MITRE

MP220434
MITRE PRODUCT

Low-Power Backscatter Sensor Communications

National Security Accelerator Program
Capstone Project

Project No.: GNA051.N300.301.CA

Eli Baum

January 2022

McLean, VA

Abstract

Wireless communications systems are frequently constrained by size and power. Occasionally, it may be desirable to communicate with extremely small, low-power systems. Here, we present a backscatter sensor communications platform, which modulates and *reflects* ambient radio frequency signals, rather than *radiating* its own signals in the manner of a standard radio transmitter. Our prototype system can communicate at approximately one kilobit per second over a distance of 1.3 meters. We draw on past research to show that more advanced schemes could permit greater transmission speeds over more useful distances. One possible application for such technology includes distributed low-power sensing.

Acknowledgments

I would like to thank Dr. Michael Balazs, Andy Chapman, and Terri Roberts of MITRE’s National Security Accelerator Program for their support and encouragement.

Samantha Palazzolo, Terry Roberts, Evan Ding, and Eric Renda provided invaluable technical guidance, and helped me find my footing in the early days of my career at MITRE.

Table of Contents

1 Introduction

Wireless communications systems require significant power and can be rather large due to the analog circuitry required to modulate and amplify radio signals. Here, we propose a method of using *backscatter technology* to build extremely low-power wireless sensors. Without the normal requirements of radio transmitters, these backscatter sensors can be small, cheap, and long-lived; furthermore, since they emit no power, they are difficult for an adversary to detect.

A backscatter transmitter modulates reflected radio signals from another source, rather than radiating signals that it generates. Since the transmitter must do nothing more than switch a transistor at a given frequency, it requires very little power: the power requirements for communication are shifted to the signal source (or "illuminator") and processing is fully offloaded to a receiver, which may be attached to larger ground, air, or maritime platforms, where power and size are less constrained.

Backscatter systems can also be cheaply manufactured to provide distributed sensing capabilities. Users could distribute thousands of sensors across an area of interest and fly data-collection drones over the sensor field. Alternatively, a fixed station could read data in real time from sensors at a distance. Such deployments have already been validated in a prototype soil-moisture sensing network.[1]

In this prototype, a software-defined radio (SDR**)** is used to generate a fixed-frequency tone at 915 MHz. The sensor modulates data at approximately 150 kHz; the receive side of the SDR is therefore tuned to 915.15 MHz. We attach a magnetic sensor to demonstrate one possible application. However, this demonstration is not the limit of backscatter sensing technology: academic research has demonstrated the ability for relatively complex backscatter systems to be powered entirely by harvested ambient radio frequency energy,[2] while other studies have used custom waveforms to receive backscatter signals over distances of more than a kilometer.[3] Furthermore, other environmental illuminators may be used, including broadcast TV or radar, removing the requirement for a separate transmitter.

[1] S. Daskalakis, S. D. Assimonis, E. Kampianakis and A. Bletsas, "Soil Moisture Scatter Radio Networking with Low Power," in *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 7, pp. 2338-2346, July 2016. doi:10.1109/TMTT.2016.2572677

[2] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota and Joshua R. Smith. 2017. Battery-Free Cellphone. *PACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 25 (June 2017), 20 pages. doi:10.1145/3090090

[3] Vamsi Talla, Mehrdad Hessar, Bryce Kellogg, Ali Najafi, Joshua R. Smith and Shyamnath Gollakota. 2017. LoRa Backscatter: Enabling The Vision of Ubiquitous Connectivity. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, Article 105 (September 2017), 24 pages. doi:10.1145/3130970

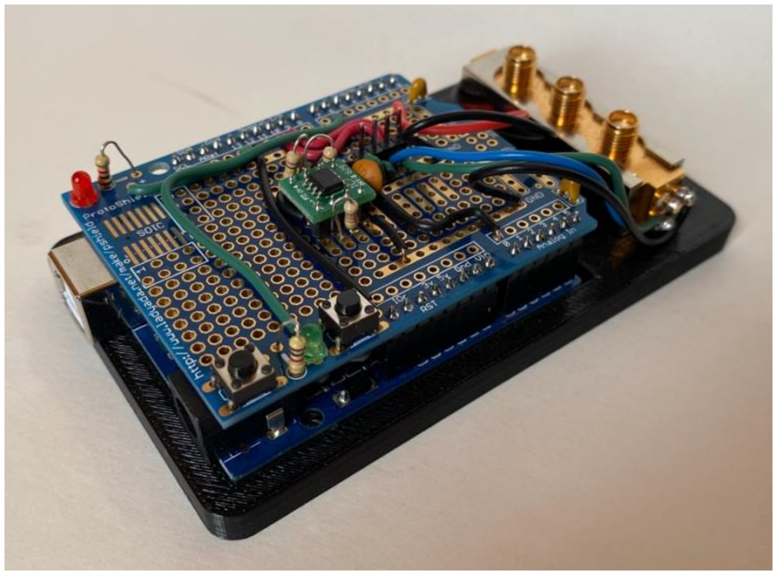


Figure 1: Our backscatter sensor module. The bulk of the current module is an Arduino. The green circuit board is a magnetic sensor; the RF switch is in the rear.

