Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio

Leslie Choong
University of California, Los Angeles

Software Defined Radio enables new protocols that are flexible and quick to deploy. Researchers can use the programmability of Software Radios to create tools that monitor and debug their work. In this paper, Open-source hardware and software is used to deploy a multi-channel monitoring tool of the IEEE 802.15.4 protocol.

Additional Key Words and Phrases: SDR, Zigbee, 802.15.4

1. INTRODUCTION

Many new digital communication technologies are looking towards Software Digital Radio (SDR) as a platform. SDR offsets the modulation and demodulation of the signal that is normally done on hardware on to a software platform. Since the core logic of the protocol is implemented in software, changes are easily implemented and complicated protocols can be implemented and field-tested in a shorter amount of time than the traditional development approach.

Cognitive Radio involves the development of protocols that are able to assess the environment they are operating in and switch operating characteristics to minimize interference and maximize throughput. By using SDR, Cognitive Radio is able to efficiently monitor and make changes to how the signal is modulated and demodulated in real-time. These changes are easy to make since the logic is implemented in software.

The Zigbee standard is aimed at Low Rate Wireless Personal Area Networks (LR-WPAN) that are cheap and simple to implement. Zigbee uses the IEEE 802.15.4 standard to implement the lower levels of the protocol. One of the frequency bands that Zigbee operates in is the unlicensed 2.4 GHz band. Since this band is unregulated, there is a high possibility of interference from other devices. A growing concern is interference caused by Wireless Local Area Networks (WLAN) which operate in the same frequency band at a much higher power level. Cognitive Radio techniques can be used to have Zigbee nodes switch channels if they encounter too much interference.

GNU Radio is an open-source effort to implement signal processing blocks that can interface with Software Radios. By using the GNU Radio code, software mod-

Author's address: L. Choong: septikus@cs.ucla.edu

ulation and demodulation of signals can be done on commodity computers. With computer processing power increasing and multi-core computing becoming more accessible the ability to create powerful Software Radio applications is getting easier and easier.

The signals that GNU Radio works with must come from or be sent by some physical hardware. The GNU Radio community developed the Universal Software Radio Peripheral (USRP) as a result of this requirement. The USRP has very little built-in signal processing hardware. Signals are sampled by an Analog-to-Digital Converter (ADC) and are passed on to computer with a minimal amount of processing. Like-wise, the USRP also supports transmission with a Digital-to-Analog Converter (DAC).

The topics just introduced lay the framework for the focus of this paper. In developing novel protocols and also to observe wireless network activity it is useful for researchers to have access to data-gathering and visualization tools. This paper presents the result of using GNU Radio and the 2nd version of the USRP (USRP2) to capture Zigbee packets on multiple channels. These captured packets can then be analyzed and visualized using packet monitoring tools such as the open-source project Wireshark. The ability to record and watch and record Zigbee channel activity is a powerful tool for researchers investigating and developing channel hopping protocols for Zigbee devices.

The motivation for using SDR to capture these packets is to demonstrate the current ability of SDR on commodity systems. Future systems may incorporate the IEEE 802.15.4 packet capture abilities with other protocols that would be harder to do in a pure hardware platform. SDR gives us the power to be flexible and perform many different types of signal processing all on one platform.

The rest of this paper is organized as follows. Section 2 will cover the development of SDR as well as some previous projects that utilized SDR. Section 3 will cover the Zigbee and 802.15.4 protocol and how it works. In Section 4, GNU Radio architecture will be covered and Section 5 will cover the USRP/USRP2 hardware platform. Section 6 will analyze Wiresharks architecture and show why it is a good fit for researchers looking for a network monitoring solution. Section 7 will discuss previous packet capture work involving IEEE 802.15.4. Section 8 covers the implementation of a real-time multi-channel packet capture solution. Section 9 discusses the evaluation method and environment. Section 10 will present results of the implementation. Section 11 will present some future work. Section 12 will have some concluding remarks on this project.

2. SOFTWARE DEFINED RADIO

Software Radio attempts to create a flexible platform by using software instead of traditional hardware to perform operations on signals. The ideal SDR platform would use as little hardware as possible and let software deal with all of the processing. An ideal receiver might have just an antenna connected to an ADC. Samples would then be read from the ADC and software would handle all signal processing.

