Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer, Communication and Electronic Engineering

Final dissertation

A Tiny Network Stack for Battery-free Communication

Supervisors

Prof. Kasim Sinan Yildirim

Prof. Davide Brunelli

Alessandro Torrisi

Candidate

Lorenzo Canciani

Academic year 2021/2022

Acknowledgments

I would like to thank my supervisors Prof. Kasim Sinan Yildirim, Prof. Davide Brunelli and Alessandro Torrisi for their help and advice with this thesis.
I would also like to thank my family that always supported me.

Contents

Abstract

The use of battery-free systems is a revolutionary frontier for the Internet of Things; in fact, it allows network nodes to be unconstrained by the use of batteries.
To achieve this, battery-free systems use ambient energy to charge a capacitor and use it as a power source. Because of the unpredictability of environmental energy sources, the node is often subject to power failure. Communication then becomes intermittent and data loss particularly frequent.
To solve this problem, we implemented a tiny communication stack that implements non-volatile memory management for data storage and the TRAP protocol for energy awareness between nodes.

1 Introduction

This thesis contains all the work done to implement a tiny network stack for battery-free communication.
The microcontroller family used in this project is MSP430. These microcontrollers are built on a 16-bit processor that provides low power consumption.
Before looking in detail at what has been implemented, it is worthwhile to provide some background information to introduce the current situation of battery-free communication.

1.1 Battery-Free Devices

The use of distributed sensors in the environment is a very interesting frontier for IoT applications; however, it is impossible to think of being able to wire all connections.
For this reason Wireless Sensor Networks (WSNs), distributed networks of sensors that communicate wirelessly, are being introduced.
In recent years processors are becoming less energivorous and batteries more durable but thinking about a sensor network that relies on batteries does not seem the best solution. Batteries need to be replaced periodically [8], and this is complicated if the nodes are in environments that are difficult or unsafe to reach.
To make the nodes energy-autonomous, researchers are working on systems that harvest energy from the environment in a capacitor to use as a power source; some examples of energy are solar [12], RF [3] , kinetic [15], thermal [9] and some bacteria species [11].
In this way, there is no need to rely on batteries, and the system can work for its entire lifetime without [6] or with minimal maintenance.
The energy harvested by these devices is employed to produce and get data from external sources, process it, and send it to other nodes in the network.
The operation of these devices is closely related to environmental conditions, and it is not guaranteed to always have sufficient energy to perform all the intended tasks.
Because of this low reliability of the energy level for task completion, nodes are prone to power failures and consequently work intermittently.
The frequency of these outages depends on many factors, such as capacitor size, available energy in the environment, and node consumption.

1.2 Problem Statement

Stored energy is precious and should not be wasted. If a node transmits a packet and the receiving side does not have enough energy to finish receiving, the packet is lost, and there is a power failure on the node (Figure 1.1). Power failures can have many other causes such as producing a data item without first verifying the availability of energy for proper production or suddenly consuming more energy than expected.
Anyway, in the event of a power failure, the node loses all the data it had produced and must consume more energy to produce the data again. At the same time, data received from other nodes in the network are lost. The latter scenario presents a considerable waste of energy; in fact, the energy used to produce, send and receive the packet is wasted.
To avoid these problems we need to:

1. Save data in non-volatile memory so that nothing is lost in case of power failure
2. Resume data management to just before power failure

3. Recognize expired data and delete them

These problems must be solved in a way that provides the application layer with the methods of producing/sending and receiving data while hiding their complexity from the end user.

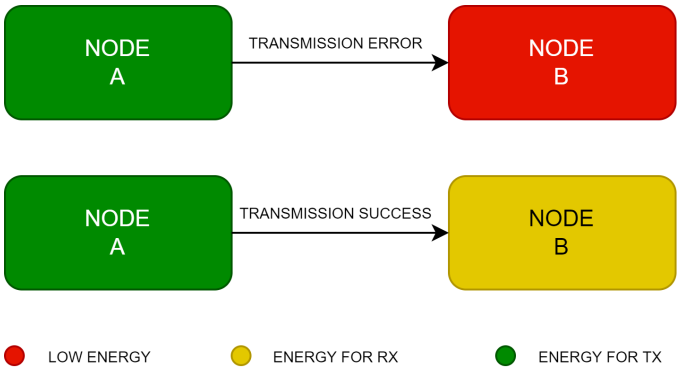

Figure 1.1: Transmission examples

1.3 Contributions

There are methodologies to limit power failures, such as the TRAP protocol [14], but it is impossible to be certain that they will not happen. Therefore, the contributions of this thesis are to present a Communication Stack for battery-free communication that:

1. Implements the TRAP protocol [14] to be aware of the energy level of neighboring nodes
2. Saves data received from the Application Layer in non-volatile memory
3. Saves data received from the Physical Layer in non-volatile memory
4. Recovers data from memory in cases of power failure
5. Delete expired data to avoid unnecessary sending or processing
6. Send and receive data to and from Application Layer
7. Send and receive data to and from Physical Layer

In addition to the problems that the stack aims to solve, we want a system that is modular and easily scalable.
In particular, the stack must be easily usable by the Application Layer and easily usable by the Physical Layer even when there are a large number of nodes in the network.
The Physical Layer must also not be binding on the stack, which must be able to work with different communication technologies, such as RF Backscatter communication [13] or VLC communication [2], by changing only the Physical Layer.
For these reasons, the nonfunctional requirements of the stack are:

- Scalability
- Modularity

2 System Design

This thesis work aims to implement a system that not only allows the least amount of data to be lost in case of power failure but also makes this management easy at the application level while hiding all the complexity of the system.
The communication stack is designed to reduce data loss issues during intermittent operation and make the best use of energy harvested from the surrounding environment.

