BuzzPass: A Digital Brush-Passing Solution

Can the Zigbee Communication protocol be used safely for covert file transfer of files?

Alexandre Robic
Department of Computer Science and Engineering
Texas A&M University
College Station, Texas
alexrbc77@tamu.edu
UIN: 432001367
Stuart Nelson
Department of Computer Science and Engineering
Texas A&M University
College Station, Texas
s.s.nelson@tamu.edu
UIN: 932002485

***Abstract*—Our first objective is to create a product that is able to transfer large data packets in the form of large files (several Mbs) from one device to an identical one in a short range (approximately 50 meters) and a short amount of time (15 seconds max). To do this, we would use a Raspberry Pi Zero W for the file processing and an XBee board for the burst transmitting or receiving of information.**
To make sure this whole system works, we would test out in an urban environment, here simulated on Texas A&M's campus to evaluate the optimal distance at which both devices can communicate with each other as well as the speed at which this can be done and potential packet loss.
Given the intended use for our product (to create a digital brush passing system), we would host a surveillance exercise here on campus using some students from the INTA 700 class (the art of counter intelligence). Students participating in this exercise would not know who the transmitter is or that the devices are being used. The goal here would be to do a brush pass under a surveillance team's nose. Once this exercise is done, we would present BuzzPass to the class and introduce the topic of technical collection systems and covert communication methods.
To ensure BuzzPass' security, we would focus the second part of our project on trying to detect the system and prevent its use. We can imagine the system somehow falls into the wrong hands and is being used against American interests. To do this, we can attempt to isolate the ZigBee signal, create a detector to hone in on the transmitter and jam the receiving device.

***Index Terms*—IEEE 802.15.4 Protocol, ZigBee, XBee, RapsberryPi, Brush Pass, IoT**

I. Introduction

Intelligence gathering and secret communications between officers and assets has considerably evolved throughout the years, especially during the Cold War when spying reached its peak in history. To communicate securely and safely, parties could generally rely on the use of One Time Pads (OTP) that would become invalid after it was used for decryption.

Some less evolved methods included the use of a Burst Radio System, nicknamed Buster by the CIA. It was used to handle very high priority targets like Major General Dmitri Polyakov, who would later one be burned by both Aldrich Ames and Robert Hannsen, two of the biggest names in American Counterintelligence history[9].

In more recent years, with the democratization of the smartphone, information sharing has moved from paper messages and packages to sending over data on the internet or using electronic storage devices like flash drives. Although such devices present a great opportunity for exchanging large quantities of data or software developed by the targeted service, they present the risk of containing malware that would alert the service of a possible intrusion. The rapid rise of street cameras and the software behind them to automate detection also poses a new threat to intelligence officers and their ability to conduct discreet meetings in public or to be able to conduct brush passes. Over surveillance of areas and the large amount of manpower certain adversarial services possess make it impossible to hand off items or messages in public without being spotted. Such acts being caught on camera are what led to the arrest of Anna Chapman and Operation Ghost Stories.

It is the combination of all of these problems that has led us to the original idea of BuzzPass, a device that could be undetected by a surveillance team, when in use, that allows data to be transferred between 2 parties in close proximity. We based our design and initial development on the idea of the PirateBox, a fixed device with its own WiFi Access Point that people would connect to in order to transfer files. Its main disadvantage is its easy detectability. IT technicians often carry around devices that allow them to find the origin of a WiFi signal to get to the device they're trying to debug.

Our device therefore has to be easy to use (have a limited amount of controls), safe (incorporate very secure cryptography schemes) and be undetectable visually (both devices should start communicating when in range and after they have authenticated each other).

II. Product development

A. Zigbee

Zigbee is a communication protocol widely used in IoT for low-cost, low-power mesh networks. It operates using the IEEE 802.15.4 radio specification and on non-regulated bands like the 2.4GHz band, used widely for RC communication.