There are many benefits to using software to process signals. Software is quick to compile and load, which gives iterative development a much higher cycle rate. Researchers and developers who are looking to experiment with different filter ap-

proaches can quickly prototype solutions and have real world results using SDR. Software that is developed for signal processing can easily be shared and adapted by others, allowing new work to re-use existing code. This code re-use is easily demonstrated by the GNU Radio community which is described in detail later.

Using commodity Personal Computers (PCs) is also advantageous because platform developers can take advantage of the high performance to economy ratio. Fast super-scalar processors with multiple cores are becoming commodity resources. This make high power computing resources available at a good price. Although specific signal processing hardware may be faster, a general purpose software solution has a place in many radio applications.

SDR has advantages to manufacturers because they only need to produce and support one hardware platform which can cut development costs. For service providers using SDR, they can quickly update and change their network with little hardware change. End users of SDR products will be able to communicate efficiently and have the most up to date features without having to buy a new hardware platform each time.

The flexibility and reconfigurability of SDRs make them prime targets for military and disaster response applications. In these scenarios, an existing communication infrastructure may be damaged or not even exist at all. Having a communication device that can adapt to changing communication conditions would allow for better response and coordination.

2.1 Military use of SDR

One product that was built using these principles was the SPEAKeasy system. SPEAKeasy was an attempt by the United States military from 1992 to 1995 to produce a modular and reprogrammable modem that could be used in the battlefield. The first phase goal was to demonstrate a multi-channel wide-band architecture for frequency hopping and spread spectrum signals. [Cook and Bonser 1999]

The success of the first phase led to the development of SPEAKEasy phase 2. The goal of phase of SPEAKeasy was to create an entirely open radio architecture from RF to user input and output. Although slated to be a 4 year project, early tests about 15 months in showed enough promise that SPEAKeasy phase 2 was pushed into production even without all features implemented.

More recently, the US Army is developing the Joint Tactical Radio System (JTRS) that seeks to consolidate legacy and new radio platforms into one. [JTRS 2009] The project is interesting because many of the specifications are openly published. The work done by the military with projects such as JTRS can trickle down into the commercial sector as well.

2.2 Civilian Disaster Uses

Another use case for SDR capable devices is for communication in disaster areas where communication infrastructure has been damaged. The SDR Forum, which is an interest group dedicated to promoting SDR, released an analysis of the London Subway bombings that occurred on July 7th, 2005 and how SDR communication devices could have aided in the response to the incident. [SDRForum 2007]

As the bombings were located in the subway system at different stations, rescuers had to approach the scene through the closest open station. By the time they

arrived on scene there was no way to communicate except by sending a runner to relay a message. The SDR Forum pointed out that radios with the ability to reconfigure themselves to act as repeaters could have been used to greatly reduce the latency in communications during the incident. These radios would need to do this automatically based on the environmental signal conditions experienced.

2.3 Cognitive Radio

The ability to reconfigure a communication network based on environmental conditions is the key to the field of Cognitive Radio. SDRs give the flexibility to change the underlying communication blocks used, while cognitive radios evaluate the operating environment and initiate the changes in the radio to maximize the goals of the radio.

Of particular interest and one of the goals of the Cognitive Radio community is dynamic spectrum access. With more wireless technologies being developed, the available spectrum is becoming more and more limited. However the usage of the spectrum varies greatly with time and location. It is of great interest to primary users and secondary users of the spectrum on how to effectively share these under-utilized bands.

A look at the FCC spectrum allocation chart shows multiple allocations of the spectrum[FCC 2009]. However, analysis of some of those bands show major under utilization. This suggests ample opportunity for new devices to make use of the under utilized spectrum as long as they do not interfere with the primary users of the spectrum.

One simple approach to spectrum sharing is to allow nodes to switch to different bands or channels if they encounter interference from other networks. Simple protocols such as this are a good fit for distributed low power nodes as they do not require much extra computation. More specifics on these adaptive radio techniques to avoid interference are covered after first discussing the Zigbee / IEEE 802.15.4 architecture.