2.1 Architecture Diagram

The proposed system consists of four parts:

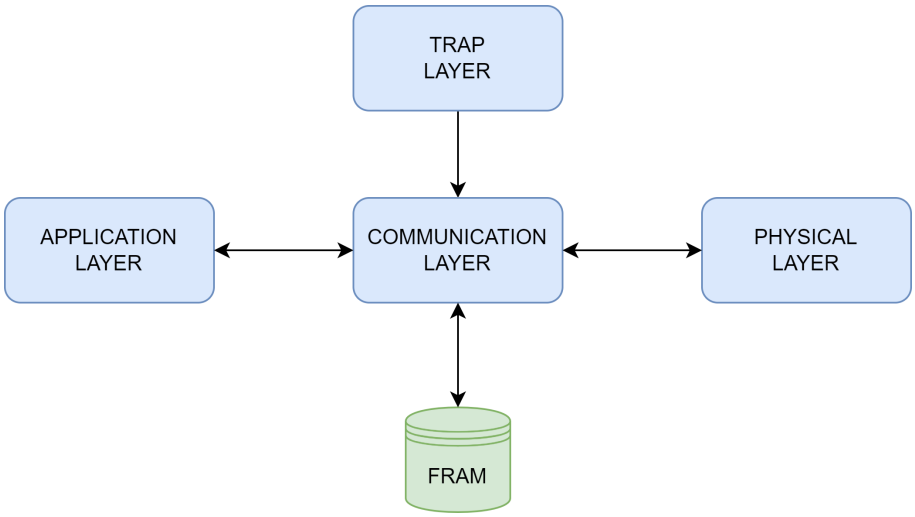

Figure 2.1: System architecture diagram: Application Layer, TRAP Layer, Communication Layer and Physical Layer

2.1.1 Application Layer

This section contains the abstractions that allow the programmer to develop an application using the implemented network stack.
The functions exposed by the Communication Layer allow high-level data sending and receiving. All memory management, transmitting, and receiving are executed automatically by the stack.
An example of an Application Layer, in pseudo code, might be as described below:

```
....
while (1)
{
    value = collectDataFromSensor();
    destinationNode = 1;
    dataProduced(value, destinationNode);
    dataSend();
}
....
```

2.1.2 TRAP Layer

The Trap Layer is the layer that implements the TRAP protocol.
It is based on the use of timers for repetitive burst sending, OOK modulation, and sending node identification.
The bursts contain encoding of the energy level of the sending node. Each neighboring node in the network sends its energy level at fixed intervals.
Recognition of the sending node is accomplished by an analysis of the burst sending frequency.

2.1.3 Communication Layer

Communication Layer is the part of the system responsible for sending and receiving data from the Physical Layer and the Application Layer.
The Communication Layer does much more than what has just been explained: it is responsible for managing the data within non-volatile memory (FRAM [1]) so it is the heart of the system.
FRAM management consists of saving and retrieving data; this mechanism helps limit packet losses during power failures.

2.1.4 Physical Layer

The Physical Layer is the layer responsible for actually sending and receiving packets.
It is easily replaceable and allows the use of different transmission technologies such as backscatter communication or visible-light communication.
In this implementation, the choice fell on the use of a UART communication.
A whole chapter will not be devoted to explaining the layer as it is of little significance to the implementation. It will be mentioned in the evaluation section to give context to the tests performed.

3 System Implementation

The purpose of this chapter is to show how the complexity of the system was hidden from the Application Layer through the use of methods exposed by the TRAP and Communication Layers.

3.1 TRAP Layer

The TRAP Layer is the layer that implements the TRAP protocol.
The purpose of this layer is to allow reliable communication between nodes by reducing power failure allowing transmission only under certain conditions. Specifically, communication can only occur if the sending node and the receiving node have sufficient energy levels to send and receive the packet [14].
As the protocol is designed, this layer is based on the repeated sending at fixed intervals in time of a series of pulses that encode the node's energy level. On the other hand, the layer must receive the energy levels of neighboring nodes within the network and correctly associate them, through sending frequency recognition, with the transmitting node. For this reason, the basic implementation idea exploits timers on the MSP430 board to perform repetitive burst sending (series of pulses, varying in length depending on the energy accumulated by the node) and externals GPIO interrupts generation to handle incoming bursts.
To correctly associate the incoming burst with the sending node, the adopted solution is to have a timer count from the beginning of the reception and stop it at the end of the reception, then calculate the sending frequency.
Based on the energy level of the node and the energy level received from other nodes, the layer decides whether or not transmission can be made to and from the node.

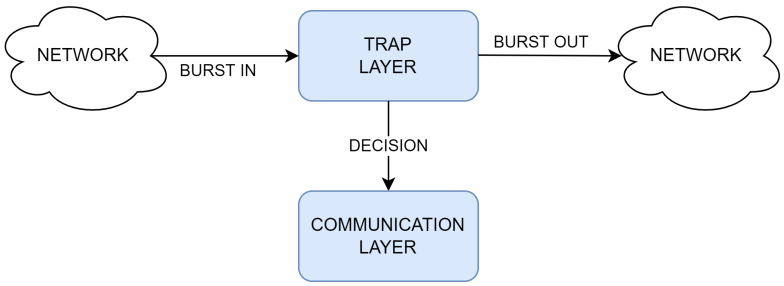

Figure 3.1: TRAP Layer representation

This Layer exposes a single function to check whether the energy level of the sending node and the receiving node is sufficient to complete the transmission. The return value of the method is 1 if sending is possible, and 0 otherwise.

3.2 Communication Layer

This section of the system contains the infrastructure needed to limit the loss of data and consequently the waste of energy following a power failure. More specifically: saving data to non-volatile memory, restoring data from non-volatile memory, sending data to the Physical Layer, sending data to the Application Layer, and checking data validity in case of expired packets.
Control over packet expiration is included within the sending of data to the Physical Layer

and the Application Layer. For this reason, it will be discussed in those sections and not in a dedicated part.
Before analyzing the implementation choices in detail, it is convenient to indicate the package structure used by the layer for processing.

3.2.1 Packet Structure

The package is defined as follows:

NODEID	DATA0	DATA1	DATA2	DATA3	CRC0	CRC1	TIMESTAMP

Table 3.1: Package structure, 64bit

The packet consists of 8 byte-sized fields. Specifically, the first field indicates the producing node, the next four the data produced, then we find two CRC fields used to detect any errors in transmission, while the last field indicates the production timestamp of the packet.
It is important to note that the fields are byte-sized, this allows atomic operations in memory.

3.2.2 Data Saving From Application Layer (Data saving TX)

The Application Layer calls the producedData() method passing as arguments four bytes of data and one byte indicating the message recipient node identifier.
After the data reception, the Communication Layer begins saving the data in non-volatile memory.
Data saving takes place in structures containing the message fields.

