Developing Low-Cost Testbeds for Enhancing Security Techniques in Wireless Sensor Network Protocols

George D. O'Mahony
Dept. of Electrical and
Electronic Engineering,
University College Cork
Cork, Ireland
george.omahony@umail.ucc.ie

Philip J. Harris
United Technologies Research
Center Ireland
(UTRC-I)
Cork, Ireland
harrispj@utrc.utc.com

Colin C. Murphy
Dept. of Electrical and
Electronic Engineering,
University College Cork
Cork, Ireland
colinmurphy@ucc.ie

***Abstract*—Wireless sensor network (WSN) applications have expanded considerably over the past decade or so and now, solutions exit for various innovative applications. These wireless networks adopt commercial off the shelf devices and standardized protocols, which inherently creates security challenges. These challenges are ever changing as malicious interference and intrusion techniques evolve and dynamic efficient hardware becomes increasingly accessible. This paper presents the development of multiple low-cost hardware and software platforms designed so security enhancements and modifications to WSN protocol architecture and packet structure can be designed and tested. Each testbed has been built satisfying the requirements of being available as unmodified commercial off the shelf (COTS) components and based on open source software. The testbeds provide versatility through operating on various operating systems including Windows and Linux, are reproducible and can be deployed in a way which replicates real world WSNs. Each distinct system provides remote access, real time and off line data analysis, specific control of each network node and the ability to upload data from the WSN. This paper describes in-detail the individual pieces of suitable hardware for WSN protocol and packet structure design and illustrates the system architecture required to form testbeds which can experimentally validate modifications to a WSN protocol. Additionally, a baseline is defined and encapsulates the ZigBee standard. Example results of the distinct testbeds in operation are provided along with the specific open source software being used.**

***Index Terms*—COTS, Hardware, IoT, Low-cost, Open Source, Packet, PHY, Protocol, Security, Spectrum, Testbed, WSN & ZigBee.**

I. Introduction

As WSN applications extend their usability and become integrated in safety critical applications [1], new challenges in terms of security emerge due to stricter operational and availability requirements. This trend is a direct result of over a decade of research and development which has lead to feasible solutions to various innovative applications [2], including space-based WSNs [1], the Internet of Things (IoT), wireless networked control systems (WNCS) [3], aerospace applications [4] and using LEO satellites as WSN components [5]. WSNs typically consist of lightweight devices used to sense the physical world, have unique security issues due to their design and have deployments which inherently require security levels higher than typical wireless networks. Usual WSN devices hinder the use of complex or computationally intensive security protocols and propriety protocols in use can typically be reverse engineered by available tools. Consequently, network compromise, whether malicious or unintentional, can have significant consequences for privacy and/or safety. Therefore, the security and availability of the communication link and the delivery of authentic and confidential packets are essential for safety critical WSNs.

Furthermore, WSN protocol security is becoming increasingly important due to the ever expanding number of IoT devices and WSN applications, leading to increased levels of congestion in the radio spectrum, especially in the 2.4 GHz industrial scientific and medical (ISM) band. WSN protocols in use have very similar PHY and MAC approaches, which are typically based on a certain standard, for example, IEEE 80.2.15.4. Thus, the packet structure seems to follow a conventional approach, which, potentially, allows certain aspects to be exploited. In turn, this may result in some interesting innovations in the research space. This issues complexity deepens by virtue of each protocol coexisting with a variety of other protocols operating at the same frequency, for example, IEEE 802.11 and Bluetooth. Clearly, the spectrum must be shared in a efficient and non disrupting manor where WSN protocols must be secure from both unintentional interference and intentional security threats.

This paper identifies low cost methods of experimenting with security techniques in WSN protocols and packet structures for safety critical applications, where security is not optional but essential. A low-cost versatile COTS hardware and open source software approach is described and designed to achieve an understanding of existing WSN security techniques, with a focus on the packet structure. This approach expands to show how modifications, aimed to improve resilience to malicious attacks and/or coexistence, to packet structure and/or

This work was supported in part by the Irish Research Council and United Technologies Research Center Ireland under the Enterprise Partnership Scheme Postgraduate scholarship EPSPG/2016/66.