3. ZIGBEE AND IEEE 802.15.4

The IEEE 802.15.4 standard defined the physical and MAC layers for a LR-WPAN. Its main goal was to create a low data rate protocol for low power applications. The Zigbee Alliance built on top of the IEEE 802.15.4 protocol by further defining the higher layers of the stack and releasing the Zigbee protocol.

3.1 Applications of Zigbee

The Zigbee Alliance aims to create interoperable products for home and industrial use. With a built in Zigbee device these products can transmit sensor data, sensor health, commands, or updates wirelessly. By following the Zigbee standard, devices from different manufacturers will be able to inter-operate.

One example involving Zigbee technology is a home control center[Adhoco 2009]. The center console aggregates sensor data of room illumination, humidity, temperature and air movement. Using the gathered data, the home control system can automatically control the lights, blinds, and air conditioning to optimize the home environment. This could be used in future energy-efficient homes and buildings where rooms are kept at optimal lighting and temperature with the minimal

amount of resources.

3.2 IEEE 802.15.4 Standard

The IEEE 802.15.4 standard supports 2 different network topologies that are useful for a wireless network: the star topology or the peer-to-peer topology. In the star topology a single node is selected to be the Personal Area Network (PAN) coordinator. All other nodes associated with the network must communicate through the coordinator. The PAN coordinator may be node with more computing resources and may be mains powered. The participating nodes are likely to be battery powered. Such as setup would be useful in home automation applications where there is a central control point.

In the peer-to-peer topology any node can communicate with any neighboring nodes within reception range. There is still a PAN coordinator, but the peer-to-peer model allows for more complicated mesh network topologies. A peer-to-peer mesh would be useful in more spread out environments, such as industrial production, inventory tracking etc.

At the PHY layer there are 3 bands that IEEE 802.15.4 is defined for:

—868 MHz (Europe)
—915 MHz (North America)
—2.4 GHz (World-wide)

Different frequency regulations mean different bands are available in different geographical locations. The 2.4 GHz IEEE 802.15.4 band has seen major development as it is available world-wide. As a result, this paper discusses capturing data in the 2.4 GHz band. There is more information regarding specifics not covered in this paper in the openly published specifications[IEEE 2006][Instruments 2007].

The 2.4 GHz band contains 16 channels that start at channel number 11 and go to channel number 26. Channels 0 to 10 are allocated to the 868 and 915 MHz bands. Channels are spaced 5 MHz apart with a spectral window of 2 MHz. The center of each channel is as follows where k is the channel number:

$$F_c(k) = 2405 + 5(k - 11)MHz, \forall k \in \{11, ..., 26\}$$

In the 2.4 GHz band, data is transmitted at a rate of 250 kbit/s. Transmitted data is first converted into 4 bit data symbols, which are then spread according to a predefined spreading sequence to a 32-bit chip sequence at a rate of 2 MChips/s. The chipping sequence is then modulated using Offset-Quadrature Phase Shift Keying (O-QPSK) and the resulting signal is sent out centered at the channel frequency.

A Physical Protocol Data unit (PPDU) frame is structured as in Figure 1. Within the PPDU there is a synchronization header, a frame length field and then the MAC Protocol Data Unit (MPDU). The frame length field is composed of 7 bits, meaning that IEEE 802.15.4 packets have a maximum MPDU size of 127 bytes. The MPDU contains the Frame Control Field (FCF), sequence number, address field, frame payload, and finally the Frame Check Sequence (FCS). The fields before the payload contain metadata regarding the contents of the payload. The FCS is an important first check in ensuring the integrity of the data. Protocols higher up in the stack may make extra checks for data integrity, but at the lower level it is implemented as the 16-bit CRC checksum of the MPDU.

