On-Off Noise Power Communication

Philip Lundrigan
lundrigan@byu.edu
Brigham Young University
Provo, Utah

Neal Patwari
npatwari@wustl.edu
Washington University in St. Louis
St. Louis, Missouri

Sneha K. Kasera
kasera@cs.utah.edu
University of Utah
Salt Lake City, Utah

ABSTRACT

We design and build a protocol called on-off noise power communication (ONPC), which modifies the software in commodity packet radios to allow communication, independent of their standard protocol, at a very slow rate at long range. To achieve this long range, we use the transmitter as an RF power source that can be on or off if it does or does not send a packet, respectively, and a receiver that repeatedly measures the noise and interference power level. We use spread spectrum techniques on top of the basic on/off mechanism to overcome the interference caused by other devices' channel access to provide long ranges at a much lower data rate. We implement the protocol on top of commodity WiFi hardware. We discuss our design and how we overcome key challenges such as non-stationary interference, carrier sensing and hardware timing delays. We test ONPC in several situations to show that it achieves significantly longer range than standard WiFi.

CCS CONCEPTS

• **Networks → Cross-layer protocols**; **Network range**; *Network protocol design*; *Home networks*; *Cyber-physical networks*; *Wireless local area networks*.

KEYWORDS

Internet of Things; long-range communication; WiFi; wireless communication

ACM Reference Format:
Philip Lundrigan, Neal Patwari, and Sneha K. Kasera. 2019. On-Off Noise Power Communication. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19), October 21–25, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3300061.3345436

1 INTRODUCTION

When WiFi devices are deployed in difficult-to-access locations over long periods of time, as is done in a variety of Internet-of-things applications, network and system management faces fundamental observability challenges. WiFi provides high data rates when the signal-to-noise ratio (SNR) is high; however, when the SNR falls below the threshold for the lowest data rate, the device achieves no data rate and the device disconnects from the network. A remote system manager cannot determine whether the device has a power outage or SNR too low. This paper explores a physical layer mechanism to provide "slow data" from a WiFi device to enable a system to know, for example, if the device is powered on, even when the device is unable to communicate using any 802.11 standard protocol. In this paper, we ask the following question: can we extend the range of communication of a WiFi device so that it can notify a receiver that it is still functioning, even though it cannot communicate over WiFi?

Our research is motivated by our past experiences managing many deployments of WiFi sensors in homes of human subject research study participants. These homes are "difficult-to-access": getting physical access to a sensor can take a deployment manager days or weeks to schedule a time that the study participant is available, and some study participants may feel inconvenienced enough to drop out of the study as a result. In one deployment, we placed a WiFi sensor in the bedroom, as required for the study. Although far from the home access point (AP), the sensor connected reliably at deployment time. Over the next weeks, the link occasionally had a low packet reception rate, but could reliably transfer the low rate sensor data as required. At one point, the sensor stopped reporting, and no data was received for more than 12 hours. We had no way of knowing if the sensor had lost power or if the WiFi link was disconnected. If the sensor has lost power, the sensor is not collecting data, which has negative consequences for the human subject study. If instead the WiFi link is disconnected, it is less problematic because the sensor is collecting data that the researchers will eventually be able to access after the sensor reconnects. As the manager could not tell which was the case, she managed the risk of data loss with the cost of inconveniencing the participant and contacted the participant to ask if the sensor was powered on. The participant moved a basket of laundry to check on the sensor, and the sensor immediately reconnected. Together they determined that the outage started at the same time the study participant placed the basket of laundry next to the sensor. Our key lesson: WiFi links that are reliable at the time of deployment may fail on any given day due to normal daily activities.

Wireless technologies exist that provide longer range than WiFi, such as cellular, LoRa [26], and 802.11ah [4]; however, these technologies have substantial drawbacks. Cost and power utilization are much higher with cellular compared to WiFi. Additionally, cell networks can suffer from dead spots inside of buildings. LoRa and 802.11ah promise longer ranges but require different radios to work. A wireless device would need to be outfitted with these radios, and the home or building would need a base station for that wireless technology, which would be costly or difficult to implement. On the other hand, WiFi is widely deployed in homes and buildings and is well studied and understood. It is also inexpensive to integrate into devices; a WiFi transceiver and microcontroller module can be purchased for less than US $3 [29]. Sensors using WiFi can be a cost-reducing measure because they are piggybacking on a widely deployed technology, which is a big incentive for budget constrained studies. To be able to continue to use WiFi hardware, but get longer range, would be a considerable advantage compared to alternative approaches.

Even with the cost reducing and widely deployed benefits of WiFi, a key challenge of dealing with WiFi devices is that WiFi is limited by the lowest data rate the access point (AP) supports. This means that if the SNR of a WiFi client is low enough such that the AP can not demodulate transmissions at the lowest data rate, the WiFi client will become unassociated from the network. The AP and device will no longer be able to communicate with each other until the SNR is higher and the WiFi client has gone through the association procedure again.

In developing a solution to this problem, we have the following design goals:

(1) A solution should have a longer range than standard WiFi. This will solve the observability problem of deployed WiFi devices by allowing a device to send some data even if it is slightly outside the range of WiFi.
(2) A solution should use the same radio that WiFi uses. This allows us to keep the benefits of WiFi (cheap, widely deployed, etc.).
(3) A solution should not change the hardware or WiFi firmware of the WiFi devices. This especially benefits already deployed devices that could be enhanced from such a solution with only a software update. This also makes such a system more realistically deployable in a real IoT environment because off-the-shelf WiFi components can be used.
(4) A solution should be 802.11 MAC compliant. This ensures that devices do not disrupt the normal function of a wireless network when deployed.

To meet these goals and thus provide a solution that supports longer ranges than standard WiFi, we build a novel protocol called *on-off noise power communication* (ONPC). ONPC is a physical layer protocol that allows a WiFi device to send data to a receiver, even when the device is unable to communicate using any 802.11 standard protocol. It is implemented purely in software and requires no changes to hardware. ONPC creates independent channels of communication at a much lower data rate by transmitting WiFi frames on and off in a pseudo-random pattern. Although the receiver is too distant to receive the data in these frames, it measures the impact of these transmitted frames on the noise floor, repeatedly, searching for the pseudo-random pattern. This allows one device to transmit data to another beyond the range of WiFi while using standard WiFi radios. For medium access control (MAC), we use orthogonal coding to allow multiple ONPC transmitters to transmit at the same time. Finally, we create *Stayin' Alive*, an application that uses ONPC to build a complete system that allows a manager to know if a WiFi device is still connected to a power source, or "alive", even if it is disconnected from WiFi, solving the observability challenge facing IoT deployments. ONPC is not designed to replace long range protocols, like LoRa, but to supplement WiFi. We recognize that there are specific scenarios where a protocol like LoRa would be a better fit. This work looks to enhance WiFi sensor deployments by creating ONPC and Stayin' Alive.

Building a system that meets our goals is challenging. First, most transceivers provide precise control of when the transmitter sends each symbol and receives each sample. Off-the-shelf WiFi devices do not provide this low-level control. Instead, frames are transmitted and the receiver measures noise floor samples at unknown times and with random delays. Some of this randomness is due to carrier sensing multiple access (CSMA), which is required to make ONPC 802.11 MAC compliant.

A second major challenge is that the noise power measurements we propose for the receiver are heavily impacted by other WiFi traffic. As we target transmitters that are further than the standard WiFi range, the effect of an ONPC transmitter on the noise floor at the receiver is very small in comparison to this interfering WiFi traffic. Our use of a pseudo-random code is designed to add coding gain to enable successful demodulation of the ONPC data. However, interfering WiFi traffic is particularly non-stationary, as individual devices' frames tend to be bursty and short-term. Thus the mean and standard deviation of the noise floor is temporarily and dramatically affected by other WiFi traffic.