A zigbee network is composed of 3 types of devices: Endpoints, Routers and a Coordinator. Each network requires at least a coordinator. The routers can either communicate between them, with an endpoint or with the coordinator, passing data along to other devices. The coordinator is a device ensuring communication between all others in the network, is responsible for connections and broadcasting the channel ID as well as being a repository for the security keys (in the case where the key is not set but generated randomly). The endpoint can only send data to and receive it from a router or coordinator but cannot pass it along. This allows it to be asleep for most of its use, increasing battery life in certain applications. In this use case, none of our devices are being used as endpoints.

The channel ID is set by the user and every device programmed with that specific ID will try connecting to the corresponding channel. A 128 bit AES key can also be set for more secure communication and to prevent unwanted devices from connecting.

B. The XBee Pro boards

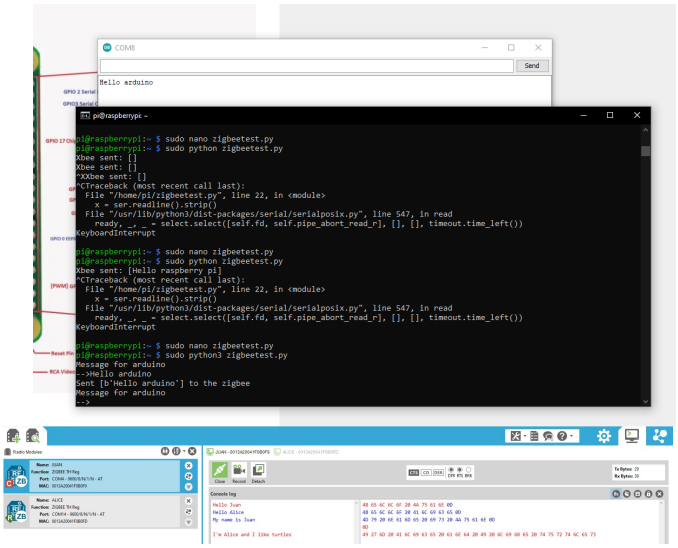
The XBee Pro RF module provides cheap wireless communication between 2 devices and

can use a wide variety of protocols including Zigbee and DigiMesh. The board we chose for this project has a range of 90m to 3200m for communication. It can be programmed using UART or SPI and can receive configuration updates through AT commands, API commands or over the air. Since we're looking at the zigbee communication protocol, our boards will be communicating over the 2.4GHz frequency band.

To configure our devices to communicate with each other, we used the recommended XCTU software provided by the manufacturer. Configuration of the boards relates directly to that of the Zigbee environment. Each device was given a nickname (Alice or Juan) for better configuration. Alice and Juan were chosen as names in reference to the classic Alice and Bob of cryptography but Juan comes from Pr Juan Garay who teaches the cryptography courses here at Texas A&M.

Alice was configured as a Zigbee router whereas Juan was configured as a coordinator since every Zigbee network needs at least one coordinator. Attempts were made to have both boards as routers but failed. Both devices were given a channel number to connect to (42069), each other's Serial numbers as well as a shared key for communication. This configuration ensured that both devices could send and receive information, tested here with a simple phrases.

It is important to note that we started out with a baud rate of 9600 but increased this to a more widely used 115200 further on in our development.



C. *Communicating using other devices as clients*

Once both XBee boards were communicating for small phrases, we moved towards development of the final product. As a median test, we connected each board to an Arduino Uno using the Serial pins and a Serial converter for each one. This allowed is to test communicating over the Zigbee network using a programmable device instead of the manufacturer provided software. It also gave us an idea of the overall architecture our final product would need to follow.

Once the Arduino tests proved conclusive, we moved towards development of the communication integration using the Raspberry Pi zero boards. To do this, the RPI's UART ports were utilized and the serial console interface was disabled. Once completely assembled, both devices were tested using a simple Python script that allowed users to send simple phrases over the Zigbee network (a chat application) called zigcomm.

After being able to transfer simple phrases, we looked at transferring files of a small but significant size. We found that we're able to send a 4KB file within 10 seconds without packet loss and a 2KB file in about half that time.

Images above: The packet sniffer used for sniffing (or attempted packet sniffing) and the BuzzPass box underneath.