TABLE I
IEEE 802.15.4 (ZigBee) PHY Parameters

Parameter:	2.4 GHz PHY Value:
Number of Channels	16
Channel Spacing / Width	$5MHz$ / $2MHz$
Data / Symbol Rate	$250kbps$ / $62.5ksymbolsps$
Chip Rate	2 $Mchipsps$
Modulation / Spreading	O-QPSK / DSSS
Pulse Shaping	Half Sine / Normal Raised Cosine

TABLE II
XBee device Specifications

Parameter:	Value
Data Rate	$250kbps$
Indoor Range	$60m$
Transmit Power	$-5dBm \rightarrow 5dBm$
Receiver Sensitivity	$-100 \rightarrow -102\ dBm$
Frequency Band / No. Channel	$2.4GHz$ / 16
Interference Immunity	DSSS
Encryption (Optional)	128-bit AES
Reliable Packet Delivery	Retries/Acknowledgments

protocol security techniques, can be experimentally developed, fine tuned and compared to current standards, like ZigBee.

This paper is organized as follows: Section II describes both the hardware used to construct the testbeds and the baseline signal model, ZigBee. Section III describes a subset of identified security threats in WSNs and ZigBee's main security feature. Section IV specifies the testbeds and their potential ability to test enhancements to security features in WSNs protocols. Section V provides validation results, section VI outlines future work, while section VII concludes this paper.

II. Hardware

Before any hardware is defined a signal model is chosen, so a baseline for WSN protocol security can be established. Here, IEEE 802.15.4 2.4 GHz radio frequency band based ZigBee is adopted and it is currently the de facto standard in WSN communication, as almost all available commercial and research sensor nodes are equipped with ZigBee transceiver chips [6]. ZigBee's relevant physical layer parameters are shown in Table I. The 16 relevant channel center frequencies are provided in (1), where F_c is the center frequency and i is the channel number. Each of these channels operate in the unlicensed ISM frequency band and have to coexist with various protocols including, but not limited to, WiFi and Bluetooth. ZigBee is constructed using the PHY and MAC from IEEE 802.15.4, where the PHY uses direct sequence spread spectrum (DSSS) and offset quadrature phase shift keying (O-QPSK) and the MAC uses carrier sense multiple access with collision avoidance (CSMA-CA). It works using PANs (Personal Area Networks) and the topology can be either mesh, star or peer-to-peer. This signal model is used both as a baseline for the testbeds in section IV, and as a comparison to any potential methodologies for enhancing the security of WSN protocols. Due to being based on IEEE 802.15.4, it is likely that any developed methodologies can transverse various protocols, as IEEE 802.15.4 is commonly adopted.

$$F_c = 2405 + 5(i - 11)\ MHz,\ for\ i = 11, 12, ...26 \tag{1}$$

- **XBee Devices**: These devices are the designated ZigBee nodes and operate ZigBee Pro 2007, but can also specifically run either the IEEE 802.15.4 or DIGI's DigiMesh 2.4 protocols. These COTS devices are low cost (3 pack kit ≈€90) and have specifications as per Table II. Each device, along with the network, is configured using DIGI's XCTU software and includes selecting the channel, PAN ID, Tx power and node use. The specific device types are [7]:

1) Coordinator: A full function device (FFD) which is responsible for controlling the entire network, relaying messages and authenticating devices.
2) Router: A FFD responsible for forwarding and relaying data packets. This device type can communicate with the coordinator and end devices.
3) End Device: A reduced function device which communicates with the coordinator or a router only and cannot communicate with other end devices or relay packets.

The devices can utilize cluster topologies, which typically improves stability, reduces energy consumption and compresses the amount of transmitted data. Cluster head networks have many uses, for example, relay nodes (RN) which aggregate data and forward to Nanosatellites [5]. Here, the XBee devices are controlled and programmed through a development board which requires a USB connection and are programmed (after initial configuration) using the digi-xbee python3 library. Remote control and power of each XBee can be obtained by using a Raspberry Pi or equivalent, providing realistic deployment scenarios and real time data analysis.