2 Backscatter Communications

2.1 Theory

Backscatter communication may seem counterintuitive compared to standard wireless communication systems. The following analogy should help clarify this mode of transmission.

Consider two parties, Alice and Bob, who wish to communicate with each other, across some significant distance, at night. (For the purposes of argument, assume a line of sight exists between them.) The simplest method for Alice to send a message would likely be for her to point a flashlight at Bob and modulate its intensity according to some scheme (say, Morse code). Bob could record Alice's pulses of light and translate them into a message.

However, there are several drawbacks to this system. Alice must carry a flashlight, which may be heavy, require large power supplies, and generate significant heat. Additionally, if Alice would like to communicate covertly (or is trying to hide her location), a flashlight might be too easy for an adversary to detect.

Alice comes up with an alternative scheme: she carries a mirror which she uses to carefully reflect moonlight in Bob's direction. The resulting light will be much dimmer, from Bob's perspective, but Alice no longer needs a flashlight.

Alice and Bob soon run into an issue – if the moon is not in the sky, or it is cloudy, they are unable to communicate. Seeking more reliable communication, Bob sets up a lantern at a third location; Alice can once again either reflect, or not reflect, the light from the lantern towards Bob.

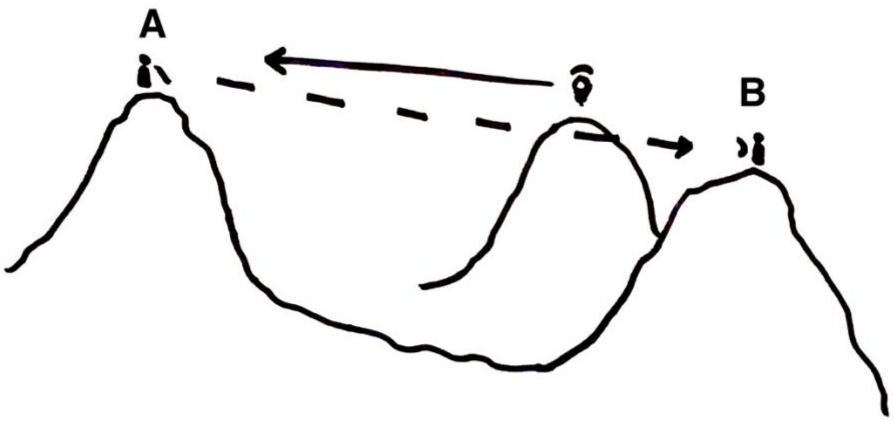


Figure 2: Alice stands on the left hill with a mirror. Bob places a lantern on the central hill and stands on the rightmost hill.

One problem remains: how can Bob see Alice's dim reflections, when the lantern is so much brighter? Perhaps Bob is in a shadow from his lantern and takes advantage of directionality. However, imagine a *fluorescent mirror* – that is, one which changes the wavelength of reflected light. Then, Bob could filter the wavelength of light coming from his lantern; only Alice's signal will remain.

Many of the concepts (and compromises) discussed above apply equally well to radio communications. The final scenario – where Bob illuminates Alice's color-changing reflector and detects the return signal – closely mirrors how backscatter communications works.

In this prototype, we use an SDR as both illuminator and detector. The transmit (TX) end of the SDR broadcasts a constant-frequency cosine tone; this is similar to Bob's lantern. The analog of the mirror is an antenna connected to the RF switch, which can either reflect or attenuate the illuminating signal.

For the "color-changing" component of Alice's mirror, we take advantage of constructive interference: given two signals A and B, with respective frequencies f_A and f_B, the addition of those two signals $A + B$ will appear to have a frequency component at $f_A \pm f_B$.[4]

In this prototype, the cosine illuminator is transmitted at 915 MHz ($= f_A$); the sensor module is configured to toggle the RF switch at approximately 150 kHz ($= f_B$). Therefore, we expect artifacts to appear at 915 ± 0.15 MHz. By tuning the receive (RX) side of the SDR to 915.15 MHz and keeping its bandwidth below 300 kHz (we use 200 kHz, the minimum of our hardware), interference from the transmitter is entirely eliminated. At this point, the reflected signal can be modulated in various ways.

In the national security domain, backscatter was perhaps most famously used by the Soviet Union during their bugging of the American Embassy in Moscow with "The Thing." This project is inspired by that use case, but we also consider witting, environmental sensing applications (rather than covert espionage uses) for backscatter technology.