D. Programming

The main library that was used was wiringpi, both in C and in python. We first started development using python since it was easier to understand and a lot of documentation for serial communication and other hardware programming involving the Raspberry Pi. The initial communication procedures we used to transfer files and early debugging of our system were achieved using wiringpi as well as the python serial library. We configured the raspberry pi to follow the same configuration as the XBee Pro, with a 115200 serial baud rate (initially at 9600 to work with the arduino but enhanced to 9600 for speed issues), no parity bits and a single end bit. A simple python program was first written in order to see if both devices could communicate together using the raspberry pi. Whenever we flashed the XBee Pros with new firmwares (Zigbee or Digimesh) as well as changed parameters (Baud rates, encryption keys, channels etc), our first test was to run the Zigcomm.py script to ensure communication was not lost.

E. Communication procedures

File transfers were initially done in python and allowed us to write our communication procedures but final experiments were done in C due to python's speed when communicating over serial with large amounts of information at a time.

Each transfer beings with FILE-NAME{wyoming.txt} and ends with ENDFILE. On the receiving console, we could therefore ensure that the file being sent over was indeed the right one. The receiving console also showed us how many bytes were received at the end of the transmission to compare with the file size on the transmitting machine. This gave us our percentage of packet loss.

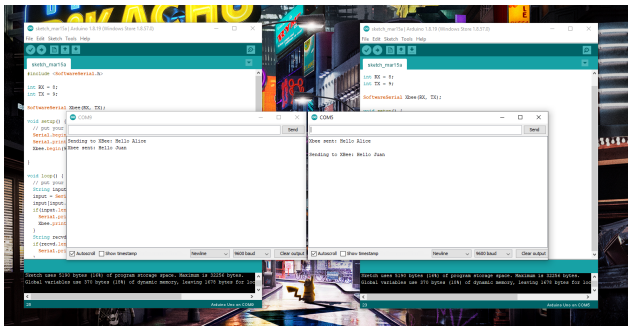
We added a screen on top of the Raspberry Pi Zero in order to allow a user to choose which file(s) are to be transferred and started work on a GUI but had to give up due to time constraints. In a version that can be provided to the Bush School, a GUI, even simple can be added in order to allow users to choose files for transfer without requiring a console.

The final code for file transfers remains relatively simple with less than 50 lines for each program. The sending program opens up serial communications and returns 3 if the link is correctly established. Due to time constraints we could not implement a lot of error cases. The file was sent in packets of various sizes and had to include a delay in ms between each packet in order to not overwhelm the receiving device. Several packet sizes and

delays were tested until we found the largest packet possible with the lowest delay that would allow us not to experience packet loss. This winning combination was achieved through experimenting and trial and error. We eventually settled on a packet size of 128 bytes and a delay of 500ms for Zigbee and repeated this experiment in Digimesh as well. Zigbee proved to be very slow for file transfer (without packet loss) but Digimesh allowed for a delay of around 60ms with packets twice the size of those that we used when implementing Zigbee.

The receiving program works in a similar way but instead of receiving packet by packet, each character is immediately written to the file.

A third program was also written but not implemented as part of experimentation. The ping program allows one box to ”ping” the other with a specific code (here the prime number 35317). If the other device does not acknowledge the code, the whole communication procedure shuts down. This ensures that in the case one device and the Zigbee key (AES 128 bit transmission key) is in the hands of a malicious party, without the code, the whole system becomes unusable. This however can pose new security issues with an adversary spamming wrong codes in order to disrupt operations. We did not run the pinging script when timing the file transfers.