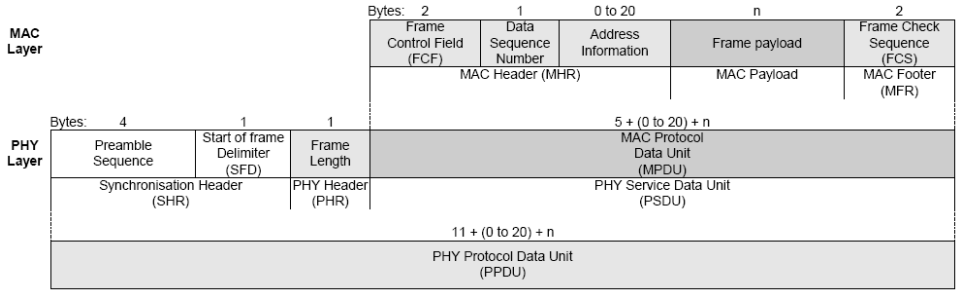

Fig. 1. Frame Layout of IEEE 802.15.4 Packet

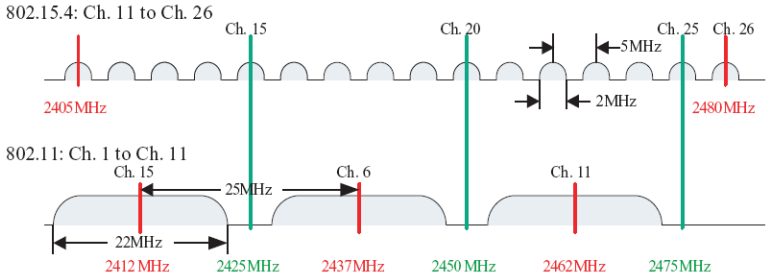

Fig. 2. Spectrum Overlap of WLAN and IEEE 802.15.4

3.3 802.15.4 ISM Band Co-Existence

The 2.4 GHz band is part of the Industrial, Scientific, and Medical (ISM) band. This unlicensed piece of spectrum is used by many different communication standards. Common technologies that operate in this spectrum are IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (Zigbee), and IEEE 802.11b/g which is for Wireless Local Area Networks (WLANs).

Since these technologies are likely to be in the same environments and have overlapping bands a concern has risen over interference between the different technologies. Figure 2 shows the overlap between WLAN and IEEE 802.15.4. Of particular concern is the performance of IEEE 802.15.4 in the range of WLAN devices. The power output of WLAN devices is much higher than that of IEEE 802.15.4 nodes. Many different studies have been done regarding the effects of WLAN on IEEE 802.15.4 communications.

The findings show that IEEE 802.15.4 channels that were centered within 7 MHz of the WLAN channel center experience significant frame loss[Petrova et al. 2006]. Unsurprisingly, longer packets showed a higher Packet Error Rate (PER) as there is more chance for corruption with a longer packet. The effects of IEEE 802.15.4 on WLAN throughput was found to be insignificant.

A number of techniques have been proposed to enable IEEE 802.15.4 to continue operating despite the possible spectral interference. All of these techniques involve channel hopping in order move the channel that the LR-WPAN is running on to a channel that has less interference.

One such protocol for wireless industrial sensors known as WirelessHART uses a Time Synchronized Mesh Protocol(TSMP)[Pister and Doherty 2008]. This protocol uses time synchronized nodes in order to coordinate channel hopping in a pseudo-random sequence. Although this protocol does hop channels, it does not do so based on channel interference.

A more adaptive approach to dealing channel interference has been presented by a few groups[Kang et al. 2007][Yun et al. 2008]. These protocols all take a cognitive radio approach. The PAN Coordinator will sense the current spectrum environment and then make decisions to switch channels based on the measurements. Maintaining the LR-WPAN while still hopping channels becomes the difficult part of implementing these protocols. Observing these protocols in the real-world is an important part of developing these protocols. To implement a SDR solution this paper will discuss the architecture of GNU Radio next and show how it can be used to create a SDR capable of packet capture.

4. GNU RADIO

GNU Radio is an open-source effort to create software that uses a minimal hardware platform to implement a radio[Blossom 2009]. The project aims to make SDRs easy to program and accessible to a larger group of users. The community is active and new developments are constantly being released.

A block based programming model is taken with GNU Radio. Different signal processing blocks can be connected together in a signal processing pipeline. These blocks can be composed of other blocks in a recursive fashion. The ability to easily reuse existing code and swap blocks makes creating an application with GNU Radio quick. Since processing logic is implemented in software, to test changes a recompile is all that is needed.