Listing 3.1: Structure containing the package fields and two supporting fields for managing the package in memory

```
typedef struct storedData
{
    unsigned char nodeNumber, data0, data1, data2, data3, CRC0, CRC1, timeStamp,
        saved, nodeRX;
} storedData;
```

These structures are arranged in a circular buffer (Buffer TX). For this purpose, it is important to keep track of the last location written to the buffer in case of power failure. This is needed to avoid inadvertent overwriting of data. To do this it is necessary to save in non-volatile memory a field that will serve as a pointer.
The four data passed to the function are saved directly, byte by byte, in non-volatile memory along with the producer node value (the latter value is already known to the Communication Layer and does not need to be passed by the Application Layer).
Next, the Communication Layer proceeds to calculate the CRC16 value of the four bytes of data passed by the Application Layer. The resulting two bytes are saved in non-volatile memory too.
At this point, the packet time stamp, and the recipient node identifier are added.
The packet will be confirmed as saved by setting a control byte to 1 and incrementing the buffer pointer.
Finite state machine of data saving from Application Layer is presented in Figure 3.2.
If no power failure is encountered during saving, the package is saved correctly and ready to be sent when requested by the application layer.
On the contrary, any power failure before the pointer increment would render the data unusable and would be overwritten at the next save request by the Application Layer.

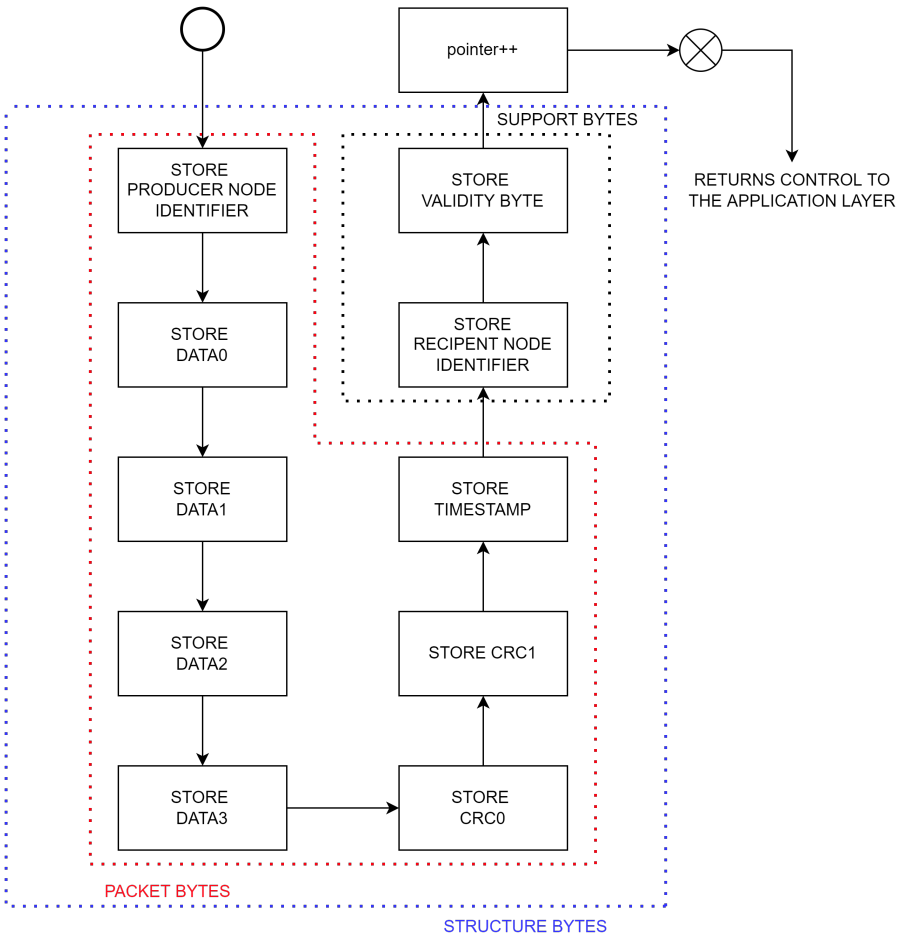

Figure 3.2: Finite state machine related to package building and saving (pointer: writePointer)

3.2.3 Data Sending To Physical Layer (Data TX)

To send the data produced, the Application Layer must call the dataSend() method without passing arguments.
When this method is invoked, the Communication Layer chooses the packet stored in non-volatile memory to send and forwards it to the Physical Layer.
The selection of the packet to be sent is the most relevant section of this method. The packets to be sent are stored in a circular buffer, so you have to keep track of the packet by a pointer stored in non-volatile memory. The logic is the same as shown in the data saving. The pointer is incremented only when the operation is completed and the validity byte is reset to 0. This is important in case of power failure, if the control byte was reset before the sending was completed, the packet would be marked as sent when in fact it was not.
The data is considered valid only if the control byte (set during packet saving) is set to 1.
Choosing the packet to be sent only with the pointer, however, is not the best solution. In fact, before each send, the canSendTRAP() method of the TRAP layer is called to check whether it can be transmitted to the recipient node.
If a recipient node A shuts down and the buffer contains messages for nodes A, B and C in that order, the insufficient energy level of A blocks the messages for B and C. In order to avoid that, the node sends the next packet in the buffer (i.e. the one for B), if possible, otherwise it tries with the next again and so on.
The solution adopted involves a priority check on the packet indicated by the pointer. If it cannot be sent, checks are made on subsequent packets until a sendable packet is reached or the entire buffer is scrolled (Figure 3.3).