We make several contributions in this paper:

(1) We introduce ONPC, the first protocol that runs on top of WiFi that extends the range of a WiFi device beyond the range of standard WiFi.
(2) We design an application, Stayin' Alive, which uses ONPC to help managers of WiFi device deployments know if a device is still functioning, even if it cannot communicate over WiFi.
(3) We implement ONPC and Stayin' Alive using off-the-shelf WiFi hardware and deploy it in various locations, including a home.
(4) We evaluate different aspects of ONPC and show that ONPC works beyond the range of WiFi. We also show that ONPC's MAC works with multiple ONPC transmitters.

The code for ONPC and Stayin' Alive are available on GitHub [19][20].

2 ONPC DESIGN

The general intuition for ONPC is as follows. If the SNR of a WiFi device drops below a certain threshold for the lowest data rate the access point supports, the device becomes disconnected, or unassociated, from the network. Though the transmission from the device cannot be decoded at the access point, due to low SNR, it will still have a small impact on the environment, namely raising the noise floor while it is transmitting. If the access point continuously monitors the noise power, it can observe those changes. The device can transmit WiFi frames in a certain pattern such that the access point can see changes in the noise floor and know that that specific device is causing the changes. Using this approach, an unconnected device can communicate beyond the range of WiFi.

With ONPC, we are trading a lower data rate for longer range communication compared to standard WiFi. Since we are using WiFi frames as a way to change the noise floor, our data rate will be much lower than standard WiFi, but with the advantage that it can be decoded from longer ranges.

2.1 Overview

ONPC consists of four components, as shown in Figure 1: WiFi devices, ONPC transmitters, an access point (AP), and an ONPC receiver. The WiFi device functions normally when within range of WiFi, transmitting or receiving data using WiFi. This device could be an IoT sensor or any other type of WiFi connected device. The ONPC transmitter is software running on the WiFi device. When the WiFi device disconnects from WiFi, an application, such as Stayin' Alive, starts the ONPC transmitter which sends frames in a pseudo-random pattern. The access point is used to collect noise measurements. The ONPC receiver requests noise measurements from the AP, processes the data, and detects the ONPC transmissions of the unconnected WiFi devices running the ONPC transmitter. This allows the receiver to receive data from the transmitter.

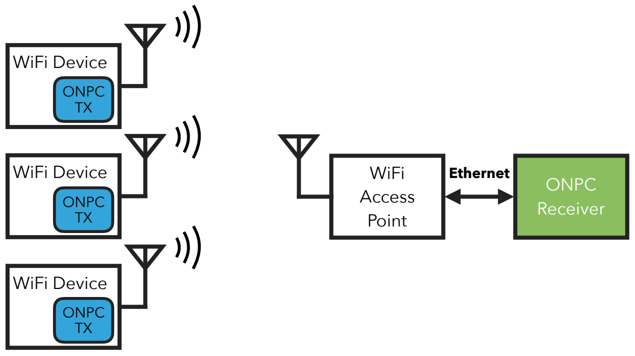

Figure 1: High-level overview of the components involved in ONPC. A WiFi device runs the ONPC transmitter when it is not connected to WiFi. The ONPC receiver uses the access point to collect noise measurements to detect the presence of an ONPC transmitter.

As stated previously, one of our goals with ONPC is not to modify any hardware or the WiFi firmware. Since we are choosing not to modify the AP, we need the ONPC receiver to process noise samples from the AP to decode the ONPC transmitter's data. One could imagine that this processing could be built into an AP's software in the future.

Next, we outline the design of the individual components of ONPC.

2.2 Transmitter

ONPC uses a similar approach to direct-sequence spread spectrum (DSSS). When the transmitter wants to transmit data to an ONPC receiver, the process goes as follows (see Figure 2). The transmitter uses a predefined pseudo-random symbol of ones and negative ones, which are called chips. To generate the symbol, we use a maximum length sequence because of its autocorrelation and cross-correlation properties [14]. This symbol is also known by the ONPC receiver. The length of the symbol plays an important role in identifying the symbol in the noise. The longer the symbol, the more likely the receiver will be able to detect the transmitter, however, a longer symbol takes longer to transmit. Unlike traditional DSSS systems where the carrier wave is multiplied by a one or negative one, our system is only able to transmit a WiFi frame (on) or not transmit a WiFi frame (off).

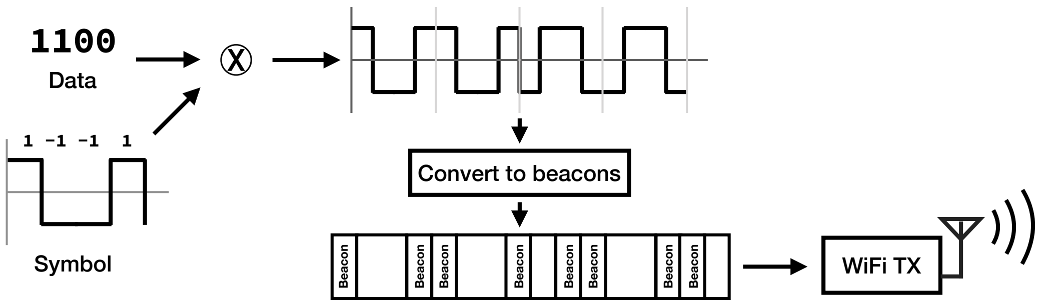

Figure 2: ONPC Transmitter. A symbol is converted into 802.11 beacon frames and then sent wirelessly.

As a result, all ones are transmissions and all negative ones are no transmissions.

For the transmission of the "on" chip, a beacon frame is used because it is a valid 802.11 frame that can be broadcast. The use of a standard WiFi frame ensures that ONPC transmitters do not negatively impact other WiFi devices. The Stayin' Alive application on the transmitter works in the following way: when it detects that the WiFi device is no longer connected to the network, the ONPC protocol begins. The transmitter repeatedly transmits its symbol for a certain amount of time. After transmitting, it pauses and tries to reconnect to any known networks. This process is repeated until the device eventually connects to a network.

ONPC transmitters are unable to sense the channel for other ONPC transmissions before transmitting because the application layer, where the ONPC transmitter is implemented, does not have direct access to the WiFi hardware. This means that multiple ONPC transmissions could potentially overlap with each other. To allow multiple ONPC transmitters to work at the same time, we use orthogonal symbols, similar to code-division multiple access (CDMA). This ensures that only the symbol that the receiver is looking for is detected. Since ONPC's range is longer than standard WiFi, it has more potential for hidden terminal issues. However, since each ONPC transmitter is assigned a unique symbol that is orthogonal to other ONPC transmitters, this allows for multiple ONPC transmitters to transmit at once. We evaluate the use of multiple transmitters in Section 5.5.

One of our design goals is to only use off-the-shelf components for our transmitter and receiver and not modify the WiFi firmware. The challenge with doing this is being able to send frames out in accurate intervals. Timing is key for ONPC to work because the receiver is looking for a certain symbol in the noise measurements. If the timing is off, then the symbols will not match. ONPC is built on the idea that symbols can be sent predictably. If the timing is not accurate, ONPC will not work. Three things could potentially affect the timing of the ONPC transmitter: hardware timing, carrier sensing, and clock skew. We describe these challenges and how we overcame them in the sections below. We also discuss the impact an ONPC transmitter might have on the wireless network.

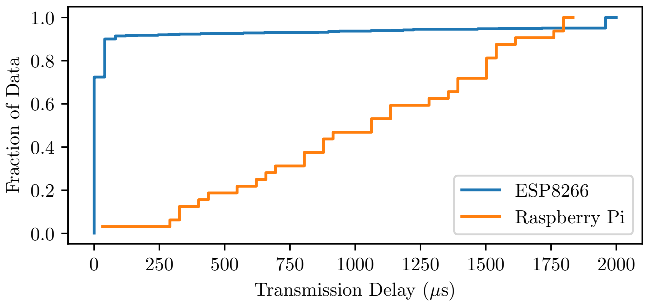

Figure 3: The delay between when a transmission was expected to occur compared to when it actually occurred, with two hardware types: Raspberry Pi and ESP8266

2.2.1 Hardware Timing. The first challenge in designing the transmitter is sending out WiFi frames in a timely and consistent manner. Since the ONPC software is running in the application layer and has no direct access to the WiFi drivers, there is a potential for the system call that transmits the frame to become interrupted or delayed by other system calls. These interruptions cause a delay between when ONPC asks the hardware to transmit a frame and when the hardware actually transmits the frame. To understand the timing characteristics, we build our design on two platforms, a Raspberry Pi [24] and ESP8266 [29]. To transmit 802.11 frames on the Raspberry Pi, we use packet injection while in monitor mode [1]. The SDK for the ESP8266 provides a call for injecting 802.11 frames into the network [11].