- **Raspberry Pi**: This is a small credit card sized affordable (≈€60) computer which can be utilized in a number of ways to provide remote control and low-cost operation of USB powered devices. Once initial setup is performed, a remote connection can be achieved through a secure shell (SSH) on Windows, using a putty terminal and the devices IP address, or on Linux by using the command "$ssh\ pi@IP-Address$". Different Raspberry Pi models are employed, including the Pi 2 Model B, Pi 3 Model B and Pi 3 Model B+ (shown in Fig. 1), and the main specifications are provided in Table III. These small computers run a Debian based operating system called Raspbian and can run full Linux applications, like GNU Radio, python scripts, terminal commands etc. Hence, each Raspberry Pi can use the digi-xbee python3 library to control a XBee device, store and analyze data/results and upload any necessary information. Built in WLAN or a WLAN USB dongle provides remote access and the ability to update code and hardware without the need to be physically at the test-bed. These devices support real world deployment scenarios as, typically, WSNs are left unattended and deployed where remote access or

TABLE III
RASPBERRY PI SPECIFICATIONS

Pi 2 Model B	A 900MHz quad-core ARM Cortex-A7 CPU 1GB RAM 4 USB ports
Pi 3 Model B	Quad Core 1.2GHz Broadcom BCM2837 64 bit CPU 1 GB RAM 4 USB Ports BCM43438 wireless LAN Bluetooth Low Energy (BLE)
Pi 3 Model B +	1.4GHz Broadcom BCM2837B0, Cortex-A53 64-bit SoC 1 GB RAM 4 USB 2.0 Ports 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac WLAN Bluetooth 4.2, BLE

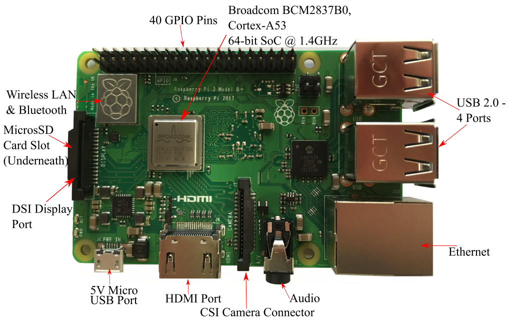


Fig. 1. Raspberry Pi Model 3 B+ with main elements identified

monitoring is the norm.

- **LimeSDR Mini**: This software defined radio (SDR) is described by the manufacturers [8] as “the perfect way to start experimenting with and building your own wireless networks, protocols, and testers”. This statement fits perfectly into the development of WSN based experimental testbeds and specifications for the LimeSDR Mini are provided in Table IV. These specifications highlight the wide range of functionality that the LimeSDR Mini encompasses, which can be used to test and develop security measures for enhancements to both protocol and frame design. This SDR is used with GNU Radio and Pothos Flow for implementation, through signal processing blocks, by exploiting the Lime-Suite tool and has full visualization as a spectral analyzer through SDRConsole. Beneficially, it can run on a Raspberry Pi by using GNU Radio and LimeSuite. Importantly, it can be used in multiple testbeds to experiment with existing and modified design approaches along with other WSN aspects, like mitigation strategies.
- **Analog Pluto SDR**: The Pluto is an Analog Devices SDR (≈€100) and has specifications as per Table IV. This SDR is controlled and analyzed by either, using Simulink, through the communications systems toolbox add on, or GNU Radio. This provides certain advantages and tests a variety of attack styles quickly, while exploiting Matlab for efficient analysis. The Simulink add on includes a transmitter and receiver Simulink block for quick set up and the communications toolbox provides various blocks for different modulation methods, filter, mixers etc., which can be applied to create various PHY layer outputs and create signals, which match specific protocols or attack styles, as seen in Fig 9.

TABLE IV
SDR SPECIFICATIONS

	LimeSDR Mini	Analog Pluto
Connectivity	USB (FTDI FT601)	USB
Frequency Range:	$10MHz - 3.5GHz$	$325MHz - 3.8GHz$
RF Bandwidth:	$30.72MHz$	$20MHz$
Sample Rate:	$30.72Msps$	$65.2ksps - 61.44Msps$
Sample Depth:	12 bits	12 bits
Transmit Power:	Max 10 dBm	7 dBm
TX/RX channels	1 TX, 1 RX	1 TX, 1 RX