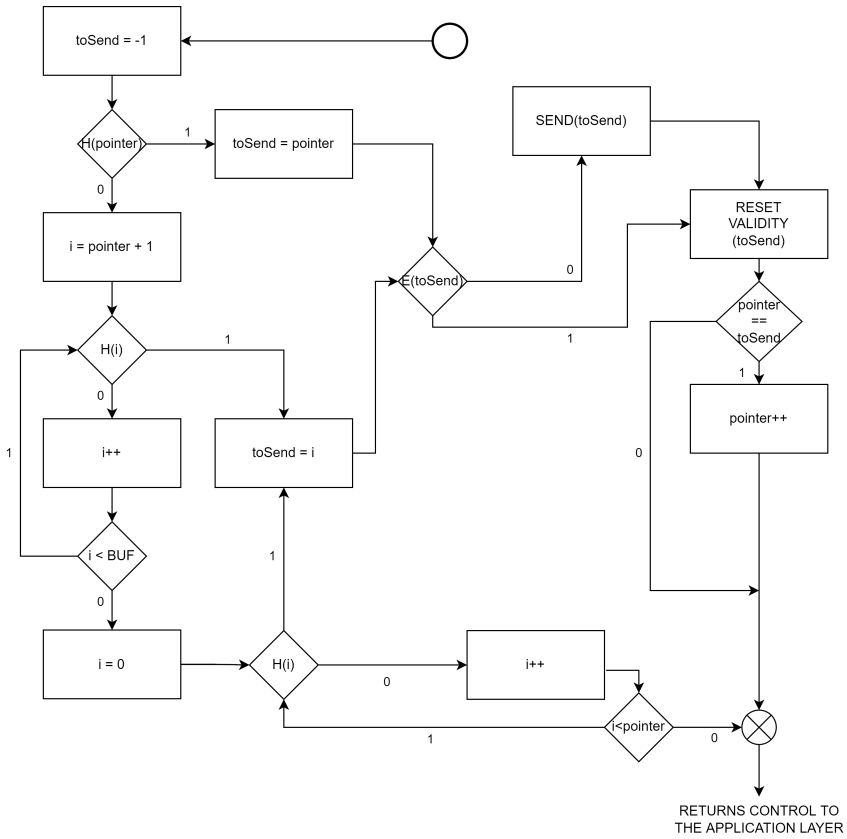

Figure 3.3: Finite state machine related to package selection and sending (H(x): (checkValidity(x) && canSendTRAP(x)), E(x): expiration(x), pointer: sendPointer, BUF: TX Buffer size)

3.2.4 Data Saving From Physical Layer (Data saving RX)

Saving data received from the Physical Layer is done in much the same way as presented for saving data from the Application Layer. Again, data saving occurs in structures containing message fields arranged within a circular buffer (Buffer RX). This requires a pointer to keep track of the last position written.
The Physical Layer notifies by triggering an interrupt the Communication Layer at the end of data reception.
The Communication Layer checks whether the received packet consists of 8 bytes. If the received packet is complete, the Communication Layer checks for errors via a CRC16 module. If no errors are detected the packet validity byte is set to 1 and the receive buffer pointer is incremented. The pointer, as explained earlier, is not incremented before packet acknowledgment because in case of power failure it could lead to marking inconsistent packets as good.

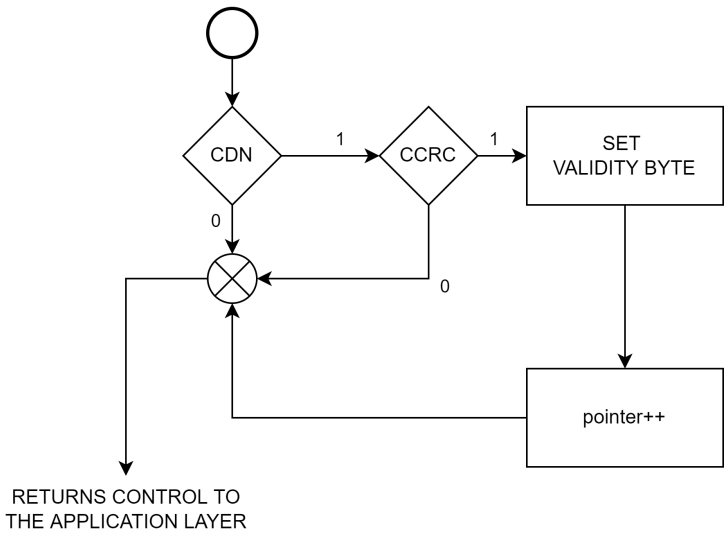

Figure 3.4: Finite state machine related to Data saving RX (CDN: checkDataNumber(), CCRC: checkCRC())

3.2.5 Data Sending To Application Layer (Data RX)

The forwarding of packets from the Communication Layer to the Application Layer is done by the call of the getData() function.
The Application Layer receives a structure containing the first available packet saved in memory if present, a packet initialized to zero otherwise.
When the getData() function is called, the Communication Layer scrolls through all the packets saved in the RX buffer to delete the expired ones.
Thereafter, the packet choice is made similarly to what was seen for sending to the Physical Layer. The difference, concerning what was presented earlier, is related to the TRAP check that does not have to be performed.
For the location of the packet to be transmitted within the buffer, a pointer is saved in non-volatile memory. This pointer will be incremented only after the validity byte of the structure containing the data is reset.

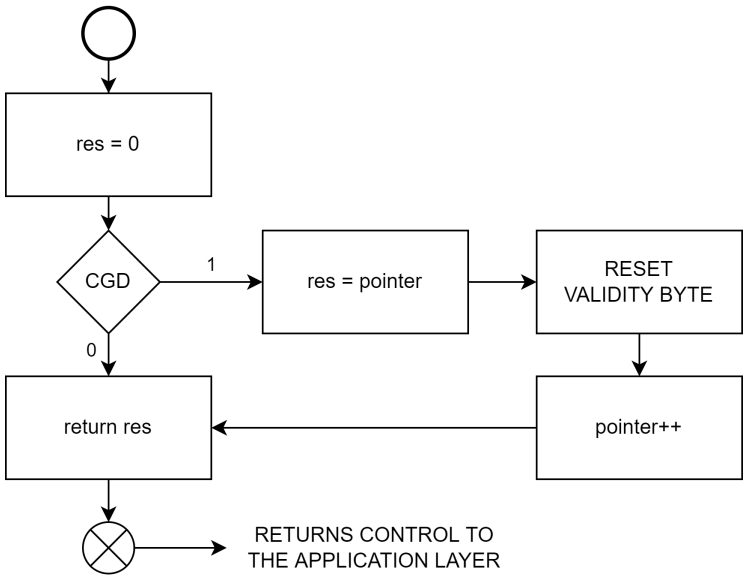

Figure 3.5: Finite state machine related to getData() function (CGD: canGetData())

3.3 Power Failure Resistance

Since power failure resilience is the novelty of this thesis, it is worth spending a few words explaining what happens in case of power failure during normal system operation such as during sending data from the Application Layer to the Communication Layer, forwarding data from the Communication Layer to the Physical Layer for sending to the network, receiving data from the network to the Physical Layer and forwarding to Communication Layer, and during data sending from the Communication Layer to the Application Layer.