Blocks can be written in C++ or Python. Normally C++ provides the simpler block functionality and Python is used to compose and glue blocks together. In order to interface the C++ blocks with Python, a C++ wrapper and interface generator called SWIG is used. Using the combination of Python and C++ allows programmers to create logic that runs at the speed of compiled code, while easily connecting these blocks with a scriptable language.

Recently GNU Radio has also added Multi-Core support and is able to thread different blocks onto different processing units, taking advantage of the growing number of multi-core computers. At the moment this threading is taken care of by GNU Radio although the programmer may be given some control over how the blocks are threaded.

Different projects have already been built using the GNU Radio architecture. For example, a few different projects have implemented a IEEE 802.11b receiver capable of receiving low data-rate transmissions. Another project has implemented a GMSK receiver able to communicate with certain satellites. The community is constantly improving and adding features to make development of projects using GNU Radio even easier. Currently GNU Radio operates on data in a stream. Certain applications would be better suited if processed in a packetized manner. This is one of the next areas of development for GNU Radio.

Having the software architecture to do signal processing is not useful without a

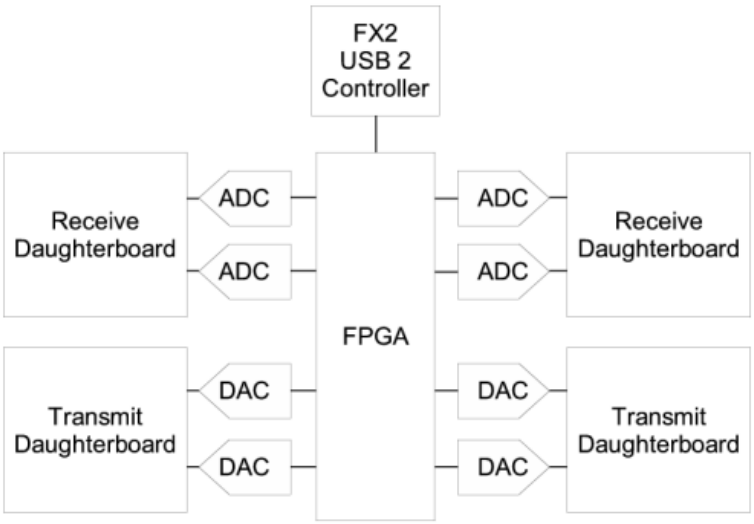

Fig. 3. Block Diagram of USRP

way to get real-world samples of the signal. Some users have tried with some success to use the Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) built in to their sound cards. However the performance and capabilities of these components severely limited the potential projects. For this reason the GNU Radio community also created an open hardware platform to interface with the world.

5. UNIVERSAL SOFTWARE RADIO PERIPHERAL

The original version of the USRP, developed by Matt Ettus[Ettus 2009], was meant to be a simple and flexible platform for software radios. It contains ADCs, DACs, and a FPGA which can be programmed to do some signal processing and is used to implement the Digital Down Converter (DDC) block. The USRP also contains a USB2 controller that is used to send and receive samples from the computer. Figure 3 shows a high level representation of the USRP. Some basics of the USRP architecture that a pertinent to design decisions are covered here and more details can be found in [GNURadio 2004].

Daughterboards act as a Radio-Frequency (RF) frontend capable of tuning to different bands depending on the Daughterboard used. A daughterboard down converts a signal to an Intermediate Frequency (IF) that the ADC can handle. There are a variety of daughterboards for different bands, such as the RFX2400 for frequencies from 2.25 GHz to 2.9 GHz, and the DBSRX which is capable of tuning from 800 MHz to 2400 MHz.

On-board ADCs are 12-bit Analog Devices ADCs with a sample rate of 64 MS/s. The DACs are 14-bit and operate at 128 MS/s. The complex samples are composed of 16-bit In-phase and 16-bit Quadrature samples for a total of 4 bytes per sample.

These samples must be transferred to the computer over the USB2 link for further processing by the host computer.

The USB2 specification has a maximum rate of 60 MB/s. However, in real world tests this maximum is not achieved for sustained transfers. The USRP is limited to transferring at most 32 MB/s over the USB2 link. This means we must decimate the signal by at least a factor of 8 in order to keep a constant stream of samples. Decimation is achieved by DDCs implemented in the FPGA. These DDCs center the signal at 0 and perform sample decimation. This decimation rate can be controlled by initialization parameters when interfacing with the USRP using GNU Radio.