TABLE V
TEKTRONIX RTSA 306B SPECIFICATIONS

Frequency Range	$9KHz\ to\ 6.2GHz$
Acquisition Bandwidth	$40\ MHz$
Typical Accuracy	$+/-\ 20ppm$
ADC Sample Rate and bit width	$112\ Msps,\ 14bits$

- **TI CC2531EMK**: This USB dongle employs TI’s Packet Sniffer software [9] to both capture and decode ZigBee packets and has been successfully tested on XBee transmitted packets. The decoded information includes PAN ID, source and destination address, packet length, packet type (data or acknowledgment) and the payload, which is decoded if no encryption is used or if the key is known. A Raspberry Pi, or a Linux system, operates the dongle in a limited capacity but can still provide address information and payload.
- **Tektronix Real Time Spectrum Analyzer (RTSA) 306B**: This is a USB powered RTSA and can analyze network operation, identify nodes, estimate node type, visualize signal structure and analyze how signals co-exist with each-other. Through Tektronix Digital Phosphor technology (DPX), the RTSA can visualize the RF spectrum and provide pixel information. Consequently, attack, protocol and coexisting signals are identifiable during an experiment. The RTSA’s main specifications are provided in Table V and, conveniently, it has Matlab support through the Instrument Control Toolbox.

III. SECURITY

WSNs require security, especially when the networks are used in hostile environments or in critical applications like military, aerospace, commercial or the IoT. Protocols in use are susceptible to various attacks which need to be understood and the effects outlined. Certain techniques have been developed to allow WSN protocols to be resilient to these attacks, however, it is likely more enhancements are required. The hardware described in Section II can be used to demonstrate the effects of certain malicious attacks on a commercially and industry adopted protocol, presenting security vulnerabilities in the process. A selection of these attacks are categorized as follows:

- Conventional Jamming Attacks, typically, aim to overpower legitimate signals with spurious radio-frequency transmissions. While higher jamming power increases effectiveness, it is also easier to detect, as such, adversaries typically optimize the interference signal to maximize packet loss while minimizing total broadcast power. Examples include, constant, deceptive, random and reactive jammers.

- MAC Layer Attacks react to the protocol being used in the network by eavesdropping on and/or sniffing (CC2531) network packets. Intelligence and learning based jammers use the analyzed results to attack based on specific network operation and include replay attacks, spoofed packets, matched protocol interference [10] and forcing a device to remain in listening mode.
- Network Layer Attacks, generally, cause either a denial of service (DoS), a privacy or an impersonation attack and include selective forwarding, sinkhole, blackhole, Sybil and HELLO flood attacks.
- System Coexistence: In certain situations spectrum coexistence and sharing is seen as an attack. Examples include when a secondary user (SU) occupies a primary user's (PU) spectrum, causing interference when the intention was to maximize spectrum use and the deliberate denial of spectrum sharing, when a specific user consumes all resources.

The main security protection in ZigBee is the 128-bit advanced encryption standard (AES). AES is dependent on the predistribution, initialization, use and storage of the 128 bit key, which has 10 transformation rounds that convert the plaintext into the final output, called the ciphertext. In ZigBee, the packet payload is encrypted while the synchronization and physical layer headers are not. The encryption and data integrity is provided through the use of the counter mode cipher block chaining message authentication code protocol.

IV. Testbed Architecture

The developed testbeds satisfy certain requirements to allow each testbed to be reconfigurable, useful in multiple situations and mimic real world environments and operating conditions. The testbeds are low cost (each SDR less than €150 and other devices less than €60), built using COTS components and leverage open source software. Therefore, the testbeds will resemble real world WSNs which are, typically, constructed using COTS components and are deployable in variable conditions and situations. The RTSA is an analysis tool and is exclusively used for spectral analysis and is not required for testbed operation. The testbeds are as follows:

1) **Baseline**

Initially, a baseline needs to be established and used in order to fully understand the security levels which are evident in a known protocol already in use in industry and commercial applications. The baseline test-bed contains the Digi XBee devices which are powered and controlled remotely using various Raspberry Pi models. Using a Raspberry Pi with XBee devices was previously described using the GPIO serial port in [11] and in [12], [13], [14], [2] and [15], where one Raspberry Pi device is normally used as a base station attached to the network coordinator and provides client services, which allows applications, like environmental monitoring, to be established.