3.3.1 From The Application Layer To The Communication Layer

In this case, we have several possibilities:

1. Power failure occurs before the packet is acknowledged through the validity byte set
2. Power failure occurs immediately after the packet validity byte set but before the pointer increment that signals the location to be written to the buffer
3. Power failure occurs after the packet validity byte set and after the pointer increment that signals the location to be written to the buffer (just before the return from the function to the Application Layer)

In the first scenario, the partially saved packet will be overwritten at the next data pass by the Application Layer. This is the worst situation because the data cannot be sent. The energy used to produce it has been wasted.

In the second scenario, the situation is slightly better than shown above. The packet will, again, be overwritten on the next pass of data by the Application Layer. However, having the validity byte set to 1 makes it possible to send it if it is not overwritten first. This operation could be useful when the Application Layer produces less data than can be sent as it would still allow the packet to be sent. The power produced to produce the data is considered lost only if the packet is overwritten.

In the third situation, the power failure is of no consequence since the packet is marked as saved correctly and thus can be sent following the Application Layer request. Similarly, the pointer to the buffer location is incremented, consequently, a new data save call by the Application Layer would not overwrite the packet since the next location in the buffer would be written.

3.3.2 From The Communication Layer To the Physical Layer

A power failure during data transfer from the Communication Layer to the Physical Layer is a remote possibility; in fact, this transfer is either authorized or denied by the TRAP protocol.
However, developing a system that relies only on the TRAP protocol and has no persistent management of packets to be sent does not seem like a good solution.
Therefore, we can consider and show how packets are handled in case of power failure during sending from the Communication Layer to the Physical Layer. It should be remembered that sending data from the Communication Layer to the Physical Layer occurs only when the TRAP protocol responds affirmatively to the possibility of sending the packet to a particular node. Data sent to the Physical Layer is immediately forwarded to the recipient node.
In this case, we have several possibilities:

1. Power failure occurs before the packet validity byte is reset
2. Power failure occurs after the reset of the packet validity byte

In the first case, the packet is not marked as sent since the validity byte is not changed. The packet will be resent when possible. The power used to send part of the packet is lost.

In the second case, the validity byte is reset, and then a power failure occurs. The way the stack is implemented, if the validity byte is set to zero, the packet can no longer be sent and will be overwritten in subsequent saves by the Application Layer. However, resetting the byte to zero implies sending the data completely. In this situation, the pointer relative to the location in the buffer of the packet to be sent is not incremented; again, this is not a problem because the Communication Layer performs checks on the validity of the packet before forwarding it to the Physical Layer.

3.3.3 From The Network To The Physical Layer

A possible power failure during the reception of data from the Physical Layer to the Communication Layer is a situation that would be unlikely to occur; in fact, even this transmission takes place under the control of the TRAP protocol.
If this happens, however, the system is designed to discard the packet if it has not been marked confirmed with the validity byte set. Specifically, the Physical Layer starts a timer upon receipt of the first piece of data. This timer is restarted with each new reception.
At the moment when the timer generates an interrupt, and thus no data has been received for a given time interval, the Communication Layer is called to perform a check on the amount of data that has arrived. This check is used to verify that all 64 bits of the packet have been received.
If a power failure occurs before the packet validity byte is set by the Communication Layer, the data is discarded and is not sent, when requested, to the Application Layer. In this case, the energy expended in receiving and transmitting the packet is wasted.

3.3.4 From The Communication Layer To The Application Layer

To receive the packet from the Communication Layer, the Application Layer must call a function that returns a packet saved in the RX buffer of the Communication Layer.
If the power failure occurs after the Communication Layer resets the packet validity byte, the packet is lost. This happens in the case where the power failure occurs during the return from the function to the Application Layer.
In the case, on the other hand, where the power failure occurs before the validity byte reset, the packet can be requested again at the Communication Layer since it will still be valid.

4 Evaluation

The system presented exploits non-volatile memory to save data to be retained in the event of a power failure. The novelty of this implementation is the resistance to data loss in case of power failure added to the implementation of the TRAP protocol to reduce this type of events. To better highlight this progress, this chapter mainly presents some emulations performed with the presented stack, compared with emulations without the provided implementation.

4.1 Technical details

The size of the implementation varies according to the number of nodes within the network, the size of the RX and TX buffers saved in FRAM, and the optimization required of the compiler. For compiling optimization the highest level (Whole Program Optimization) is used, this also helps to reduce the power consumption of the system [4].
To provide some data, with 2 nodes in the network and the receive and send buffer (buffer RX and buffer TX) of size 3, the memory overhead is 5223 bytes.
The figure below shows the memory allocation provided by the Code Composer Studio IDE.

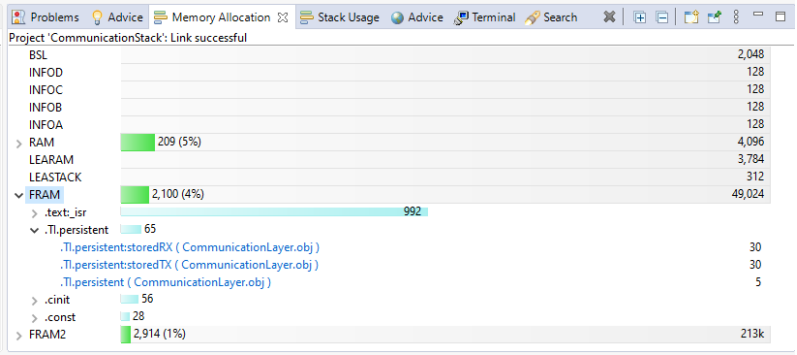

Figure 4.1: Memory overhead for network with 2 nodes and buffer size 3

If the network nodes are increased to 16 and the size of both buffers to 32, the overhead will be 5307 bytes.
It is then possible to estimate the memory overhead of the implementation at about 5.5KB.

4.2 Emulation

The setup used for the emulations involves communication between two nodes; consequently, two TI MSP430 boards were used:

1. TI MSP430FR5994
2. TI MSP430FR5969

For communication regarding the energy level (TRAP Layer), the backscatter channel was emulated using a wired connection. However, the operation is similar to using RF for transmission; in fact, it is based on toggling a GPIO pin for sending and generating external GPIO interrupts for receiving.
Communication between nodes for sending data is via the UART and is therefore also

wired.
The choice to use UART rather than a GPIO pin to send the packet bit by bit was dictated by the difficulty of managing the latter implementation. Using a GPIO pin and sending either a high value or a low value depending on the bit value of the sequence to be sent would require sampling the signal at the sending frequency. This would result in adding complexity to the system.
In any case, since the communication stack presented is modular, replacement of the Physical Layer from UART to an implementation working with other types of communications does not require modification of the upper layers.
To observe the behavior of the nodes, a wired connection was made between the computer and the boards through another UART.