[4] This effect is often noted in music, where the resulting perceived *beat frequency* can be used to tune an instrument, but we can just as easily create a frequency-shifting effect at radio frequencies.

2.2 A Word on RFID

Some commercial radio frequency identification (RFID) systems use backscatter technology. These systems are not extensible, and provide no way to transmit sensor data, as was desired here – instead, they generally transmit a fixed identification number – but they may provide future avenues for exploration.

The most well-known of such technologies is passive RFID, commonly encountered in electronic access control systems and contactless payment. In these systems, a tag reader generates an "interrogation" signal which is then reflected, and modulated, by the tag. Often (as in the case of a badge reader), the tag contains no power supply of its own, and instead harvests power from the interrogation signal itself.

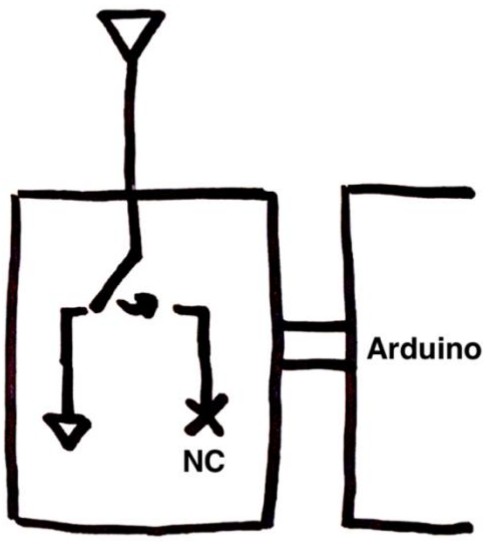
Usually, passive systems work only at sufficiently small distances where the reader and tag antennas can *magnetically couple* (on order of centimeters); no true radio signals are exchanged. This region is known as the “near field.” However, some passive RFID systems *do* indeed work via radio waves (or the “far field”) in a manner similar to the backscatter communication module presented here – certain electronic toll collection systems are a common example.[5]

Passive RFID contrasts with *active RFID*, which operates more like a normal wireless transmitter. Here, the interrogation signal "wakes up" the RFID tag, which then replies using its own dedicated transmitter. Such systems operate over greater distances, and with lower error, but require their own batteries and use much more power than purely reflective systems.

3 Prototype System

The prototype backscatter sensor module consists of an Arduino Uno (with an ATmega328 microcontroller), an RF switch, and a magnetic sensor. We take readings from the magnetic sensor, process them on the Arduino, and transmit data using backscatter. The SDR used was a USRP B210, but any similar SDR should work.

[5] Lazar N. Spasovic, Wen Zhang, Athanassios K. Bladikas, Louis J. Pignataro, Edip Niver, and Stanley Ciszewski. 1995. Primer on Electronic Toll Collection. *Transportation Research Record No. 1516, Intelligent Transportation Systems.* https://trid.trb.org/view/453148



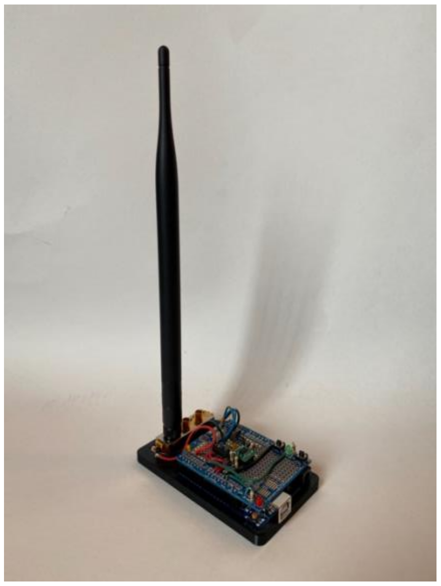


Figure 3: A diagram of the hardware setup, left. The antenna is either grounded (through a 50Ω resistor), or not connected. The completed sensor module, right.

3.1 Physical Layer

The physical layer consists of the actual modulation of information onto the 915 MHz carrier wave. We designed a simple physical layer for ease of implementation and rapid prototyping.

3.1.1 Hardware

The centerpiece of this prototype is an absorptive single-pole double-throw (SPDT) RF switch (Minicircuits ZX80-DR230-S+). The switch is connected to one of the microcontroller’s timer compare outputs, which oscillates at approximately 150 kHz. We connect an antenna to only one of the switch's ports and leave the others disconnected.

When the switch is in the open position, the antenna is connected to the unterminated center port. Therefore, incoming RF will be reflected. When the switch is closed, the antenna is terminated to ground via an internal 50Ω load, so all energy will be absorbed.

3.1.2 Amplitude Shift Keying

We use amplitude-shift keying (ASK) to modulate data; this is the simplest scheme that could be used to build a viable prototype.