A minimum decimation of 8 of 64 MS/s gives us 8 MS/s (complex samples), which gives us an equivalent sampling window of 8 MHz. This window is only wide enough to cover 2 IEEE 802.15.4 channels in the 2.4 GHz band at once. This limitation led to choosing the USRP2 for multi-channel decoding.

The USRP2 released by Matt Ettus is a more powerful version of the USRP. The design is similar to the USRP and is compatible with the original daughterboards. Changes include using ADCs capable of 14-bit 100 MS/s, DACs capable of 16-bit 400 MS/s, and a Gigabit Ethernet (GigE) link instead of USB2. GigE has a maximum transfer rate of 125 MB/s which is equivalent to about 30 MS/s. Thus, the USRP2 has a sampling window maximum of 25 MHz when using a decimation rate of 4. This sampling window is wide enough to cover 5 consecutive IEEE 802.15.4 channels in the 2.4 GHz band. Although there are 16 total channels in the ISM band, using the USRP2 for a maximum of 5 channels is a good start to seeing the power of SDR. As technology improves, a SDR capable of sampling the entire band of 16 channels is a possibility.

The previous sections have gone over the architecture to create a SDR capable of capturing IEEE 802.15.4 packets in the ISM band. Creating an application to view these captured packets and generate statistics would be of great help to researchers and developers trying to observe and troubleshoot their protocols. The open source project Wireshark provides that packet analysis functionality.

6. WIRESHARK

Wireshark is an open source network real-time packet analyzer. It features detailed packet dissection, display filters, and statistical plug-ins. The community supporting and developing Wireshark is very active[Wireshark 2009]. Although Wireshark is primarily used to explore and analyze TCP/IP networks, it still provides a very good framework for analyzing any type of packetized network. Since the code is freely available for modification by the public it is easy to build upon this framework and extend it for packet analysis of IEEE 802.15.4 networks.

Figure 4 shows the architecture of Wireshark. GTK 2 is an open source library for building Graphical User Interfaces (GUIs) that is cross-platform and provides the graphical support. The Core block ties all the different pieces together. The Wiretap and Capture components deal with incoming packets coming in from the hard disk and the network respectively. Libpcap is a separate open source library to interface with different Network Interface Cards (NICs). Captured packets can come in through the libpcap interface from a NIC, or through the hard drive in a variety of simple packet capture file formats.

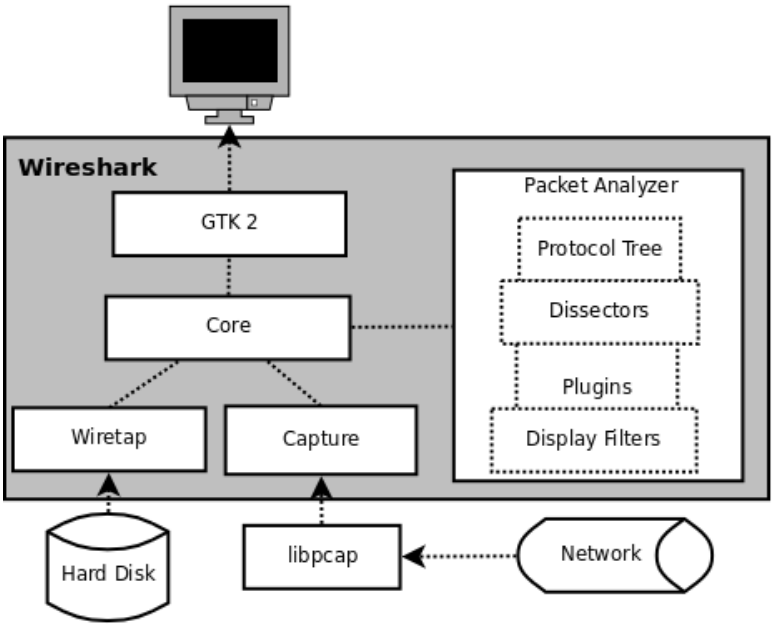

Fig. 4. Architecture of Wireshark

The packet analysis portion of Wireshark contains the logic for dissection of packet contents and gives statistic plugins access to packets as they are processed. Packet Dissection works like a tree with a packet being handed off to different dissectors which analyze different parts of the packet data. For instance, an incoming IEEE 802.15.4 packet may be dissected by a PHY layer dissector, then handed off to a MAC layer dissector, and then to a Zigbee protocol dissector and so on.