Tests consist of 20 minutes of emulated system operation; it is important to note that these tests are limited to a short period of time and to limited cases. In order to get a more realistic overview of system performance, longer tests need to be conducted.

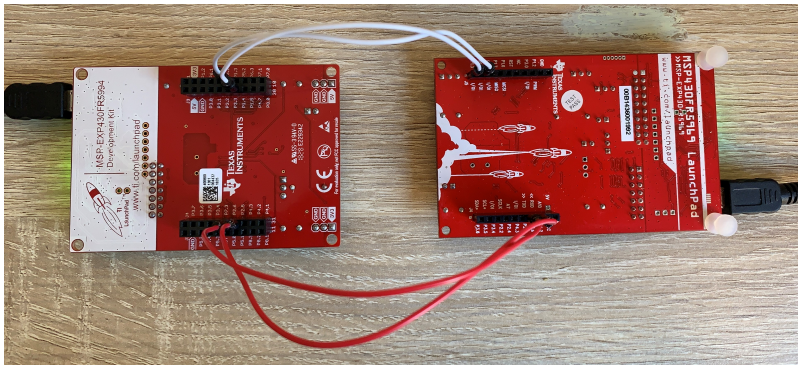

Figure 4.2: Test bed with MSP430FR5994 on the left and MSP430FR5969 on the right, white wires are for TRAP Layer, red wires are for data transmission

4.2.1 Energy Params For Emulation

During the emulations, we do not have a system that can collect energy from its surroundings; therefore, the node capacitor is charged by a power source based on a random generation function.
The average percent increase in capacitor charge per second is declared before the test results.
Capacitor energy is consumed for data collection from the environment and forwarding to the application layer; this consumption during the emulation is fixed at 32% of the maximum capacity.
Data transmission requires, in this emulation, a charge level of 64% or more of the maximum capacitance of the capacitor, unlike reception, for which consumption of 62% of the capacitance of the capacitor is simulated.
These estimated consumption values were decided by taking as reference a 2mF capacitor, a BGM220S BLE module [10] for transmission, and a DHT11 humidity and temperature sensor [5] to collect the data from the surrounding environment.
Compared to SRAM, FRAM access results in a somewhat higher power consumption [7], so when the stack is used, an additional 1% consumption will be added to these values. In fact, the FRAM is used in receiving to save the data, in sending to retrieve the data, and in producing the packet to do the saving.
It is important to point out that the values presented are used only for emulation and do not reflect exactly the real capacitor capacity and consumption of the board.
When the simulated charge level of the capacitor reaches 0%, the node is reset. To reset the node, an instruction from an invalid memory allocation is fetched.

Energy status update period is set to 1. This data item indicates every when the node energy information is sent to the network.

4.2.2 Production Of Packages During The Emulation

Data production on the nodes was simulated at fixed intervals, the emulations were performed with two different production patterns.
For the first test group, data was produced repeatedly every 60 seconds on the first node and every 90 seconds on the second node.
For the second test group, data was produced repeatedly every 60 seconds on the first node and every 75 seconds on the second.
This is useful for understanding the performance of the system with different production patterns.

4.2.3 First test group

In this first set of emulations, the operation of the system with and without the presented stack is tested. The energy level of the nodes is sent via Layer TRAP, and packets are produced repeatedly every 60 seconds on one node and every 90 seconds on the other.
Summary of the parameters of the emulation group:

Packet production repetition time on node 1:	60 seconds
Packet production repetition time on node 2:	90 seconds
Energy status update period:	1 second
Simulated energy consumption in TX:	64% (+1% if the stack is used)
Simulated energy consumption in RX:	62% (+1% if the stack is used)
Simulated energy consumption for packet production:	32% (+1% if the stack is used)

1. First emulation: average energy increment 0.71% per second
   For the first test an average energy growth per second inside the capacitor of about 0.71% is emulated.
   The result with the stack presented shows the sending of 5 packets and the successful reception of 5 packets, as opposed to not using it with which a result of 8 packets sent and 0 packets received is obtained (Figure 4.3).
   The success rate is then 100% using the stack and 0% without using the presented implementation.
   It is important to note that fewer packets are sent with the use of the stack than with emulation without the stack. This result is closely related to the delay introduced by the TRAP protocol to allow the nodes to reach the correct energy level before starting transmission.
   However, it is certainly better to successfully transmit fewer packets than to send more packets but not be sure of correct reception by the receiving node.

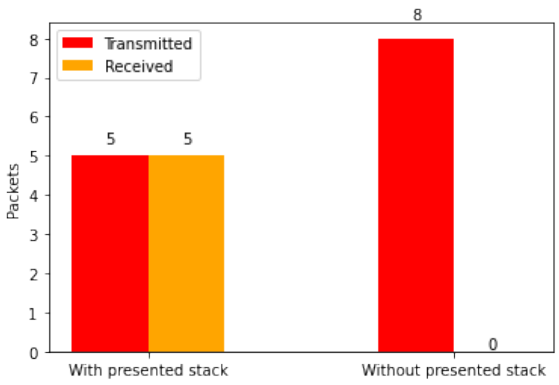

Figure 4.3: Emulation results with average energy increment of 0.71% per second

2. Second emulation: average energy increment 1.25% per second
   This second emulation assumes an average energy growth of 1.25% of the capacitor capacity per second.
   If we go to analyze the data obtained from the tests, we can see that the results without using the stack are slightly better than what was seen in the previous emulations. This improvement is due to higher average energy growth.
   In any case, the results obtained with the stack presented are still better than those obtained without using any protocol.