We experimented with frequency-shift keying (FSK) but decided not to move forward with it. FSK provides superior information density but is less noise-tolerant and more computationally complex to decode. Furthermore, the usefulness of an FSK system was limited by the granularity of the microcontroller's internal oscillator.

Other schemes, such as phase-shift keying (PSK) and quadrature amplitude modulation (QAM) were not possible with the hardware available on the sensor module. The RF switch contained a purely resistive internal load; phase-related modulations would only be possible with reactive (i.e., capacitive or inductive) loads. A fieldable backscatter solution would likely require more advanced hardware capable of these advanced modulation schemes.

3.1.3 FM0 Encoding

Once we can modulate the signal, the next problem is to send bits. We need a *clock recovery* method – how do we align our receiver's data stream with that of our reflector? In this prototype, we have used *FM0 encoding*, an extremely simple run-length limited (RLL) code. RLL codes are designed to contain sufficient transitions to enable clock recovery; without FM0, a string of identical values could be difficult to differentiate.

In fact, FM0 is quite similar to Morse code. *One* is encoded as a long pulse, while *zero* is encoded as two short pulses (hence the "FM" or "frequency modulation" in its name). The actual level of these pulses is unimportant; rather, we only measure their *transitions*. In practice, after every bit, we toggle the signal, but for a `0`, we also toggle halfway through that bit period. For example, to send the data `0110`, we encode:

Data	X	0	1	1	0	X
Pulses	X	SS	L	L	SS	X
Transmit	0	10	11	00	10	1

where `SS` represents two short pulses; `L` represents one long pulse.

While this is not a particularly efficient scheme (it takes 2 symbols to send one bit, making FM0 a rate-½ code), it is simple to implement and appropriate for prototyping.

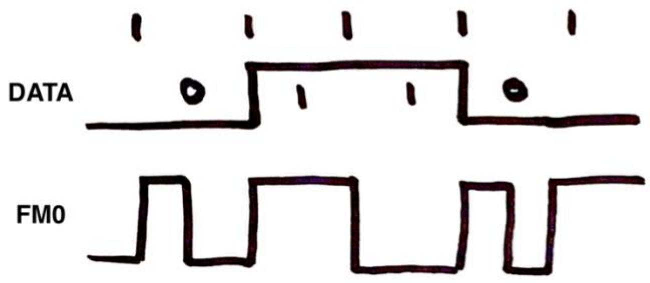


Figure 4: Example FM0 encoding of data.

3.2 Packet Layer

Various data-transmission strategies are possible; we chose continuous data transmission since this is simple and beneficial for demonstration purposes. We designed a packet format, as follows:

Access Code	32 bits: `0xe15ae893`
Sequence Number	8 bits
Data Length in bytes	8 bits
Data	8 bits per reading

The access code, `0xe15ae893`, is a commonly-used non-cyclical access code.[6] This access code correlates poorly with itself at all offsets except zero, which makes it easy for the receiver to synchronize with the incoming packet.

The sequence number provides some level of error detection. It is incremented for each packet (overflowing from 255 to 0); the decoder only accepts a packet if the decoded sequence number is one greater than the previous. More advanced schemes with error detection *and* correction would be desirable in a fieldable system. We implemented a Hamming encoder and decoder but decided not to include it in the prototype for the sake of simplicity.

4 Decoder Implementation

The Python decoder is implemented as a series of layers. Each layer receives data from the one above, provides some processing functionality, and passes data down.

1. **SDR sampling.** GNU Radio is used to read samples from the SDR.
2. **ASK decoding.** A simple ASK decoder converts samples into binary values. Since the sampling rate is much higher than the symbol rate (by about two orders of magnitude), there are multiple digitized samples per actual ASK symbol.
3. **Pulse length determination.** FM0 decoding depends on the length of pulses in the input stream. First, we perform a *coarse* search – this layer simply returns the time between binary transitions.
4. **Pulse length clustering.** Due to noise and timing inaccuracies, not all pulses will precisely match the expected short or long length. In this step, we cluster the received pulses around their estimated true length and account for noise. For example, a burst of noise in the middle of a long pulse of length 2 may make that pulse look like three short pulses of lengths 0.9, 0.2, and 0.9. This layer detects the noisy burst and converts the set of pulses into the correct long pulse.
5. **FM0 decoding.** Now, we can turn pulses into symbols: two short pulses are a zero; one long pulse is a one.
6. **Packet decoding.** Each packet begins with the non-cyclical access code. Using a sliding 32-bit window, we correlate against the access code to find the start of a packet (the correlation reaches a maximum at perfect alignment).[7] Next, we check the sequence number, and if correct, we read the length, and then the prescribed number of data bytes.
7. **Data graphing.** Finally, data values are plotted in real time with `matplotlib`.