We run an experiment to understand and quantify the timing characteristics of these devices. We set the Raspberry Pi and ESP8266 to transmit a 50 byte 802.11 beacon frame and then pause for 1000 μs. We connect these devices to a spectrum analyzer by RF cable. Figure 3 shows a CDF of the transmission accuracy of each device. The y-axis shows the

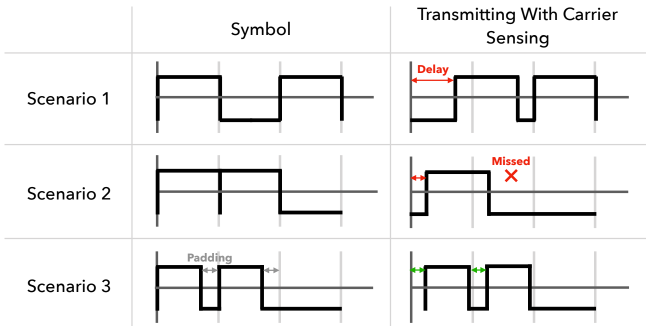

Figure 4: Different scenarios showing how carrier sensing can change the transmitted symbol. By reducing the transmission time relative to the chip time, carrier sensing does have as much of an effect on the symbol.

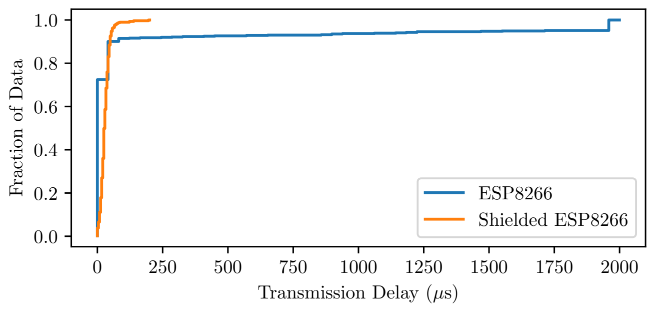

Figure 5: The CDF of the delays between when a transmission was expected to occur compared to when it actually occurred. There are two configurations, an ESP8266 and an ESP8266 that has been shielded in a metal box.

delay between when the device transmitted a frame compared to when it was expected to transmit. Ideally, the delay would be zero, meaning a frame is transmitted exactly when expected. The ESP8266 provides much better timing characteristics compared to the Raspberry Pi. With the ESP8266, 90% of its delays are less than 250 μs. The results show that the Raspberry Pi does not provide as much timing accuracy as the ESP8266. This confirms our intuition, given that the Raspberry Pi is running Linux, which is not a real-time OS, and the ESP8266 is an embedded system so interrupts and delays are kept to a minimum. As a result of this experiment, we implement ONPC on the ESP8266. More details of this implementation are given in Section 4.

2.2.2 Carrier Sensing. Carrier-sense multiple access (CSMA) is a feature of the 802.11 MAC which tries to minimize collisions between transmitting devices by waiting a random amount of time after the channel is clear. As a result of carrier sensing, when a frame is "transmitted" in the application layer, it might not actually be transmitted by the network adapter until sometime later.

Carrier sensing is an important aspect of the 802.11 MAC which helps to prevent collisions, but it is a problem for the timing constraints of ONPC, which requires the symbol pattern is followed. With ONPC, if a transmission is delayed because of carrier sensing, the symbol could be changed enough to be no longer detected by the receiver. Also, with how the ESP8266 SDK implements the frame injection call, if a transmission is currently pending or the device is transmitting and another frame is injected, the system call will fail. From the SDK, there is no way to know if a transmission is currently going on. This can lead to missed transmissions. Figure 4 shows two examples of cases where carrier sensing would change the symbol. In scenario one, the symbol `1 -1 1` gets transmitted like `-1 1 1` because the first transmission gets delayed by carrier sensing. In the second scenario, the symbol `1 1 -1` gets transmitted like `1 -1 -1` because the second transmission fails because the first transmission is still occurring.

We run an experiment to understand the effect of carrier sensing. We connect the ESP8266 to a spectrum analyzer via RF cable, transmitting a 50 byte 802.11 beacon frame and then pausing for 2000 μs, repeated. Though the transmitter is wired to the spectrum analyzer, the transmitter is still sensing transmissions from other devices, leading it to delay its transmissions because of carrier sensing. Next, we try to shield the ESP8266 from all transmissions by putting the ESP8266 in a metal box while still connected to the spectrum analyzer. By putting the ESP8266 in a metal box, we cut down on transmissions the device might be sensing. Figure 5 shows the results for the two device configurations. All of the shielded ESP8266's offsets are below 250 μs, whereas 10% of the unshielded ESP8266's offsets are greater than 250 μs. These higher offsets are caused by carrier sensing.

One option to deal with carrier sensing would be to disable it. This would allow the ONPC transmitter to transmit frames without first checking to see if another device is transmitting, reducing the offset time and making transmissions predictable. However, our transmitter would become harmful to the network, potentially interfering with other transmitters. Also, this goes against our goal of not modifying WiFi drivers.

To account for carrier sensing, we add padding to the chip time. Rather than making the transmission time the same as the chip time, we add extra time to the chip period so that if the transmission gets delayed, it will still be within the chip time window. This allows us to handle a certain amount of carrier sensing delay and still keep the symbol the same. This is shown in Figure 4, scenario 3. The amount of padding added to our system is described in Section 5.1.1.

2.2.3 *Clock Skew.*
The third challenge regarding timing is clock skew. Since the transmitter and receiver are using different clocks and there is no synchronization between them, clock skew could lead to problems. As the two clocks drift apart, the received symbol would no longer match the transmitted symbol.

To understand the problem of clock skew, we run the same experiment from the previous section, using the shielded ESP8266 to minimize the effect of carrier sensing. We measure the clock skew to be 30.9 μs skew per second. Given the time it takes to transmit our symbol, 13.73 seconds (details are given in Section 4.2), this amount of clock skew will not have a large impact on our system. Since we are already adding padding to the chip time to help deal with carrier sensing, this will also help with clock skew.

2.2.4 *Network Impact.*
By ensuring we are using standard WiFi hardware, without any modifications, an ONPC transmitter will have no additional negative effect on a WiFi network compared to an ordinary WiFi transmitter. The ONPC transmitter uses carrier sensing so it will generally not interfere with other transmitters. Other devices on the same channel as the ONPC transmitter will have to share the channel which could potentially cause a slowdown. We study the effect of ONPC on WiFi traffic in Section 5.6.

2.3 Access Point

The challenge with the access point is getting timely noise value measurements. To achieve this, we use DD-WRT [10] running on our router. DD-WRT is an alternative firmware for wireless routers. It provides additional tools compared to stock firmware for access points, such as command-line tools to monitor the network and an SSH server. One of our design goals is not to modify the firmware or drivers of the WiFi hardware, and we feel that installing DD-WRT does not violate this goal. We are using a standard DD-WRT firmware with no modifications necessary to make ONPC work. Using DD-WRT gives us access to command line tools that can be run over ssh.

DD-WRT provides a proprietary tool called `wl` which provides commands for managing and monitoring the wireless interface of the access point. One of the commands provides noise measurements. From what we can gather from the documentation, it takes a certain number of samples from the antennas and returns three numbers, the received signal strength for each antenna. The time it takes to complete the system call can range from 5.0 ms to 6.5 ms, depending on the load (amount of wireless transmissions currently active) of the access point. We assume that the difference between the time it takes to collect the samples and the time it takes for the system call is due to system call overheads. This means that there is a certain amount of time when we do not receive samples. To deal with this, we increase the size of the 802.11 beacon frame the transmitter is transmitting to the maximum frame size the ESP8266 SDK can inject into the network, 1400 bytes. Measuring this transmission on a spectrum analyzer, it takes 11.42 ms to transmit. We set our chip time to 13.73 ms (see Section 5.1.1 for details), slightly larger than the transmit time, to account for the random carrier sensing delay. This ensures that we will not miss a chip transmission between system calls.