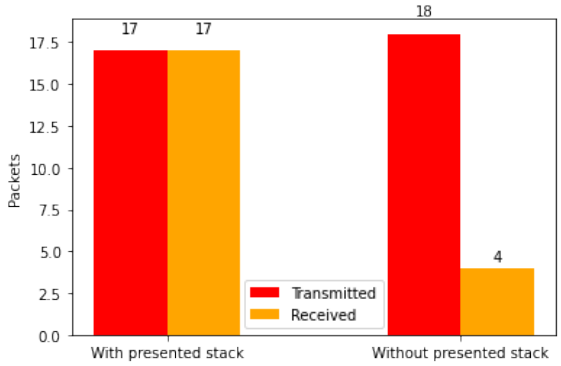

Figure 4.4: Emulation results with average energy increment of 1.25% per second

3. Third emulation: average energy increment 2% per second
   Following the trend of previous tests, the average value of per-second increment of energy within the capacitor is also changed in these emulations. In particular, an average increment of 2% is proposed.
   Similar to what highlighted in the previous test, the result without the stack achieves a higher transmission success rate than the first case presented, this increase is due to a higher growth of the energy inside the capacitor.
   However, the stack allows the system to achieve better performance.
   It is interesting to note that, during this emulation, the number of total transmissions using the stack is higher than the number of transmissions without the stack (Figure 4.5). This increase in transmissions is due to the saving of packets already produced in non-volatile memory and consequently as soon as possible the saved packets are sent without waiting for a new production.

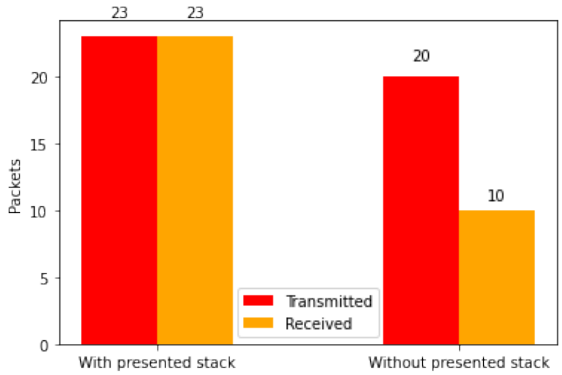

Figure 4.5: Emulation results with average energy increment of 2% per second

4.2.4 Second test group

This second group of emulations follows the same energy pattern presented with the previous group; however, having different periods for packet production the results obtained in relation to the number of transmissions are slightly different. The production periods are set to 60 seconds for the first node and 75 seconds for the second.
Summary of the parameters of the emulation group:

Packet production repetition time on node 1:	60 seconds
Packet production repetition time on node 2:	75 seconds
Energy status update period:	1 second
Simulated energy consumption in TX:	64% (+1% if the stack is used)
Simulated energy consumption in RX:	62% (+1% if the stack is used)
Simulated energy consumption for packet production:	32% (+1% if the stack is used)

1. First emulation: average energy increment 0.71% per second
   The first emulation assumes a percentage energy growth of 0.71% of the capacitor capacity per second.
   It can be seen from the results obtained that the number of successful transmissions is significantly higher if the presented stack is used (Figure 4.6).
   The result is particularly significant in that it shows that, in the case of low energy growth within the node, the presented stack, due to the implementation of the TRAP protocol, performs better than not using any control during transmission.
   Again, the lower number of transmission actions observable with the use of the stack is due to the TRAP protocol delaying sending until the energy levels required to successfully complete the transmission are reached.

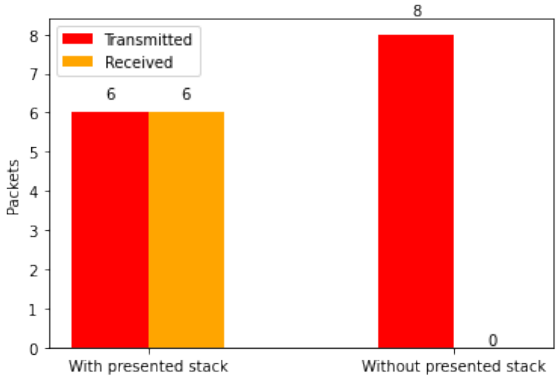

Figure 4.6: Emulation results with average energy increment of 0.71% per second

2. Second emulation: average energy increment 1.25% per second
   Also in this tests, results obtained with the stack are better in relation to the one without.
   However, it is interesting to note that, with a higher energy growth than in the previous test, more packets can be sent using the stack. This increase is due to the use of FRAM memory by the stack, which then creates a persistent queue of packets to be sent that is not dropped on power failure.

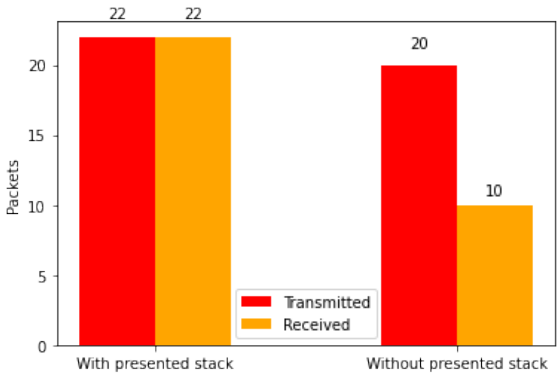

Figure 4.7: Emulation results with average energy increment of 1.25% per second

3. Third emulation: average energy increment 2% per second
   The third test presents very similar results (Figure 4.8) to those described in the second test (Figure 4.7), also in this emulation the success ratio of the stack is 100% as opposed to the one without the stack which is significantly lower (about 37%). We can also highlight in this test the higher number of packets sent using the stack despite the delay in transmission introduced by the TRAP protocol to achieve the energy levels required for successful transmission.

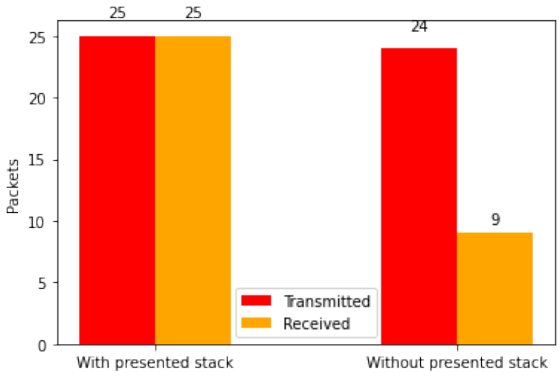

Figure 4.8: Emulation results with average energy increment of 2% per second

4.3 Results Discussion

The tests carried out showed that in the case of using the stack presented in this thesis, the performance of the system in terms of successfully sent packets goes up significantly. Calculating an average relative to the success rate obtained with the stack we observe that is almost 70 percentage points higher than the result obtained without the use of any protocol (Figures 4.9).