5 Sensor Demonstration

The focus of this project was on backscatter communication itself, not on the data sent over the backscatter link. However, to demonstrate real-time communication capabilities, we add a magnetic sensor to the backscatter module. We take a moving average of the sensor readings and

[6] GNURadio cites this access code "historically interesting" but we find few other references to it on the Internet. Regardless, the access code worked well for our use case.

[7] In a later version of the prototype, we use the Hamming weight of the bitwise XOR rather than statistical correlation since this ends up producing fewer false positives. However, for binary values, these two measures are similar.

send 16 8-bit values per packet. With these parameters, each packet is therefore 176 bits. At a symbol rate of 2000 baud, we can send 1000 bits per second, giving about 5.7 packets, or 91 readings, per second.

An example data transmission is shown below. The sensor was not turned on until $\boldsymbol{t = 5}$; values prior to that point are noise. A value of 127 represents a magnetic field of zero, with values above or below 127 reflecting a positive or negative magnetic field, respectively. Over the course of one minute, a magnet was moved around the sensor in various orientations.

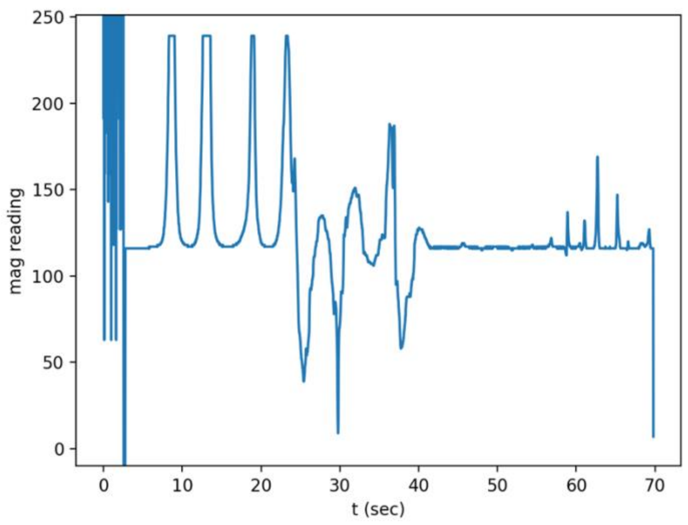


Figure 5: A plot of sensor readings transmitted via backscatter.

6 Results

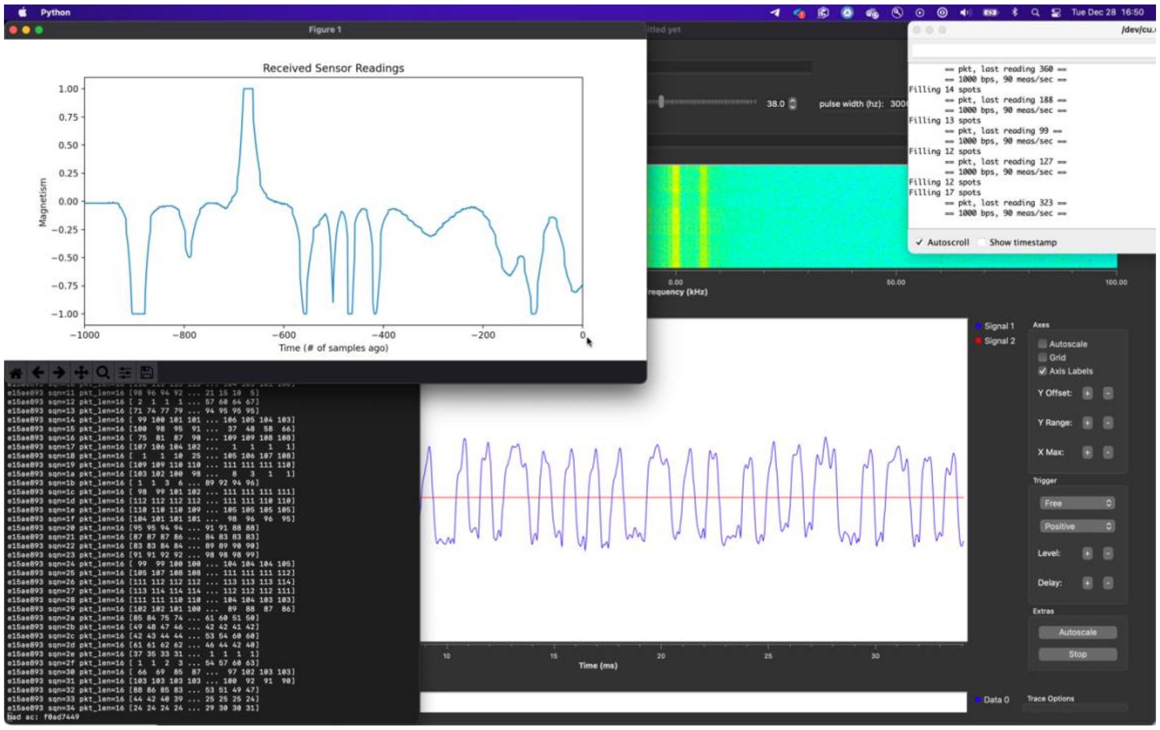
This prototype was meant as a proof of concept for future study and has demonstrated that backscatter is a viable method for low-power, low-bitrate communications. We hid a magnet inside of a toy garbage truck and demonstrated how the backscatter sensor module was able to detect the presence of the truck. Of course, with other sensors, various environmental parameters of interest could be measured, recorded, and analyzed.