2.4 ONPC Receiver

The ONPC receiver pulls measurements from the access point and processes the measurements in the following manner, as shown in Figure 6. The receiver first runs the noise measurements through a limiting function, which is described in more detail in the section below. Next, it correlates the limited measurements with the known symbol of the disconnected device. If the correlation crosses a certain threshold, τ, the receiver starts to interpret the values as bits, a positive value as a one and a negative value as a zero. τ is the number of standard deviations away from the mean a correlation must be to be considered a bit.

ONPC supports multiple transmitters because each transmitter has its own pseudorandom symbol. The pseudorandom symbol looks like noise if you are not looking for that particular symbol.

One challenge we encountered when designing the receiver is decoding an ONPC transmission when non-ONPC WiFi devices transmit nearby the access point. ONPC is designed to extend the range of communication, and as a result, the protocol looks for slight variations in the noise measurements. When a transmitter close to the access point transmits, the received power at the access point jumps up, causing large variations in correlation relative to when the non-ONPC WiFi device is not transmitting. Intuitively, the nearby non-ONPC WiFi transmitter has more weight compared to a distant ONPC transmitter because the received signal is so much higher. To deal with this, instead of correlating the raw received signal strengths with the symbol, run the signal strength measurements through a limiting function that applies a ranking data transformation to the received signal strengths. This ensures that all values from 1 to n are represented, where n is the number of signal strength samples per symbol. This ensures that the distribution of signal strength measurements after they are transformed is the same, regardless if there is a nearby transmitter that produces large signal strength readings. We also subtract the mean of the ranked samples and divide by the total number of samples to get values between -1 and 1. We evaluate the use of ranking compared to the raw samples in Section 5.3.

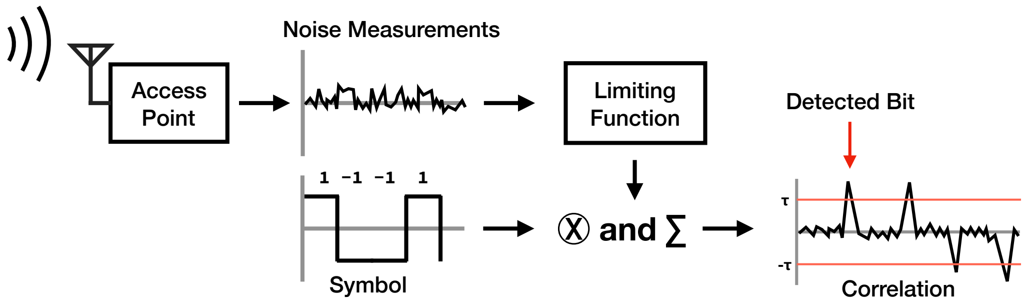

Figure 6: Detecting ONPC transmissions. The access point collects noise measurements and the ONPC receiver runs the noise measurements through a limiting function and correlates the measurements with a known symbol to detect symbols.

3 STAYIN' ALIVE DESIGN

Stayin' Alive is designed to know if a device is still alive even if it is disconnected from WiFi. As a manager of an IoT system, it is impossible to know if a device that is unresponsive is because it is disconnected from WiFi or because something else occurred, like a power outage or hardware malfunction. We found that in long-term deployments, devices disconnect from WiFi often. Of 30 sensors we had deployed for over a year, sensors disconnected from WiFi on average 6.75 times per month per sensor and the average disruption time was 22 minutes, with some disruptions lasting for hours. These times represent times when the manager of the IoT system did not know the state of the devices. Our experience agrees with the conclusion of Hnat et al. [15], that "homes are hazardous environments" for wireless devices.

Stayin' Alive uses the ability of ONPC to transmit data beyond the range of normal WiFi. Stayin' Alive runs on top of the ONPC transmitter and ONPC receiver. On the WiFi device, when Stayin' Alive detects that the WiFi device has lost its connection, it starts transmitting data using ONPC. The Stayin' Alive application running on the ONPC receiver interprets the received data and notifies the network manager that the device is disconnected from WiFi but still alive.

4 IMPLEMENTATION

We briefly discuss the steps to develop and implement ONPC. We use a synthetic wireless trace generator and controlled experiments to ensure the implementation of ONPC works correctly and inform our design. Then we give the details of ONPC's implementation on off-the-shelf hardware for each component, to show ONPC's viability. Finally, we describe the implementation of Stayin' Alive.

4.1 ONPC Development

As a proof of concept and to help understand and build the ONCP transmitter and receiver, we first build a framework to generate synthetic wireless traces of an ONPC transmission. To help select realistic parameters, we use noise measurement traces taken by a spectrum analyzer. Generating synthetic traces of an ONPC transmitter allows us to change parameters about the received signal, such as the power of the transmitter or how much interference is in the network and run it against the ONPC receiver software. This helps us to understand the boundaries of what the ONPC receiver software is capable of handling as well as debug the ONPC receiver algorithm.

After using the synthetic wireless trace generator, we develop and implement the ONPC transmitter using off-the-shelf hardware (Section 4.2). We connect the ONPC transmitter to a spectrum analyzer through RF cables, adding different amounts of attenuation to emulate distance. Using the high-resolution samples of the spectrum analyzer, we run the ONPC receiver software and validate that the transmitter is working as expected at various amounts of attenuation. Next, we implement the ONPC receiver (Section 4.4) using off-the-shelf hardware, pulling samples from a commercial AP (Section 4.3). We connect the ONPC transmitter to the AP through RF cables with different amounts of attenuation. This allows us to ensure ONPC is working correctly at various transmission power levels while minimizing the effects of carrier sensing and interference. Each of these steps helps to inform the design and implementation of ONPC.

4.2 Transmitter

The transmitter is built using a Wemos D1 mini Pro [29], which is based on the ESP8266. We use C code to interact directly with Espressif's SDK for the ESP8266. We set up a hardware timer that fires periodically. This interval represents the chip rate of the transmitter. When the hardware timer fires, either a beacon frame is transmitted into the network (represents a one) or nothing happens (represents a negative one), depending on the chip that is being transmitted. We use the `wifi_send_pkt_freedom` call to inject frames into the network.

`wifi_send_pkt_freedom` has a few important limitations. First, it can only send frames that are less than or equal to 1400 bytes. Second, if called during a transmission, the call will fail (return -1). This can occur when carrier sensing has delayed the transmission such that the previous frame is still being transmitted when the next frame needs to be sent. From the SDK, there is no way to determine if the device is transmitting and the call will fail.

In our implementation, we transmit beacon frames that are 1400 bytes, which takes 11,423 μs to transmit and select a chip period of 13,423 μs. This gives 2000 μs of padding to account for carrier sensing. We select 1400 bytes as the beacon frame size because this is the largest frame size the ESP8266 could inject into the network. The time it takes the access point to collect samples ranges from 5.0 ms to 6.5 ms (see Section 2.3). By sending a frame that is 1400 bytes and takes 11,423 μs to transmit, roughly two samples will be collected during a transmission. In Section 5.1.1, we experiment with different chip periods to find which one reduces the effect of carrier sensing. To enable transmissions that can be recovered with low SNR, we use a symbol with a length of 1023 ($2^{10} - 1$). With these parameters, it takes 13.73 seconds to transmit the whole symbol.

4.3 Access Point

For our access point, we use the Netgear R7000 wireless router, running DD-WRT. DD-WRT provides ssh access so that management commands can be run. We use the `wl phy_rxiqest` command to obtain received power estimates. Three values are returned from this command, one RSS value for each antenna. `wl phy_rxiqest` has eight poorly documented options. Of the options, we use the `-r` flag, which allows us to select coarse or fine grain measurements. Course measurements are rounded to the nearest integer, and fine measurements return RSS values to the nearest quarter dBm. We select fine grain measurements to provide more accuracy. We also use the `-s` flag, which allows us to select how many samples should be considered when running the command. Possible values range from 2^{10} to 2^{15}. We select 2^{15} so that we can capture more of the channel in one system call. We try the other parameters, but they do not provide any benefit for our particular application.