F. Problems encountered during development

”Packet loss” became a frequent term in our communications. This term was mainly employed when talking about a file we were trying to transfer but arrived at the second device with missing pieces. An illustration of this would be if a phrase was typed out on one device reading ”I like trains and donuts” and arrived one the second device as ”I likuts”, here the loss would be ”e trains and don”. Since we first thought this was a software problem, we looked at several packet size and delay configurations. Eventually, we found out the main problem was in the hardware we were using, the FTDI chip on the serial converter was a knock off chip, meaning that it would overheat and cause loss of information (or as we called it, packet loss). Once a swap of the defective hardware was done, we were then able to settle on a packet size of 128 bytes being able to be transferred every 100ms. This configuration allowed us to be able to transfer a 4kb file within 10 seconds. Further experiments were done using the Digimesh communication protocol instead of Zigbee. We were able to get much better results on transferring our wyoming.txt test file. Hhowver, we did notice that after a few file transfers, we were still getting packet loss. It seems that going through an FTDI serial converter before the data gets to the raspberry pi might be our bottleneck. Our next design iteration can focus on skipping the serial converter or using a platform that has zigbee combined into it. For this, we can look at using a Lilygo RISC-V based development board which should hopefully give us better performances and lower costs since it would only be $8 as opposed to the $40 of the XBee Pro.

III. Security evaluation

Security means a lot to the people who would be using our product in an operational environment. This is the reason that led us to choosing the Zigbee communication protocol over WiFi or Bluetooth. Since it is not used very

much outside the IoT environment, it should not be too detectable to a surveillance team. To that can be added the integrated support for 128 bit AES that we utilized in order to keep communications secured and unreadable.

We maintained a consistent key every time in order to rule out a change of key or a key regeneration to be the cause of latency when transferring files. We did not observe a difference in delays when having encryption on as opposed to no encryption meaning security would not hinder speed in an operational setting.

One experiment we conducted as part of our security evaluation was to have one device have a correct key with another one having a slight variation (last character deleted). We expected to see encrypted traffic come through but instead both devices ceased to communicate with each other.

IV. Performance Analysis

A. Speed and Reliability

Six different files were transferred during the preliminary tests and most files under 300kb were able to send at almost exactly 30 seconds. Two files, an image and a pdf, suffered from severe delays and packet losses. The tests indicate that the sender was able to send the file in about 10 seconds, meaning that the rest of the time was either spent processing the data on the receiving end or while attempting to detect the recipient.

File Type	Size (kb)	Duration (s)	Retained (%)
ping.c	1.2	29.542	100%
wyoming.txt	3.561	31.544	100%
magikarp.ani	8.656	34.543	100%
hello.c	15.718	35.042	100%
image.jpeg	32.993	167.112	100%
AlexRobicCV.pdf	42.795	344.677	90.46%

TABLE I
Speed and Quality of each file type

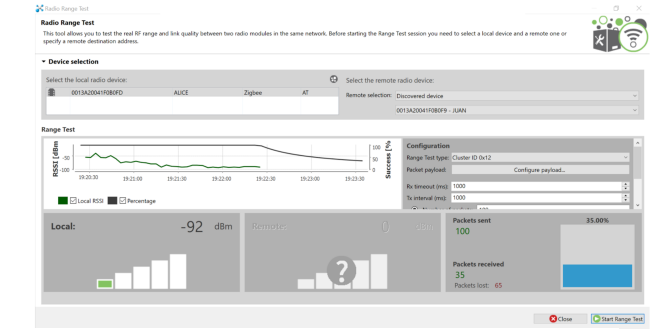
B. Field testing

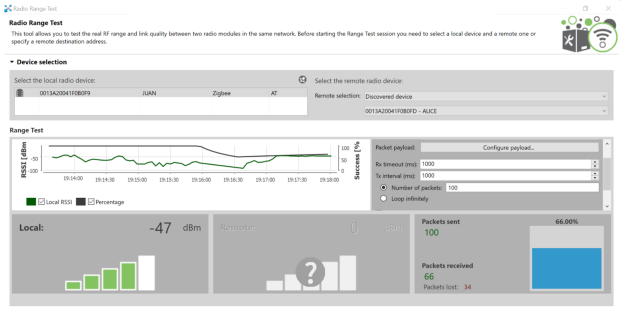
Three different test of the BuzzPass devices were conducted: a stationary test to test the difference between exposed hardware, a distance test to analyze the correlation between communication distance and packet loss, then a noise test to analyze the correlation between distance, digital signal noise, and packet loss.