A summary of the test parameters follows:

- Physical layout
  - 132 cm between transmitter and sensor
  - 132 cm between receiver and sensor
  - Approximately 100 cm between transmitter and receiver (to prevent in-band interference)
- RF parameters
  - TX gain: 89 dB (SDR maximum)
  - RX gain: 26 dB
  - Lowpass filter width: 3 kHz

    - Frequency calibration: -1.4 kHz
- Packet parameters
    - 2000 symbols per second (1000 bps)
    - ~ 91 measurements transmitted per second

Placement of the TX and RX antennas in relation to the sensor is an important factor for the successful operation of the sensor module. At very close distances (on order of the wavelength of the carrier wave), magnetic coupling dominates, and the system is more akin to a passive near-field RFID system. For the 915 MHz carrier used here, one wavelength is 32.76 cm. To ensure we are operating in the RF far-field, antenna separation should be two wavelengths or more.



```
        == pkt, last reading 360 ==
        == 1000 bps, 90 meas/sec ==
Filling 14 spots
        == pkt, last reading 188 ==
        == 1000 bps, 90 meas/sec ==
Filling 13 spots
        == pkt, last reading 99 ==
        == 1000 bps, 90 meas/sec ==
Filling 12 spots
        == pkt, last reading 127 ==
        == 1000 bps, 90 meas/sec ==
Filling 12 spots
Filling 17 spots
        == pkt, last reading 323 ==
        == 1000 bps, 90 meas/sec ==
```

```
e15ae893 sqn=11 pkt_len=16 [98 96 94 92 ... 21 15 10  5]
e15ae893 sqn=12 pkt_len=16 [ 2  1  1  1 ... 57 60 64 67]
e15ae893 sqn=13 pkt_len=16 [71 74 77 79 ... 94 95 95 95]
e15ae893 sqn=14 pkt_len=16 [ 99 100 101 101 ... 106 105 104 103]
e15ae893 sqn=15 pkt_len=16 [100  98  95  91 ...  37  48  58  66]
e15ae893 sqn=16 pkt_len=16 [ 75  81  87  90 ... 109 109 108 108]
e15ae893 sqn=17 pkt_len=16 [107 106 104 102 ...   1   1   1   1]
e15ae893 sqn=18 pkt_len=16 [  1   1  10  25 ... 105 106 107 108]
e15ae893 sqn=19 pkt_len=16 [109 109 110 110 ... 111 111 111 110]
e15ae893 sqn=1a pkt_len=16 [103 102 100  98 ...   8   3   1   1]
e15ae893 sqn=1b pkt_len=16 [ 1  1  3  6 ... 89 92 94 96]
e15ae893 sqn=1c pkt_len=16 [ 98  99 101 102 ... 111 111 111 111]
e15ae893 sqn=1d pkt_len=16 [112 112 112 112 ... 111 111 110 110]
e15ae893 sqn=1e pkt_len=16 [110 110 110 109 ... 105 105 105 105]
e15ae893 sqn=1f pkt_len=16 [104 101 101 101 ...  98  96  96  95]
e15ae893 sqn=20 pkt_len=16 [95 95 94 94 ... 91 91 88 88]
e15ae893 sqn=21 pkt_len=16 [87 87 87 86 ... 84 83 83 83]
e15ae893 sqn=22 pkt_len=16 [83 83 84 84 ... 89 89 90 90]
e15ae893 sqn=23 pkt_len=16 [91 91 92 92 ... 98 98 98 99]
e15ae893 sqn=24 pkt_len=16 [ 99  99 100 100 ... 104 104 104 105]
e15ae893 sqn=25 pkt_len=16 [105 107 108 108 ... 111 111 111 112]
e15ae893 sqn=26 pkt_len=16 [111 112 112 112 ... 113 113 113 114]
e15ae893 sqn=27 pkt_len=16 [113 114 114 114 ... 112 112 112 111]
e15ae893 sqn=28 pkt_len=16 [111 111 110 110 ... 104 104 103 103]
e15ae893 sqn=29 pkt_len=16 [102 102 101 100 ...  89  88  87  86]
e15ae893 sqn=2a pkt_len=16 [85 84 75 74 ... 61 60 51 50]
e15ae893 sqn=2b pkt_len=16 [49 48 47 46 ... 42 42 41 42]
e15ae893 sqn=2c pkt_len=16 [42 43 44 44 ... 53 54 60 60]
e15ae893 sqn=2d pkt_len=16 [61 61 62 62 ... 46 44 42 40]
e15ae893 sqn=2e pkt_len=16 [37 35 33 31 ...  1  1  1  1]
e15ae893 sqn=2f pkt_len=16 [ 1  1  2  3 ... 54 57 60 63]
e15ae893 sqn=30 pkt_len=16 [ 66  69  85  87 ...  97 102 103 103]
e15ae893 sqn=31 pkt_len=16 [103 103 103 103 ... 100  92  91  90]
e15ae893 sqn=32 pkt_len=16 [88 86 85 83 ... 53 51 49 47]
e15ae893 sqn=33 pkt_len=16 [44 42 40 39 ... 25 25 25 24]
e15ae893 sqn=34 pkt_len=16 [24 24 24 24 ... 29 30 30 31]
bad ac: f0ad7449
```

