TI/MSP430.html. Accessed 20 Mar. 2022.

[35] 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, https://www.ti.com/lit/ds/symlink/cc2420.pdf. Accessed 20 Mar. 2022.

[36] Datasheet SHT3x-ARP, https://sensirion.com/media/documents/EA647515/61641D0C/Sensirion_Humidity_Sensors_SHT3x_Datasheet_analog.pdf. Accessed 20 Mar. 2022.

[37] Christian Rohner, 02/02/2022, Lecture 3 - Intermittent Computing, Uppsala University

[38] âDo You Know about the AIMD Method in TCP Congestion Control?â Learn Steps, 22 Feb. 2020, https://www.learnsteps.com/do-you-know-about-the-aimd-method-in-tcp-congestion-control/.

[39] https://martybugs.net/electronics/msp430/images/msp430_A20462_250.jpg. Accessed 20 Mar. 2022.

[40] CC2420 Datasheet, Pinout ,Application Circuits Single-Chip 2.4 GHz IEEE 802.15.4 Compliant And ZigBee(TM) Ready RF Transceiver. http://www.datasheetdir.com/CC2420+ZigBee. Accessed 20 Mar. 2022.

[41] SHT31-DIS-B Digital Humidity and Temperature Sensor. https://sensirion.com/products/catalog/SHT31-DIS-B/. Accessed 20 Mar. 2022.