Here, the approach is different as the grove development board is exploited to allow complete control and access to every node, which has been setup using XCTU software. This concept authorizes full control of each node, packet analysis on each receiver and transmitter and data analytics at each node location. This is integral, as the vision for this WSN test-bed is to experimentally test and highlight network, protocol and frame vulnerabilities, initially using ZigBee. The node connections can be visualized in Fig. 2 and it mimics typical WSN operation as the nodes are static, use the cluster head network model and are sufficiently dispersed. This ZigBee based test-bed includes one coordinator (cluster head) and multiple receivers (cluster members) and each individual XBee device is connected to a Raspberry Pi, which uses the digi-xbee python3 library to control the XBee nodes and allow remote access through built in wireless LAN or USB dongles. Basic example code which uses a XBee device to transmit a message every 2 minutes and the corresponding receiver code is provided in Fig. 3. Under baseline operation, it is envisaged that each cluster member will send a data packet every x minutes, while all packets are monitored by each Raspberry Pi device. The remote access aspect is typical of WSN deployments, as they are usually deployed and left unattended and the data is uploaded through a gateway that can be accessed from clients monitoring the network.

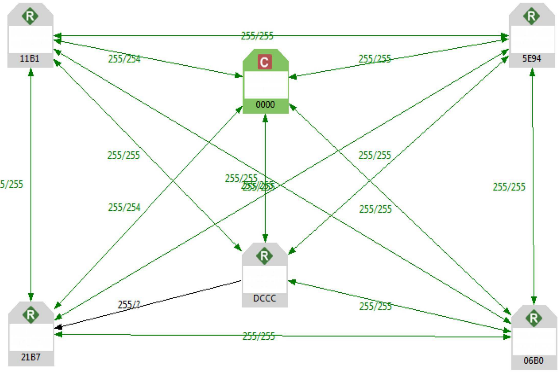


Fig. 2. XBee configuration, visualized using DIGI's XCTU software

This setup was previously validated, by the authors, in [10], where it was tested for a combined 111 hours of operation. The validated testbed allows basic operation of a WSN to be monitored and for various WSN attacks, applied using a SDR, to be experimentally validated for their effectiveness on a working ZigBee WSN and for the identification of specific attack events. Results can be compared with theoretical analysis and modeling of various protocols and frame structures currently in use, including ZigBee, WirelessHart, ISA100.11a, MiWi, 6LoWPAN and Thread. By using this testbed, identified WSN vulnerabilities can be logged and transferable security enhancing methodologies designed.

2) **Protocol Development**:

This testbed has been developed so enhancements to the security of existing WSN protocols and methodologies can be designed. The main components are SDRs, which, here, are the LimeSDR Mini devices. These SDRs can be controlled

```
from digi.xbee.devices import ZigBeeDevice, RemoteZigBeeDevice
from digi.xbee.reader import XBee64BitAddress
import time
# Transmitter running on XBee Router
XBee_Router = ZigBeeDevice("/dev/ttyUSB0",9600)
XBee_Router.open()
Remote_XBee_Coord = RemoteZigBeeDevice(XBee_Router,\
XBee64BitAddress.from_hex_string(Coord_64_Bit_Address))
try:
while True:
Data = "Test Packet"
XBee_Router.send_data(Remote_XBee_Coord, Data)
time.sleep(120)
except:
XBee_Router.close()
# Receiver running on XBee Coordinator
XBee_Coord = ZigBeeDevice("/dev/ttyUSB0",9600)
XBee_Coord.open()
try:
while True:
Received_Message = XBee_Coord.read_data(1000)
Remote_XBee_Receiver = Received_Message.remote_device
Received_Data = Received_Message.data
except:
XBee_Coord.close()
```