The time it takes to complete the call to `wl phy_rxiqest` varies depending on the amount of wireless activity. We assume this is because the access point is decoding frames which slows down the call. In our experience, values range from 5.0 ms to 6.5 ms. This call is the limiting factor in our system and dictates the rate at which we can send our symbol.

4.4 Receiver

We implemented the receiver as a Python application. The application logs into the AP through ssh, collects samples (stored locally on the AP as a file), and transfers the sample file for processing. Before correlating the noise measurements with the symbol, we process the data. Since ONPC is looking for weak signals in the noise measurements, a nearby transmitter can negatively affect our algorithm. We pass the samples through a limiting function, as discussed in Section 2.4. The effectiveness of this filtering is shown in Section 5.3. Next, the receiver correlates the limited samples with the symbol of the device it is looking for. Finally, it determines a threshold, τ, at which a symbol will be detected. τ is set to 4.0 standard deviations away from the mean of the correlation values. If a correlation value crosses this threshold, then a symbol is detected.

In deployments, the ONPC receiver is Ethernet connected to home's access point. Since processing does not need to be done in real-time, the hardware requirements are low for the receiver. We have run the receiver on a Raspberry Pi as well as a 2016 MacBook Pro.

Note that the receiver does not have to be local to the WiFi devices or access point, nor does a receiver have to be tied to one access point. If there was a way of securely getting noise measurements from an access point remotely, the receiver could run in a different location. One could imagine a cloud receiver that monitors all deployed sensors. We leave this for future work.

4.5 Stayin' Alive

Stayin' Alive works in this general flow. First, it must determine what WiFi devices are not currently functioning. What it means for a WiFi device not to be functioning and how to determine that depends on the WiFi device and deployment architecture. For our deployed sensors, which send data every minute, we query a database to see if a sensor has uploaded data recently. After a certain amount of time with no new data, Stayin' Alive determines that the sensor is offline. Stayin' Alive looks up the symbol that corresponds to the offline sensor through a database query and runs the ONPC receiver, passing the symbol. If the receiver detects the device's symbol, Stayin' Alive knows that the WiFi device is still powered on and is functioning normally but is disconnected from WiFi. This information is reported to a database. To reduce the possibility of a false positive, we ensure the symbol is received multiple times and that they are correctly spaced apart.

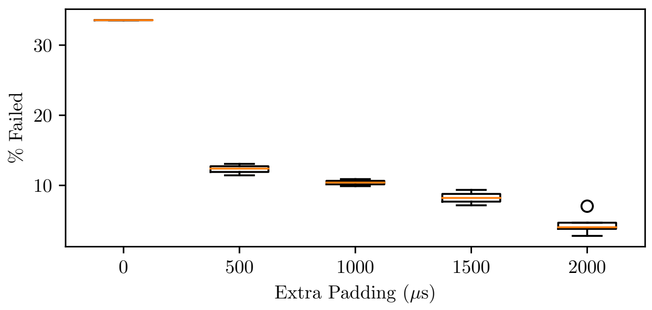

Figure 7: The percent of frames failed to send across different padding measurements (μs).

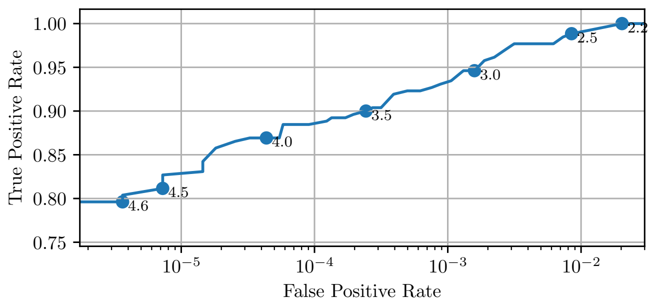

Figure 8: Receiver operating curve for ONPC. Points along the curve represent different threshold values.

5 EVALUATION

We evaluate various aspects of ONPC and demonstrate that ONPC works beyond the range of WiFi in an indoor and outdoor environment. We also demonstrate that ONPC's MAC functions with multiple ONPC transmitters and can function even when near powerful transmitters.

5.1 ONPC Parameters

5.1.1 Padding. As discussed in Section 2.2.2, we add padding between the transmission and chip period to negate the effect of CSMA. If CSMA delays a transmission, the extra padding helps the delay not to affect the next transmission. However, because of WiFi's CSMA behavior, there is no way of knowing the right amount of padding for all situations. Depending on how many WiFi clients are present, there is still a chance that the amount of padding is not enough to negate the effects of CSMA. For our system, it becomes a trade-off of adding padding to reduce the effects of CSMA and lowering our data rate. To understand this trade-off, we run an experiment in a moderately crowded WiFi environment, evaluating transmissions failures as a result of carrier sensing. The system call fails if a transmission is currently active when injecting a frame into the network. The experiment consists of constantly transmitting frames back to back. We count the number of times a transmission fails with different amounts of padding. The results are summarized in Figure 7. As the amount of padding increases, the percent of failed transmissions decrease. For our system, we select a padding of 2000 μs which has a loss rate of about 5%, since it is enough padding to reduce most the effects of carrier sensing.

5.1.2 Threshold. In our system, when the correlation is above a threshold, it is considered a detected symbol. We set this threshold to τ standard deviations above the mean. Determining how the threshold is picked affects the detection rate and false positive rate. To understand the trade-offs, we run ONPC on a set of data with different threshold values, generating a receiver operating curve, shown in Figure 8. The threshold value can be picked based on the needs of the application using ONPC and its ability to tolerate false positives. For Stayin' Alive, we use a threshold value, τ, of 4.0.

5.2 Data Rate

Given the parameters used in Section 4, one symbol (PN code) is transmitted every 13.7 seconds. There are approximately as many possible codes as the PN code length. For example, the Walsh-Hadamard code set for length N has N orthogonal codes. A receiver could convey log_2 1024 = 10 bits of information by the code it chooses to transmit. In this case, the bit rate is 10 / 13.7 = 0.73 bps. Alternatively, we could select two codes and transmit some linear combination of these two codes using a modulation like M-ary QAM. In this case, we would be sending log_2 M bits per symbol or log_2 M / 13.7 bps.

While these rates may seem slow, particularly in comparison to standard 802.11 protocols, we note that the application of ONPC will not be to replace WiFi. For Stayin' Alive, as an example, the problem of identifying whether a WiFi-enabled sensor is powered on or off is one that takes perhaps days, and human action, to resolve. For this application, 13.7 seconds is a very quick resolution.

5.3 Nearby Transmitters

ONPC is designed to find weak signals in noise and, as a result, is sensitive to powerful transmitters. This can occur when a WiFi device is close to the access point, but the ONPC transmitter is far away. The close wireless client will have a significant impact on the RSS measurements of the access point. This can be seen in Figure 9. The top graph shows the raw noise measurements, and the middle graph shows the correlation between raw noise measurement and symbol. The green dashed line represents the correlation threshold, and the stars are when a symbol is detected, and a red x is a false positive. In this data collection, the ONPC transmitter is constantly transmitting so a symbol should be

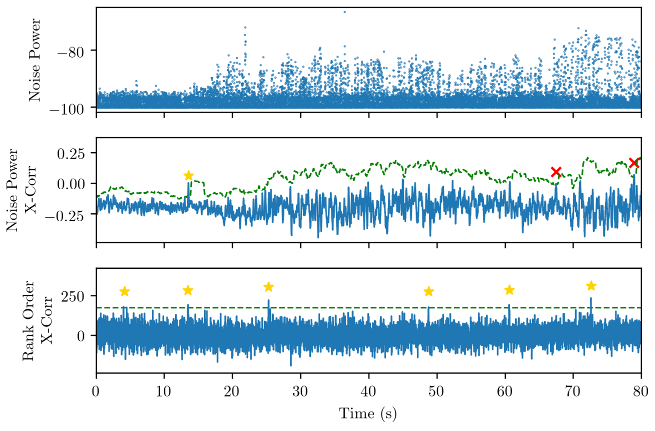

Figure 9: The top graph is noise power measurements as received by the access point with respect to time. A transmission occurs part way through the data collection. The bottom two graphs are the correlation with the symbol using the noise power measurements and ranked noise power measurements, respectively.