For the first test, three sub-trials were conducted: one with both devices exposed, one with the recipient concealed in the black case, and one with both concealed. All three were performed while both devices were in close proximity with each other. All three transfers in the sub-trials concluded successfully with no packet loss and the code execution time seemed to have no correlation to whether the hardware was exposed or concealed. The code executed the fastest in the second test with the recipient concealed and the third test with both concealed only experienced a delay of 90ms compared to the first test.

For the distance test, the control test was conducted with both devices 0ft apart, the second was at 40ft apart, and the third was at 120ft. Strangely enough, the test at 120ft completed communication 350ms faster than the 0ft test.

To continue the distance test with movement behind static structures, two identical tests were performed with a moving recipient that moved behind metal library bookshelves. Obstruction caused substantial packet loss and suffered notable speed loss as well.





For the noise test, tests were conducted outside with moderate to light foot traffic on Texas A&M campus between the Evans library entrance and an adjacent building. For the control test, the devices were placed at 0ft, then 40ft, and 120ft. For these tests, the recipient was placed in a backpack with an active laptop inside of it to provide additional noise. The test at 40ft experienced minimal packet loss while the test at 120ft experience substantial packet loss, which is believed to be due to the bluetooth, cellular, and wifi-based noise at similar frequencies to the ZigBee bands.

Something that we noticed in the outdoor noise test was that once our device got in range with bluetooth headphones, we experienced substantial packet loss, which leads us to believe that performing a brush pass procedure in close proximity to multiple bluetooth devices emitting a strong signal could invalidate a communication instance.

V. Spoofing Experiments

Spoofing experiments could not be completed since the CC2531 ZigBee packet sniffer used in tandem with Wireshark was unable to detect the ZigBee protocol in the experiments. After spending a long time flashing the device and reconfiguring the drivers for the device, it was disappointing to see that this portion could not be completed. After a failure with Wireshark, we attempted to use KillerBee, which had plenty of unresolved issues of its own found in the GitHub source code that we were using. We spent some time attempting to debug it, but had no luck doing so.

The KillerBee package is a pending project that allows the user to detect, reroute, and jam ZigBee based devices. This project is still in development, so this may benefit us in future works so that we are able to pair it with our CC2531 USB Stick.

Test Type	Duration (s)	Packets retained (%)
0ft Open	31.559	100%
0ft Recipient Cl.	31.542	100%
0ft Both Cl.	31.562	100%
40ft Indoor	31.542	100%
120ft Indoor	31.542	N/A%
0ft Outdoor	31.584	100%
40ft Outdoor	31.542	100%
120ft Outdoor	31.543	97.1%

TABLE II
Speed and Quality of each Static Field Test- Each test used a 3.561kb txt file

VI. Conclusion

The novelty claims for this project are an analysis of burst communication with ZigBee and file transfer over ZigBee as well as the design and construction of a device that can consistently communicate back and forth using ZigBee protocol.

As a test following the presentation, we were able to successfully transfer the same file used in the ZigBee tests with DigiMesh. The code execution took substantially less time with DigiMesh, so DigiMesh would likely be a more promising protocol for transfer if the work is to be continued. The table below shows the latency

Latency(ms)	Duration (s)	Packets retained (%)
500	25	100%
250	12.5	100%
125	6.2	100%
64	3.2	100%
32	1.6	74%

TABLE III
DIGIMESH LATENCY TEST

between devices on DigiMesh.