Figure 6: A screenshot of the full software stack. Clockwise from top right: 1) debug output from the microcontroller; 2) a time-domain view of the received ASK waveform; 3) debug output from the Python decoder; 4) processed magnetic sensor readings.

6.1 Caveats and Future Work

After many unreliable tests, we realized (or at least suspect) that due to the short distances used in this demonstration, poor performance was occurring when the backscatter antenna was in a node (or null) of the transmitting tone. The radio waves are not traveling far enough for multi-path effects to make a difference, and since the illumination signal is a single sine wave, there will exist some nulls in the signal strength. We solved this problem by ensuring the backscatter sensor was some multiple of the wavelength away from both the TX and RX sensors (here, about 33 cm). In more realistic environments, frequency hopping or spread-spectrum modulation could help mitigate this problem.

We are also interested in exploring more robust, higher bitrate transmissions. As mentioned above, there are many physical layer modulation schemes, such as FSK or PSK, that could enable faster transmission. Frequency-swept chirps, as used in LoRa communications, have shown success in backscatter systems, with particularly impressive performance over long distances. However, different RF hardware (such as reactive components) would be required (as well as a high-precision digital timer), particularly to enable phase shifts. Even so, we caution that high bandwidth communications will likely not be possible with backscatter: since the returned power is much lower than a normal radio transmitter, the channel capacity is necessarily limited.

Additionally, more complex coding schemes (rather than FM0) would allow for a greater symbol rate and more efficient clock recovery; error correction would also be an improvement over naive sequence number checking.

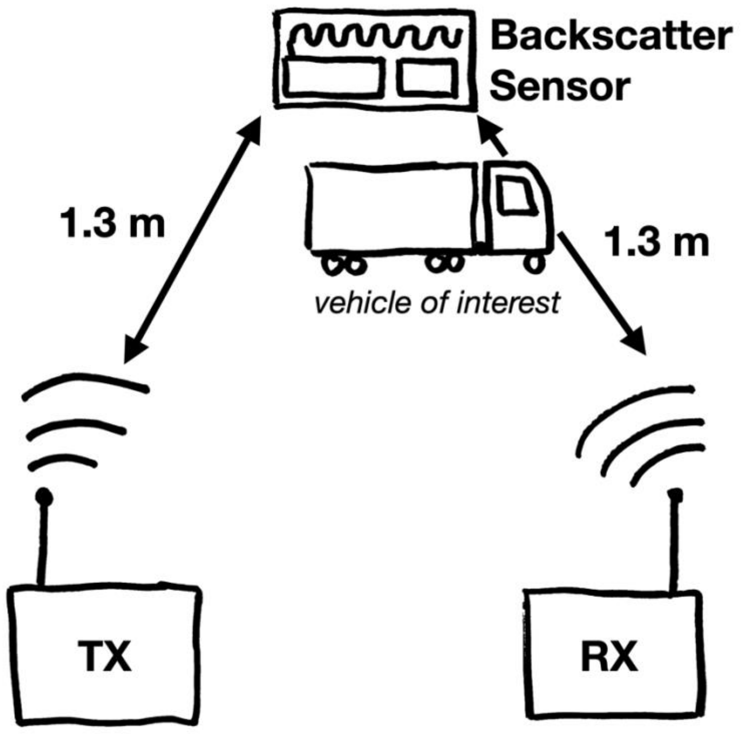


Figure 7: Overview of our test setup. Transmit (TX) and Receive (RX) are in this case run via the same software-defined radio, but with spaced-apart antennas.