Statistic plugins are configured to be notified whenever a new packet is received. The dissected packet information is available for further information processing and statistic calculation. Many plugins already exist for Wireshark, such as a Input/Output graph, and Source/Destination time flow graph. These plugins use GTK to create a GUI to display results. These statistic plugins give packet analyzers the ability to build their own custom real-time visualizations of packet activity.

Within the last year, the support for the dissection of the MPDU of a IEEE 802.15.4 packet was added to the Wireshark project. This support allows Wireshark to recognize the different fields contained within the MPDU such as the sequence number, addresses, payload data, and the FCS. Display filters automatically work within Wireshark, allowing filter strings such as `'wpan.fcs_ok==1'` to only display received packets whose CRC of its contents match the FCS. Since Wireshark already has support for the IEEE 802.15.4 MPDU, additional layers to dissect higher layers of the Zigbee protocol could be easily added.

Wireshark's list of built-in packet analysis features and support for IEEE 802.15.4 packets makes it an attractive option for building a graphical packet analyzer.

7. PREVIOUS PACKET CAPTURE WORK

There are currently a number of different packet analysis tools available. They vary in the number of channels they can capture and the types of visualizations they can produce. Texas Instruments has released their Packet Sniffer that works with their development kits[Instruments 2008]. It is capable of monitoring one channel at a time and can report the Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) metrics that are returned by the hardware platform for each packet. RSSI is a measure of the energy contained in a received signal and is reported in dBm. The IEEE 802.15.4-2006 specification only specifies that the LQI must range from 0 to 255 and should relate to the relative signal quality of the channel. On TI transceiver chips such as the CC2420, LQI is calculated by averaging the chip correlation value of 8 consecutive symbols.

There are also some commercial single-channel analyzers such as the Microchip ZENA and the Frontline IEEE 802.15.4 Packet Analyzer. These perform similar functions to TI's Packet Sniffer although they do not feature reporting RSSI. They work with a provided hardware dongle.

Perytons produces a number of packet capture solutions. The Peryton-M is capable of capturing all 16 channels of IEEE 802.15.4 simultaneously. Some other work has been done to create a hardware array of transceiver chips in order to receive all channels simultaneously[Ban et al. 2007]. The hardware array used custom boards that contained the Texas Instruments CC2420 IEEE 802.15.4 transceiver chip. By linking 16 boards together they were able to listen on 16 channels in the ISM band. They built a GUI around the received packets and produced a functioning packet analyzer.

In the interest of exploring IEEE 802.15.4 demodulation and modulation with SDR, Thomas Schmid created a block for GNU Radio[Schmid 2006]. This implementation worked well for a single channel and because of the block design could be adapted to multiple channels. Figure 5 shows the architecture of the GNU Radio demodulation block. Since the chipping rate of IEEE 802.15.4 is 2 MChips/sec and the clock recovery block needs at the minimum 2 samples to decode each chip the input to this block is a 4 MS/s stream. The quadrature demodulation and clock recovery blocks are standard blocks provided by GNU Radio. The IEEE 802.15.4 packet sink takes the demodulated chip stream and builds packets following the specification. Using a state machine, the demodulated data is monitored for the synchronization sequence, packet contents are demodulated, and then the packet payload is forwarded to a callback function. More details can be found in the paper cited above.

Mikhail Tadjikov and Leslie Choong implemented a simple multi-channel demodulator using GNU Radio and a USRP[Choong and Tadjikov 2008]. Their implementation used Wireshark to display the output of the packets but not much was extended to show packet statistics or more advanced packet dissection. Due to the limitations of the USRP platform discussed above, this implementation was able to demodulate only 2 IEEE 802.15.4 channels simultaneously.