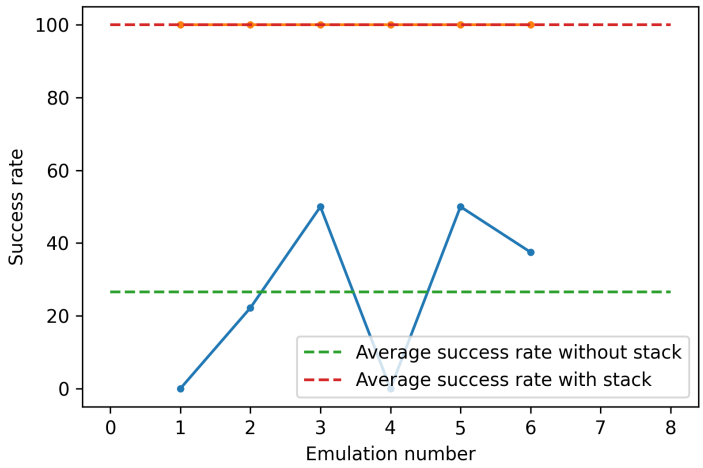

Figure 4.9: Plot representing the success rate in transmission obtained with and without the presented stack. The dashed line represents the average value obtained. The stack improves system performance by almost 70%.

In particular, it can be seen that under conditions in which the capacitor is charged by an energy source capable of providing only a small percentage charge per second (Figure 4.3 and Figure 4.6) the performance of the system using the presented stack is very good in relation to the number of packets successfully sent.
This improvement is mainly due to the use of the TRAP protocol to make the node aware of the energy level of neighboring nodes and thus avoid wasting energy on transmissions that cannot be completed.

It is important to remember how the tests were not run for long periods of time and how they did not take into account all possible situations that the system might encounter during ordinary operation. However, they do allow for a general idea regarding the performance of the system with and without the use of the presented stack.

5 Conclusion And Future Work

5.1 Conclusion

In this thesis, a tiny network stack for battery-free communication is designed, implemented and evaluated.
The presented system is published in an open source manner at the project's GitHub page [1] where it is also possible to find up-to-date documentation.
The stack has the features required in Chapter 1, thus saving data in non-volatile memory, retrieving it to perform sending to the network nodes or to the Application Layer to process it, recognizing expired packets to avoid wasting energy in sending/receiving invalid data, and implementing the TRAP protocol to make nodes aware about the energy available on neighboring nodes.
In addition to this, the presented system is modular in that it adapts to different transmission technologies by replacing only the Physical Layer and is scalable since it is not tied to a fixed number of nodes in the network.
Another feature of the system presented is that it hides implementation complexity from the Application Layer. For this reason, the system exposes only a small set of functions that can be used by the Application Layer. An example of its use is as follows:

```
....
while (1)
{
    dataProduced(datax, datay, dataj, dataz, destinationNode);
    dataSend();
    storedData dataRec = getData();

}
....
```

From the evaluation chapter, it was found that the results obtained with the presented stack are better than not using any protocol. In fact, the transmission success rate stands at 100% if the stack is used, around 26% if it is not used.
In addition to the loss of data in transmission the stack helps to reduce energy waste, in fact it is important to note that all packets sent and not received due to low energy on the receiving node meant a loss for both nodes: the sending node lost energy for producing and sending the data, while the receiving node lost energy for receiving a portion of the packet.

5.2 Future Work

Due to lack of time (tests on real data are usually very demanding and take several days to complete a single trial), numerous modifications, tests and experiments have been postponed. Future research will focus on deepening certain mechanisms, new suggestions for testing, new implementation techniques, or simple curiosity to develop something ever better.

[1] https://github.com/cancianilorenzo/Tiny-network-stack-for-battery-free-communication

Bibliography

[1] R. Bailey, Glen Fox, Jarrod Eliason, M. Depner, Dong-pyo Kim, E. Jabillo, J. Groat, J. Walbert, Theodore Moise, Scott Summerfelt, K. Udayakumar, J. Rodriquez, K. Remack, K. Boku, and J. Gertas. Fram memory technology - advantages for low power, fast write, high endurance applications. pages 485–, 11 2005.

[2] Taner Cevik and Serdar YILMAZ. An overview of visible light communication systems. *International Journal of Computer Networks and Communications*, 7:139-150, 11 2015.

[3] HeeJin Chae, Daniel Yeager, Joshua Smith, and Kevin Fu. Wirelessly powered sensor networks and computational rfid. *Springer Science & Business Media*, 01 2013.

[4] Shuhaizar Daud, R.Badlishah Ahmad, and Nukala Murhty. The effects of compiler optimizations on embedded system power consumption. pages 1–6, 12 2008.

[5] OSEPP Electronics. Dht11 humidity & temperature sensor.

[6] Walaa Elsayed, Mohamed Elhoseny, Sahar Sabbeh, and Alaa Riad. Self-maintenance model for wireless sensor networks. *Computers & Electrical Engineering*, 70:799–812, 2018.

[7] William Goh, Andreas Dannenberg, and Johnson He. Msp430™ fram technology-how to and best practices.

[8] Xiaosong Hu, Le Xu, Xianke Lin, and Michael Pecht. Battery lifetime prognostics. *Joule*, 4(2):310–346, 2020.

[9] Ravi Anant Kishore. 3 - harvesting thermal energy with ferroelectric materials. In Deepam Maurya, Abhijit Pramanick, and Dwight Viehland, editors, *Ferroelectric Materials for Energy Harvesting and Storage*, Woodhead Publishing Series in Electronic and Optical Materials, pages 85–106. Woodhead Publishing, 2021.

[10] Silicon Labs. Bgm220s wireless gecko bluetooth® module data sheet.

[11] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ENSsys'19, page 8–14, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Davide Sartori and Davide Brunelli. A smart sensor for precision agriculture powered by microbial fuel cells. In *2016 IEEE Sensors Applications Symposium (SAS)*, pages 1-6, 2016.

[13] Alessandro Torrisi, Kasım Sinan Yıldırım, and Davide Brunelli. Enabling transiently-powered communication via backscattering energy state information. In Sergio Saponara and Alessandro De Gloria, editors, *Applications in Electronics Pervading Industry, Environment and Society*, pages 192–201, Cham, 2021. Springer International Publishing.

[14] Alessandro Torrisi, Kasım Sinan Yıldırım, and Davide Brunelli. Reliable transiently-powered communication. *IEEE Sensors Journal*, 22(9):9124-9134, 2022.

[15] Dibin Zhu and Stephen Beeby. Kinetic energy harvesting. *Energy Harvesting Systems: Principles, Modeling and Applications*, 01 2011.