7 Conclusion

Backscatter technology will not replace standard wireless transmission in areas where it is currently used. However, due to its low cost, power, and size requirements, backscatter has the potential to revolutionize distributed sensing applications. We demonstrate a prototype backscatter sensing module that can wirelessly transmit 1000 bps of data reliably over distances of about a meter. For applications where low-bitrate communications are acceptable, we hope that future research could significantly increase the maximum transmission distance.

Alternatively, the maximum bitrate of the system could be improved, allowing for for other kinds of communications data, such as text or audio.

7.1 Possible Use Cases

Consider a hypothetical scenario where an organization wishes to monitor vehicle traffic in a large area. Rather than installing thousands of vibration sensors with high-power transmitters (say, cellular modems), low power *backscatter* sensors could be distributed along roads in the area. These sensors would have much longer lifespans (and be harder to detect); from time to time, friendly vehicles with backscatter transmitter/receiver pairs could drive by and offload data from the sensors.

Backscatter would also be beneficial in maritime ports, where sensors could be attached to shipping containers. This deployment would, in a sense, be a form of advanced RFID – but a more complex backscatter tag could relay collected data from the container's voyage, rather than just an identification code. When a container arrives in port, cranes or other autonomous systems would be able to ingest the data, all while requiring no power supply on the container itself.

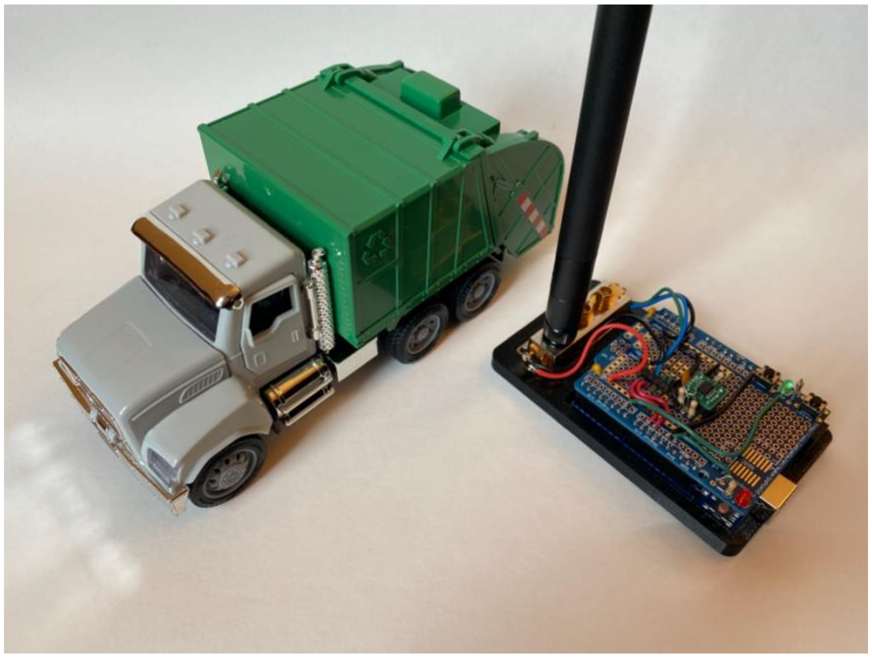


Figure 8: A neodymium magnet was installed inside a toy garbage truck to demonstrate a possible use case.

Appendix A Software

The full code for this project is available on Github:

`https://github.com/elimbaum/backscatter-capstone`

Below, we include pseudocode for both the reflector (transmitter) and receiver to provide some insight into their operation.

A.1 Reflector

```
# Reflector Pseudocode
# Continuously takes sensor readings and sends them out
#
# Target: Arduino Uno (ATMega328)

setup:
    configure i/o
    configure timers

    # load access code, sqn, length
    enqueue packet header

loop:
    s = count empty slots in the queue
    repeat s times:
        r = measure magnetic sensor
        enqueue r

        if packet is full:
            # load access code, sqn, length
            enqueue packet header

# Runs every 500 microseconds
timer interrupt:
    pull a bit off the queue
    if bit == 1:
        toggle RF switch at 150 kHz
    else:
        stop toggling RF switch
```

A.2 Receiver

```
# Receiver Pseudocode

# This section runs in GNURadio
receiver:
    load samples from radio
    retune to ASK center frequency
    low pass filter
    convert complex values to magnitudes
    send to decoder

# All other sections run in Python
decoder:
    ingest raw values from receiver
    binary slice

    # coarse pulse extraction
    pulses = run-length encode sliced data

    if any pulse is below noise threshold:
        combine adjacent pulses

    symbols = FM0 decode pulses
    packet = unpack symbols

    match = correlate packet against access code

    if no match:
        continue

    sqn = read sequence number
    if sqn is out of order:
        continue

    n = read packet length
    r = read n bytes
    add r to sensor_values

plot:
    continuously replot sensor_values
```