Fig. 3. Python3 code controlling a Raspberry Pi connected XBee device

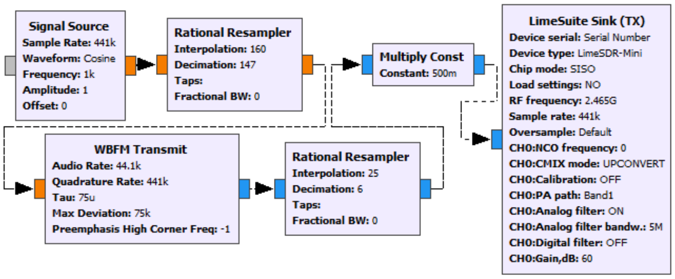
using GNU Radio running on a desktop PC or on a Raspberry Pi device. The LimeSDR Mini devices have the ability to run multiple protocols and output different signal structures, which means the concept is not restricted to only one protocol but can, instead, start with ZigBee and expand as time progresses. Initially, two LimeSDR Minis are used to design and initialize a full duplex communication session between two nodes, which must be comparable to the baseline setup and the packet sniffer can be used to see if the packets are credible. Fundamentally, this approach tries to design and experimentally test, tune and compare modified protocols, where the alterations focus on improving the security of WSN protocols and frame structures. By adding a third LimeSDR Mini, the node-to-node communications can be tested under no attack and attack conditions. Firstly, this allows for protocol and frame vulnerabilities (identified through baseline experimentation and theoretical analysis) to be identified and compared to the baseline. Any modifications can then be put through a series of tests to see if any significant improvement on the baseline is achieved. By having an embedded Linux option using the Raspberry Pi, variable deployments and channel effects are achievable and facilitate a more realistic setup.

3) **Comparison**:

By exploiting the Raspberry Pi devices, remote access, real time and off-line data analysis and specific control of network nodes, a comparison testbed is achievable. This approach combines the baseline and protocol development approaches to facilitate a real time comparison of an existing protocol and a modified version. The XBee nodes and SDRs operate under similar channel and environmental conditions, while mimicking real world scenarios by running simultaneously on a Raspberry Pi platform. Thus, by utilizing communication between three Raspberry Pis, three XBee nodes and three LimeSDR Minis, fully comparable results are attainable.

V. Example Results

To highlight the versatility of the low-cost open source testbeds described in section IV, operation was confirmed on Windows 7, Ubuntu 18.04 LTS and Raspbian stretch version 9 running on a Raspberry Pi Model 3 B+. Each OS was tested using specific GNU Radio flow graphs and variations of the python code in Fig. 3. A simple cosine signal was used as the transmitting signal and the flow graph is provided in Fig. 4, while it is visualized by the RTSA in Fig. 5. A simple FM radio receiver using a known local signal, 106.1 MHz here, was used to test reception and the flow graph is provided in Fig. 6. The signal structure of a ZigBee packet transmitted using a XBee node is provided in Fig. 7 and it confirmed, along with monitored packets on associated Raspberry Pi devices, a successful transmission. Additionally, the coexistence issue explained in section III is visualized using the RTSA in Fig. 8, where a ZigBee signal coexists with Bluetooth and WiFi signals. Finally, the versatility of using a SDR was highlighted by transmitting a matched ZigBee signal [10] (Fig. 9) using the Pluto SDR and Simulink software.



Signal Source
Sample Rate: 441k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Rational Resampler
Interpolation: 160
Decimation: 147
Taps:
Fractional BW: 0

Multiply Const
Constant: 500m

LimeSuite Sink (TX)
Device serial: Serial Number
Device type: LimeSDR-Mini
Chip mode: SISO
Load settings: NO
RF frequency: 2.465G
Sample rate: 441k
Oversample: Default
CH0:NCO frequency: 0
CH0:CMIX mode: UPCONVERT
CH0:Calibration: OFF
CH0:PA path: Band1
CH0:Analog filter: ON
CH0:Analog filter bandw.: 5M
CH0:Digital filter: OFF
CH0:Gain,dB: 60

WBFM Transmit
Audio Rate: 44.1k
Quadrature Rate: 441k
Tau: 75u
Max Deviation: 75k
Preemphasis High Corner Freq: -1