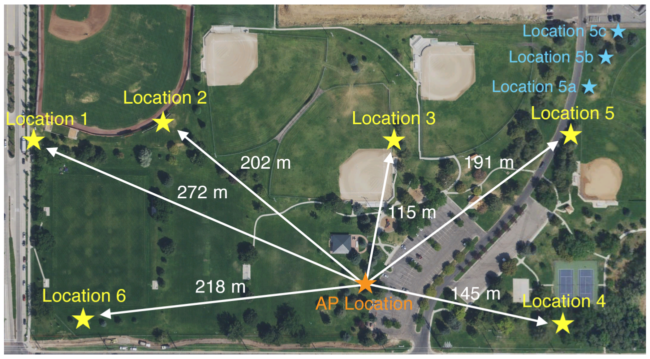

Figure 10: A map of the locations where ONPC transmitters are placed relative to the AP location. WiFi was unable to communicate at each location.

detected every 13.7 seconds. However, using the raw noise measurements to cross-correlate with the symbol, only one symbol is detected. To solve this problem, we limit the raw noise power measurements using the rank order method, as described in Section 2.4. Once limited, ONCP can detect symbols even during a transmission, as shown in the bottom graph of Figure 9. ONPC can detect the transmitted symbol, even when another transmitter is transmitting. All other results in this section use the rank order limiting function.

5.4 Range Enhancement

To test that ONPC works beyond the range of WiFi connectivity, we separate an ONPC transmitter from an access point until the transmitter consistently remains disconnected from

Table 1: The efficiency of ONPC at different locations.

Loc	Distance (m)	Symbols Tx	Symbols Rx	False Positives
1	272	29	29	0
2	202	29	27	0
3	115	30	8	3
4	145	29	29	4
5	191	29	29	1
6	218	29	26	2

WiFi. We program the ESP8266's onboard LED to flash when the device is disconnected from WiFi. A transmitter that is disconnected from WiFi will try to reconnect for 30 seconds. If unsuccessful, it will run ONPC for 2 minutes before trying to reconnect again for 30 seconds. This process is repeated. We initially run our experiments in an outdoor location to reduce the effect of interference of other WiFi devices. Though this is not the target environment for our application, Stayin' Alive, this will allow us to understand how well ONPC works compared to WiFi in an environment with less interference compared to an indoor environment. Even in this outdoor location, there are seven APs on 802.11 channel 1, where ONPC is running. We repeat this experiment multiple times, each time picking a different location for the transmitter such that the transmitter is close to the edge of WiFi connectivity, but not connected. The locations are shown in Figure 10. At each location tested, ONCP was successful where WiFi was unable to connect. Table 1 shows the symbol reception rate and false positives for the transmitter at the six locations. Except for location 3, each location's symbol reception rate is 90%. These results show that ONPC works beyond the range of WiFi. We note that the distances of our experiments are highly dependent on many factors such as the environment, interference, hardware, etc. The absolute distances of these experiments are not important but rather that the range of ONPC extends past WiFi's range.

Next, we test the limits of ONPC at location 5. We want to understand how much further ONPC works past the range of WiFi. Starting at location 5, we move to three locations away from the access point. These locations can be seen as the blue text (top right corner) in Figure 10. The results are shown in Table 2. At 258 m (67 m beyond location 5), the symbol reception rate starts to drop. This shows that ONPC extends the range of WiFi by a significant distance.

5.5 Multiple ONPC Transmitters

To test multiple ONPC transmitters, we first set up one ONPC transmitter and measure the percent of symbols received correctly. Next, we add another transmitter at approximately the same distance from the receiver that transmits an orthogonal

Table 2: The efficiency of ONPC at different locations.

Loc	Distance (m)	Symbols Tx	Symbols Rx	False Positives
5	191	29	29	1
5a	220	29	28	2
5b	245	29	28	4
5c	258	29	20	4

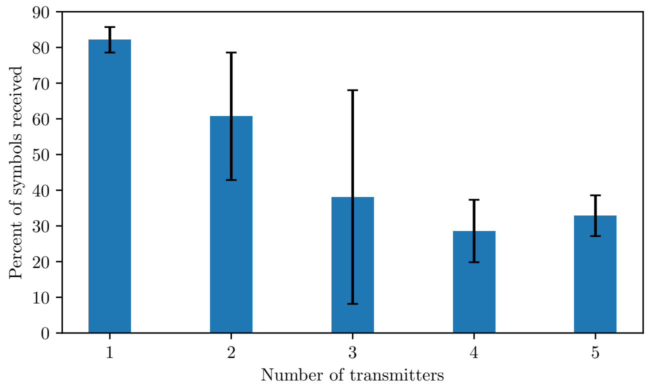

Figure 11: Overall percentage of symbols received as more ONPC transmitters are transmitting at the same time.

symbol and measure the percent of symbols received from both transmitters. We continue this process until we have five ONPC transmitters transmitting orthogonal symbols at the same time. The results are shown in Figure 11. The percentage of received symbols decreases as the number of transmitters increases. However, for our use case, as long as one symbol is received occasionally, that is enough to know if a sensor is still functioning. These results show that ONPC works for up to five transmitters for our use case.

Next, we test how well ONPC works with two ONPC transmitters are transmitting at the same type (orthogonal symbols), but with a large disparity between the signal strengths of the transmitters. The ESP8266 devices we use as ONPC transmitters have controllable transmit powers. The default value for transmit power is 82, which is the maximum value.[1] Using a spectrum analyzer, we determine that setting the power to 41 drops the signal strength by 10 dB. We can create a signal strength disparity of 10 dB by placing two ONPC transmitters next to each other and equal distance from the receiver. One of the transmitters is set to the default transmit power and the other transmitter is set to 42. Under these conditions, the ONPC receiver can receive the symbol of the higher SNR transmitter 96.6% of the time and 24.1% of the time for the lower SNR transmitter. This shows that even with a large disparity in signal strength, the ONPC receiver is still able to detect symbols from both transmitters.

We do not evaluate ONPC transmitters in a hidden terminal scenario because ONPC does not depend on CSMA for correct operation. We use orthogonal symbols to allow multiple transmitters to transmit at once, regardless of if they are in a hidden terminal scenario or not.

5.6 Effects on non-ONPC communication

To understand the effect ONPC has on normal WiFi transmissions, we run the following experiment. We place our ONPC transmitter so that it is far enough away not to be connected to WiFi, but running the ONPC protocol. We use a wirelessly connected laptop as a `iperf` client and an Ethernet connected Raspberry Pi acts as an `iperf` server. We run `iperf` in UDP mode at different speeds (unlimited, 5 Mbps, and 1 Mbps) with and without ONPC running. We place the

[1]This value is defined in the ESP8266 SDK and does not represent a specific unit of measure.

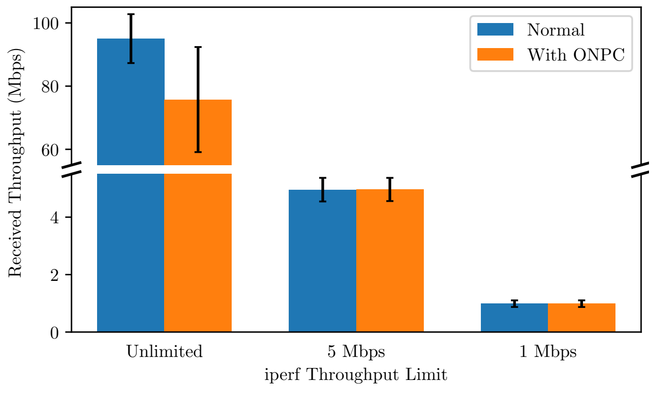

Figure 12: The effect of ONPC when a WiFi transmitter is close to the AP.