VII. FUTURE WORKS

Finally, to continue the work, work with Wireshark, EtherApe, or another application would be continued to manually detect and interfere with the signal. Interference could also potentially be performed with some kind of signal scrambler running at the same frequency without the need to detect it. One means of interruption that was discussed was using bluetooth headphones or other bluetooth-based devices to interrupt the communication since we experienced packet loss due to bluetooth devices in the outdoor experiment. Some other interruption software for bluetooth could also be used to interrupt, such as bss or bluefog. Bss is a project in the BlackArch suite that allows for the fuzzing and stack smashing of a bluetooth signal. Since Bluetooth and ZigBee both run in the 4.2GHz range, this could be used to interrupt communication. Bluefog is another piece of the BlackArch suite that allows numerous devices to be searched, then the names are duplicated multiple times to create fake devices which can become honeypot exploitations. Secondly, we would buy more expensive and more reliable hardware. An issue with ocurred with some of the hardware where one of the chips repeatedly overheated, which caused packet loss in the earlier stages of the experiment. Alex had to narrow down the hardware issues to get this working again. Third, having to wait 30 seconds to transfer such a small file in ZigBee is a lot of seemingly unnecessary time, so in the future, this bottleneck would be investigated.
Further experiments were conducted with both XBee pro boards using Digimesh instead of instead of Zigbee to compare performances. We were able to get impressive results for a small file transfer (around 3.5kB), much better than with Zigbee. DigiMesh's maximum packet size is of 256 bytes, twice the size of Zigbee's 128 and supports latencies of minimum 60ms as opposed to zigbee where we used 500ms between each packet. It seems that digimesh would be a better candidate for the kind of operation. We did however notice significant packet loss on files over 10kB large as well as heating of the FTDI serial converter. We should therefore retry all of our experiments with a higher quality product, add a heat sink on the FTDI converter or skip it altogether.
Another thing to consider when looking at our project is the operability and more specifically in an urban environment. Our project is conveniently fitted inside a Pelican 1020 box since it was transported in a bag across campus and passed along from one project member to another quite frequently. In order to minimize detection, future versions would need to implement some kind of urban camouflage. Ideas currently being explored involve a Starbucks cup, a notepad or a book, a toolbox or a paper sandwich bag.

VIII. GITHUB

Our code can be found on github. It includes the programs for sending, receiving and a ping script to run the sending script once the other device has been found and authenticated.
https://github.com/thessnelson/BuzzPass

REFERENCES

[1] H. Pirayesh, K. Sangdeh, and H. Zheng. "Securing ZigBee Communication against Constant Jamming Attack Using Neural Network", IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4957-4968, March 2021.

[2] T. Centers, “ZigBee Security Research Toolkit“, River Loop Security, github.com/riverloopsec.killerbee, March 2022.

[3] G. Combs, A. La Goutte, et al, “Wireshark“, Wireshark, https://github.com/wireshark/wireshark, April 2022.

[4] K. Kanters, “Zigbee2mqtt.io Documentation“, Zigbee2MQTT.io, https://github.com/Koenkk/zigbee2mqtt.io/tree/master/docs, April 2022.

[5] S. Ergen “ZigBee/IEEE 802.15.4 Summary, The University of Wisconsin-Madison Department of Electrical and Computer Science, September 2004.

[6] I. Kuzminykh, A. Snihurov and A. Carlsson, ”Testing of communication range in ZigBee technology,” 2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2017, pp. 133-136, doi: 10.1109/CADSM.2017.7916102.

[7] W. Wang, F. Cicala, S. Rafiul Hussain, E. Bertino, and N. Li. 2020. “Analyzing the attack landscape of Zigbee-enabled IoT systems and reinstating users’ privacy.“ In Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec ’20). Association for Computing Machinery, New York, NY, USA, 133–143. DOI:https://doi.org/10.1145/3395351.3399349

[8] P. Seong-Ro, R. Seong-Ro and L. Seong-Ro. “Performance Analysis of XBee ZB Module Based Wireless Sensor Networks.“ International Journal of Scientific & Engineering Research. 4, 2013.

[9] J. Olson, “To Catch a Spy: The Art of Counterintelligence“, Georgetown University Press, May 2019.

[10] C. Wheelus, X. Zhu, “IoT Network Security: Threats, Risks, and a Data-Driven Defense Framework“, IoT pp. 259-285, https://doi.org/10.3390/iot1020016, October 2020.

[11] K. Tsiknas, D. Taketzis, K. Demertzis, C. Skianis, “Cyber Threats to Industrial IoT: A Survey on Attacks and Countermeasures“, IoT pp.163-186. https://doi.org/10.3390/iot2010009, March 2021.