Rational Resampler
Interpolation: 25
Decimation: 6
Taps:
Fractional BW: 0

Fig. 4. Example flow graph for transmitting a signal using LimeSDR Mini

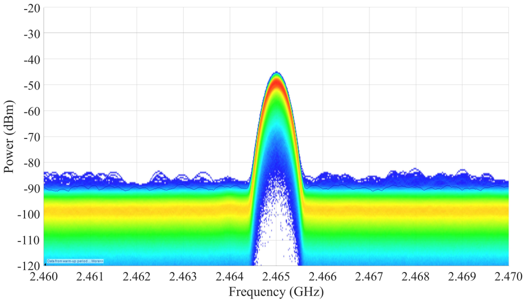
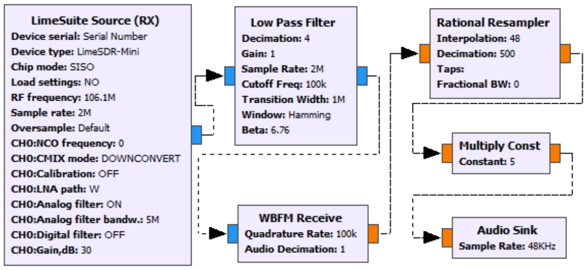


Fig. 5. DPX image proving successful LimeSDR Mini transmission



LimeSuite Source (RX)
Device serial: Serial Number
Device type: LimeSDR-Mini
Chip mode: SISO
Load settings: NO
RF frequency: 106.1M
Sample rate: 2M
Oversample: Default
CH0:NCO frequency: 0
CH0:CMIX mode: DOWNCONVERT
CH0:Calibration: OFF
CH0:LNA path: W
CH0:Analog filter: ON
CH0:Analog filter bandw.: 5M
CH0:Digital filter: OFF
CH0:Gain,dB: 30

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 2M
Cutoff Freq: 100k
Transition Width: 1M
Window: Hamming
Beta: 6.76

Rational Resampler
Interpolation: 48
Decimation: 500
Taps:
Fractional BW: 0

Multiply Const
Constant: 5

WBFM Receive
Quadrature Rate: 100k
Audio Decimation: 1

Audio Sink
Sample Rate: 48KHz

Fig. 6. Example flow graph for FM radio reception

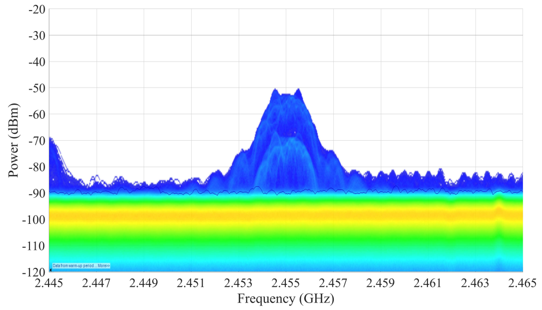


Fig. 7. DPX image of a ZigBee signal at 2.455 GHz

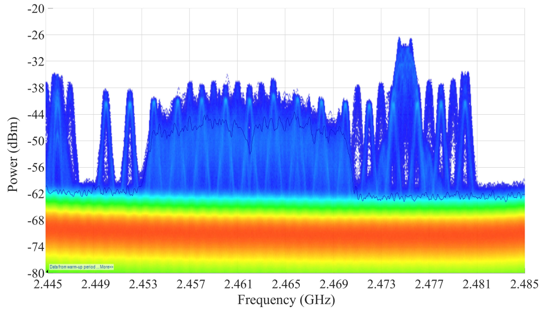


Fig. 8. ZigBee signal coexisting with WiFi and Bluetooth

VI. Future Work

Future progressions of the testbeds described in this paper include the analysis of existing protocol structures and security techniques. This analysis, executed with an emphasis on why each security aspect is used, will provide insights into how to improve WSN protocols in terms of security and design methodologies. Additionally, the specific packet structures in use will be analyzed and the reasons behind its design identified, so as to find methodologies which may help to improve security. This analysis work can be tested using the baseline setup to get results from a working network. Any design improvements will be experimentally tested using the protocol development testbed and a real time comparison will examine whether changes make a significant improvement. The baseline and protocol development testbeds must be compared before any design alterations are made, so as to clarify a working baseline in the SDR approach.