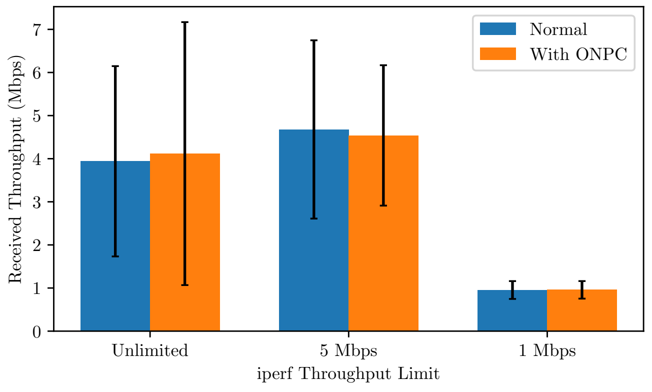

Figure 13: The effect of ONPC when a WiFi transmitter is far from the AP.

iperf client laptop at two locations: next to the AP and next to the ONPC transmitter (67 meters away). We run iperf for 2.5 minutes, collecting 150 samples, in each condition.

When the iperf client laptop is next to the AP, ONPC only affects WiFi transmissions when a client laptop is continuously using the channel (iperf set to unlimited). The results are shown in Figure 12. The maximum achievable throughput, for this particular configuration, when running ONPC deceases by 20.3% compared to when ONPC is not running. However, when running at lower data rates, 5 Mbps and 1 Mbps, ONPC does not affect the data rate. The decrease of 20.3% is due to the client laptop carrier sensing the ONPC transmitter's transmissions. Since the ONPC transmitter is transmitting a beacon frame half the time, we expected to see a decrease of throughput of about 25%. When the iperf client laptop is far from the WiFi AP, next to the ONPC transmitter, ONPC does not affect the WiFi transmissions, as seen in Figure 13. The reason we see no effect on throughput in this situation when sending unlimited data is because the overhead of lost frames is greater than the impact of ONPC. The results show that ONPC as no effect on most wireless transmissions. Only under very specific scenarios can the effect of ONPC be seen. We believe the impact of ONPC will be unnoticeable for general use cases since research has shown that home's WiFi networks are underutilized [23]. If needed, ONPC can be used sporadically rather than continuously, to minimize its effects.

5.7 House

We deploy ONPC in a residential house. The house has three floors, including a basement. The access point is located in the basement and the WiFi device on the top floor, on the opposite side of the house such that the WiFi device is no longer connected to the AP in the basement and the ONPC transmitter runs. We estimate that the distance between these two devices is roughly 13 m, across three floors and many walls. We run ONPC on channel 1, where there are five other access points (neighbors) transmitting on the same channel. We run ONPC continuously and collect data for 6 minutes.

In this experiment, the symbol reception rate is 51.7%, with no false positives. Two-thirds of the way through the experiment a nearby WiFi device transmits, causing a significant change in correlation. This reduced the number of symbols detected in this experiment. These results show that ONPC works beyond the range of normal WiFi in a home environment.

6 RELATED WORK

Two WiFi standards already exist that support lower data rates and longer ranges: HaLow (802.11ah) [4] and White-Fi (802.11af) [12]. Both can increase their range in part because of their lower center frequencies experience lower penetration losses. HaLow uses the 900 MHz band (902-928 MHz in the US), and White-Fi runs on the TV white space frequencies, 54 to 698 MHz. Other wireless protocols exist, such as LoRa [26] and ZWave [9], that make similar trade-offs between increasing range and reducing data rate. These protocols use different PHY layers to help increase the range. As a result of changing the PHY and protocol, a different radio is required, making these standards not compatible with the standard WiFi APs used in typical residential houses (i.e., 802.11n). Since these technologies are not widely deployed in homes, to use one of these technologies, new access points would need to be deployed in each participant's home. Human subject research studies are usually cost constrained, so buying new equipment for each home is unreasonable. For our application, we are using a house's WiFi network, and when a sensor is unable to communicate, we switch to ONPC. Since we are building on top of standard WiFi, our protocol works with standard WiFi transmitters and receivers, adding no extra cost.

An alternative to ONPC would be to use an ad hoc wireless network. Rather than each device communicating with an AP, devices communicate with each other, creating a mesh network. This allows wireless devices to chain together to transmit data rather than connecting to one central node (the AP). This approach would potentially allow a device that is too far away from an AP to transmit data to its nearby neighbors; however, this only works when dealing with multiple wireless devices. If you have only one wireless device deployed, then this approach does not help. It also requires that WiFi devices are located in such a way that they are in range of each other. Lastly, even with a mesh network, a critical wireless link between devices can be weak and periodically fail. In such a case, ONPC would be beneficial.

ONPC shares similarities with WiFi backscattering [18]. WiFi backscattering allows a low powered RF device to transmit and receive data. The RF device reflects WiFi signals causing changes in RSS to transmit data and detects energy during WiFi transmissions to receive data. Although ONPC and WiFi backscattering both use changes in RSS and WiFi transmissions to convey information, ONPC differs from WiFi backscattering in three major ways. First, WiFi backscattering uses specialized hardware to detect energy during WiFi transmissions, whereas ONPC uses off-the-shelf components to detect WiFi transmissions. Furthermore, WiFi backscattering requires extra hardware to reflect WiFi signals. ONPC is implemented purely in software and requires no extra hardware. Second, WiFi backscattering uses a CTS-to-Self frame to clear the channel while a WiFi device transmits to the RF device, ensuring that no other devices will transmit during this time. ONPC deals with other transmissions directly by accounting for carrier sensing and does not require the channel to be clear to function. Third, the focus of these works is different. ONPC is about extending the range of WiFi, whereas WiFi backscattering focuses on a low energy RF device communication at short ranges. This leads to different system design decisions.

Long Range WiFi (LR WiFi) has been researched and deployed in various places around the world, where Internet connectivity is not readily available, such as rural areas [7]. For example, M. Zennaro et al. deployed long-range WiFi in Malawi at a distance of 162 km [21]. S. Unni et al. set up LR WiFi for over-the-sea communication [28]. These efforts require directional antennas to be precisely pointed to each other, and modifications to 802.11's MAC protocol. ONPC requires no specialized hardware and no changes to the 802.11's MAC protocol. Also, ONPC's goals are different from LR WiFi. ONPC is designed to supplement current wireless deployments, not supplant long-range protocols. ONPC can complement LR WiFi; for example, Stayin' Alive may benefit an LR system that is reliable at setup by providing more information about the state of the other end of the link, when changes in weather as described in [21] make the channel unsuitable for WiFi packet reception. Also, ONPC could communicate some relative location or angle information in cases when one end's mobility makes the link fail, as a means to automatically reposition the directional antennas.

Interference detection is particularly important in cognitive radio for the detection of primary users, and energy detection methods are proposed for this purpose [5][27], including methods referred to as transmitter detection, cooperative spectrum sensing, and interference based detection [2]. Interference avoidance is important for any multiple access system, and energy detection is designed to work for arbitrary modulation types [22][8], for example, to allow a ZigBee network to change channel when it detects WiFi interference. While ONPC uses noise power (or energy) as a measurement, it is not doing so for interference avoidance. Instead, ONPC is re-purposing a sensing operation already available in WiFi hardware to enable a different communication protocol. In this sense ONPC is more similar to cross-technology communication (CTC) systems in which a transceiver designed for one protocol is able to send data to a receiver of a different protocol, e.g., BLE transmitter to Zigbee receiver [17]. However, ONPC is not serving to cross between two existing protocols; both ends are WiFi devices. ONPC alters both transceivers for a new protocol on top of WiFi to enable longer range communication, which is not achieved in reported CTC systems. By altering both WiFi transmitter and receiver to operate a new protocol, ONPC is similar to [25], which uses changes in the transmit signal filter to encode extra (secret) bits in the channel state information. These secret bits require WiFi packet reception, thus the secret bits cannot be decoded at a longer range than WiFi itself.

ONPC shares similarities with infrared light communications [30]. In such systems, data can be transferred in a unipolar fashion (on or off), similar to how ONPC transmits a beacon (on) or does not transmit a beacon (off). Infrared light communications must also deal with interference from other light sources, like fluorescent lights, but the infrared channel is less crowded than the 2.4 GHz band, and infrared interference sources tend to be stationary (time-invariant statistics). Since infrared transmitters and receivers are designed to communicate via turning an incoherent power source on or off, they can achieve a significantly higher data rate; while ONPC must deal with hardware that is not designed for this purpose. Such systems tend to focus on the speed of communication, whereas ONPC is focused on the distance of communication.

Other work has been done on using WiFi as an out-of-band communication or covert channel. Many of these works find places in the 802.11 header to insert covert information [13][6] or use the timing between packets [16][3]. Though ONPC uses timing to convey information, ONPC's goal is to extend range, so it takes a different approach by implementing a spread spectrum like technique using 802.11 frames.

7 CONCLUSION

In this paper, we present the novel protocol ONPC, which extends the range of WiFi using standard WiFi. ONPC uses off-the-shelf WiFi hardware and piggybacks off of existing technologies, making ONPC inexpensive to implement and widely deployable. Through our evaluation of different aspects of ONPC, we demonstrate that it works beyond the range of WiFi. We show that ONPC is robust against nearby transmitters and can detect transmissions even when another transmitter is transmitting. We demonstrate that multiple transmitters can use ONPC at the same time. We successfully deployed ONPC in a residential house. Using ONPC, we design and deploy an application, Stayin' Alive, that assists remote system managers in determining whether a device is still functioning, even if it cannot communicate over WiFi, solving the observability challenges of devices deployed in difficult-to-access locations.

ACKNOWLEDGMENTS

Research reported in this publication was supported by NIBIB of the US NIH under award number 1U54EB021973-01. We are grateful for the help from our shepherd, Dr. Ashutosh Sabharwal, and anonymous reviewers which helped to greatly improve the final version of this paper.

REFERENCES

[1] 2018. Linux Wireless - Monitor Mode. https://wireless.wiki.kernel.org.

[2] Mahmood Abdulsattar and Zahir A. Hussein. 2012. Energy Detection Technique for Spectrum Sensing in Cognitive Radio: A Survey. *International journal of Computer Networks and Communications* 4 (09 2012), 223–242. https://doi.org/10.5121/ijcnc.2012.4514

[3] R. Archibald and D. Ghosal. 2012. A Covert Timing Channel Based on Fountain Codes. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. 970–977. https://doi.org/10.1109/TrustCom.2012.21

[4] S. Aust, R. V. Prasad, and I. G. Niemegeers. 2012. IEEE 802.11ah: Advantages in Standards and Further Challenges for sub 1GHz Wi-Fi. *Communications (ICC), IEEE International Conference on* (2012).

[5] Danijela Cabric, Artem Tkachenko, and Robert W. Brodersen. 2006. Experimental Study of Spectrum Sensing Based on Energy Detection and Network Cooperation. In *Proceedings of the First International Workshop on Technology and Policy for Accessing Spectrum (TAPAS '06)*. ACM, New York, NY, USA, Article 12. https://doi.org/10.1145/1234388.1234400

[6] Telvis E. Calhoun, Xiaojun Cao, Yingshu Li, and Raheem Beyah. 2012. An 802.11 MAC layer covert channel. *Wireless Communications and Mobile Computing* 12, 5 (2012), 393–405. https://doi.org/10.1002/wcm.969

[7] Kameswari Chebrolu, Bhaskaran Raman, and Sayandeep Sen. 2006. Long-distance 802.11B Links: Performance Measurements and Experience. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MobiCom '06)*. ACM, New York, NY, USA, 74–85. https://doi.org/10.1145/1161089.1161099

[8] Chulho Won, Jong-Hoon Youn, H. Ali, H. Sharif, and J. Deogun. 2005. Adaptive radio channel allocation for supporting coexistence of 802.15.4 and 802.11b. In *VTC-2005-Fall. 2005 IEEE 62nd Vehicular Technology Conference, 2005.*, Vol. 4. 2522–2526. https://doi.org/10.1109/VETECF.2005.1559004

[9] Sigma Designs. 2018. ZWave. http://z-wave.sigmadesigns.com.

[10] embeDD GmbH. 2018. DD-WRT. https://dd-wrt.com.

[11] Espressif. 2018. ESP8266 Non-OS SDK API Reference. https://www.espressif.com/.

[12] Adriana B. Flores, Ryan E. Guerra, Edward W. Knightly, Peter Ecclesine, and Santosh Pandey. 2013. IEEE 802.11af: A Standard for TV White Space Spectrum Sharing. *IEEE Communications Magazine* 51 (2013), 92–100.

[13] L. Frikha, Z. Trabelsi, and W. El-Hajj. 2008. Implementation of a Covert Channel in the 802.11 Header. In *2008 International Wireless Communications and Mobile Computing Conference*. 594–599. https://doi.org/10.1109/IWCMC.2008.103

[14] Solomon W. Golomb and Guang Gong. 2004. *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar.* Cambridge University Press, New York, NY, USA.

[15] Timothy W. Hnat and et al. 2011. The Hitchhiker's Guide to Successful Residential Sensing Deployments. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. ACM, New York, NY, USA, 232–245. https://doi.org/10.1145/2070942.2070966

[16] R. Holloway and R. Beyah. 2011. Covert DCF: A DCF-Based Covert Timing Channel in 802.11 Networks. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*. 570–579. https://doi.org/10.1109/MASS.2011.60

[17] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. 2017. Bluebee: a 10,000 x faster cross-technology communication via phy emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems.*

[18] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R. Smith, and David Wetherall. 2014. Wi-fi Backscatter: Internet Connectivity for RF-powered Devices. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, New York, NY, USA, 607–618. https://doi.org/10.1145/2619239.2626319

[19] Philip Lundrigan. 2019. ONPC Receiver. https://github.com/NET-BYU/onpc_receiver.

[20] Philip Lundrigan. 2019. ONPC Transmitter. https://github.com/NET-BYU/onpc_transmitter.

[21] M. Zennaro, C. Fonda, E. Pietrosemoli, A. M. S. Okay, R. Flickenger, and S. Radicella. 2008. On a long wireless link for rural telemedicine in malawi. In *Proceedings of the 6th International Conference on Open Access.*

[22] Peizhong Yi, A. Iwayemi, and Chi Zhou. 2010. Frequency agility in a ZigBee network for smart grid application. In *2010 Innovative Smart Grid Technologies (ISGT)*. 1–6. https://doi.org/10.1109/ISGT.2010.5434747

[23] Ramya Raghavendra, Jitendra Padhye, Ratul Mahajan, and Elizabeth Belding. 2009. Wi-Fi networks are underutilized. *Microsoft Research Technical Report* (2009).

[24] Raspberry Pi Foundation. 2018. Raspberry Pi. https://www.raspberrypi.org.

[25] Matthias Schulz, Jakob Link, Francesco Gringoli, and Matthias Hollick. 2018. Shadow Wi-Fi: Teaching Smartphones to Transmit Raw Signals and to Extract Channel State Information to Implement Practical Covert Channels over Wi-Fi. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*. 256–268.

[26] N. Sornin and et al. 2015. LoRa Specification 1.0. http://www.lora-alliance.org.

[27] Mansi Subhedar and Gajanan Birajdar. 2011. Spectrum sensing techniques in cognitive radio networks: A survey. *International Journal of Next-Generation Networks* 3, 2 (2011), 37–51.

[28] S. Unni, D. Raj, K. Sasidhar, and S. Rao. 2015. Performance measurement and analysis of long range Wi-Fi network for over-the-sea communication. In *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. 36–41. https://doi.org/10.1109/WIOPT.2015.7151030

[29] Wemos. 2018. WEMOS D1 mini Lite V1.0.0. https://www.aliexpress.com/store/1331105.

[30] K. K. Wong and T. O'Farrell. 2003. Spread spectrum techniques for indoor wireless IR communications. *IEEE Wireless Communications* 10, 2 (April 2003), 54–63. https://doi.org/10.1109/MWC.2003.1196403