VII. Conclusion

This paper described a low-cost approach to design and test modifications made to WSN protocols and make a comparison to the industrially and commercially adopted protocol, ZigBee. The approach involved distinct testbeds based on ZigBee and SDRs, which were comparable to real world applications, due to the use of Raspberry Pi devices for remote access and dispersed deployment. Analysis was provided through a spectrum analyzer, packet sniffer and embedded Linux on Raspberry Pi. The testbeds were low-cost versatile COTS open source approaches to WSN research and were tested on multiple operating systems, including Windows and Linux. Uses include identifying WSN security vulnerabilities, analyzing both WSN security techniques and attacks on WSNs and for developing transferable methodologies for secure packet structure and protocol design.

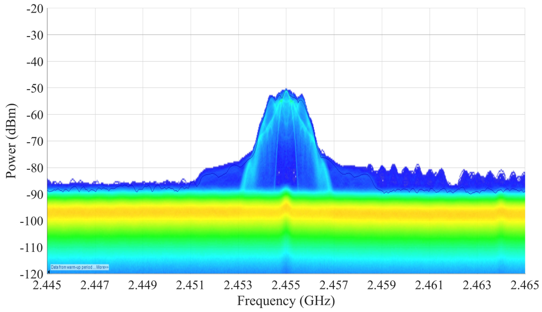


Fig. 9. A matched ZigBee signal produced by a SDR

References

[1] T. Vladimirova, C. P. Bridges, J. R. Paul, S. A. Malik, and M. N. Sweeting, "Space-based wireless sensor networks: Design issues," *IEEE Aerospace Conference*, pp. 1–14, 2010.

[2] S. G. Nikhade, "Wireless sensor network system using Raspberry Pi and zigbee for environmental monitoring applications," *International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials*, pp. 376–381, 2015.

[3] P. Park, S. C. Ergen, C. Fischione, C. Lu, and K. H. Johansson, "Wireless Network Design for Control Systems: A Survey," *IEEE Communiations Surveys and Tutorials*, vol. 20, no. 2, pp. 978–1013, 2018.

[4] R. K. Yedavalli and R. K. Belapurkar, "Application of wireless sensor networks to aircraft control and health management systems," *Journal of Control Theory and Applications*, vol. 9, no. 1, pp. 28–33, 2011.

[5] A. Addaim, A. Kherras, and Z. Guennoun, "Design of WSN with Relay Nodes Connected Directly with a LEO Nanosatellite," *International Journal of Computer and Communication Engineering*, vol. 3, no. 5, pp. 310–316, 2014.

[6] B. Stelte and G. D. Rodosek, "Thwarting attacks on ZigBee - Removal of the KillerBee stinger," in *Proceedings of the 9th International Conference on Network and Service Management*, 2013, pp. 219–226.

[7] Q. Yang and L. Huang, *Inside Radio: An Attack and Defense Guide*, 2018.

[8] Lime Microsystems, "LimeSDR Mini." [Online]. Available: https://www.crowdsupply.com/lime-micro/limesdr-mini

[9] Texas Instruments, "SmartRF Protocol Packet Sniffer." [Online]. Available: http://www.ti.com/tool/PACKET-SNIFFER

[10] G. D. O Mahony, P. J. Harris, and C. C. Murphy, "Analyzing the Vulnerability of Wireless Sensor Networks to a Malicious Matched Protocol Attack," in *52nd IEEE International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2018, pp. 1–5.

[11] Chenxi Ouyang, "Design and Implementation of a Wireless Zigbee Mesh Network," Ph.D. dissertation, 2014.

[12] S. G. Nikhade and A. A. Agashe, "Wireless sensor network communication terminal based on embedded Linux and Xbee," in *International Conference on Circuits, Power and Computing Technologies*. IEEE, 2014, pp. 1468–1473.

[13] M. Tao, X. Hong, C. Qu, J. Zhang, and W. Wei, "Fast access for ZigBee-enabled IoT Devices Using Raspberry Pi," *Proceedings of the 30th Chinese Control and Decision Conference*, pp. 4281–4285, 2018.

[14] S. R. Akbar, W. Kurniawan, M. H. H. Ichsan, I. Arwani, and M. T. Handono, "Pervasive device and service discovery protocol in XBee sensor network," in *International Conference on Advanced Computer Science and Information Systems*. IEEE, 2017, pp. 79–84.

[15] S. Ferdoush and X. Li, "Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications," *Procedia Computer Science*, vol. 34, pp. 103–110, 2014.