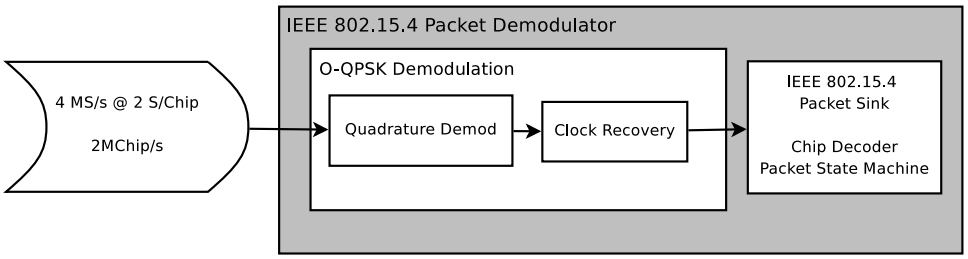

Fig. 5. Architecture of GNU Radio IEEE 802.15.4 Demodulation Block

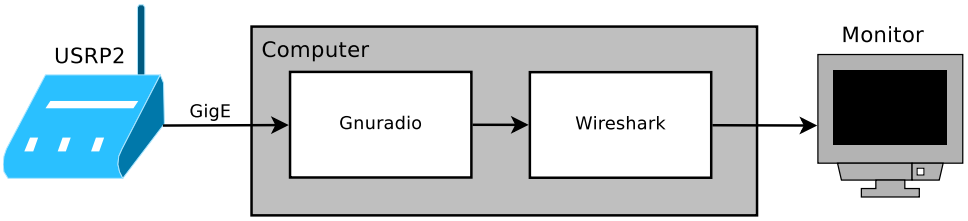

Fig. 6. Architecture of IEEE 802.15.4 Multi-Channel Packet Capture

8. PACKET CAPTURE IMPLEMENTATION

A more fully featured packet capture solution was implemented by extending the previous work done by Schmid, Tadjikov, and Choong. Figure 6 shows the high level structure of the solution. The USRP2 was chosen over the USRP because of its ability to sample a spectral window that contains 5 channels. GNU Radio was used to separate the sampling window into the different channels and demodulate the packets. Captured packets were then piped to a modified version of Wireshark which performs packet dissection and capture statistics.

Decimating at a factor of 4, the USRP2 streams 25 MS/s over the GigE link. By centering the USRP2 in the middle of a center channel as in Figure 7, a total of 5 IEEE 802.15.4 channels can be captured. These samples enter the GNU Radio block which is depicted in Figure 8. The incoming samples are routed into a power squelch block. This squelch block can be configured to drop all signals which do not pass a threshold strength in dB. Having a squelch block helps limit the amount of processing the host computer has to perform. Without it, the computer would constantly be using cycles to demodulate noise. Tuning this squelch filter to just above the noise level is important to make sure incoming packets are not dropped.

Samples from the squelch filter are then sent in parallel to 5 software based DDCs. As in the FPGA, these DDCs translate the signal by a frequency offset. After frequency translation, these blocks downsample and then low pass filter the signal to select a narrower band. Thus, these blocks act as a channel selection filter. Each block translates the signal by a different amount corresponding to the center of the desired frequency. The samples are decimated by a factor of 5 and low pass filtered to a 2 MHz window.

At the output of the DDCs, samples are being produced at a rate of 5 MS/s.

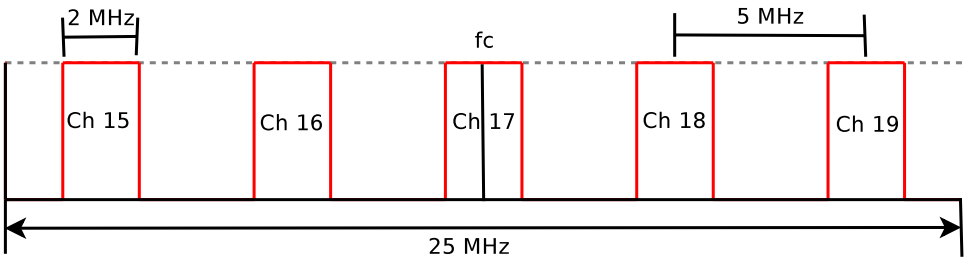

Fig. 7. Tuning the USRP2 to capture 5 channels. Center frequency denoted by fc

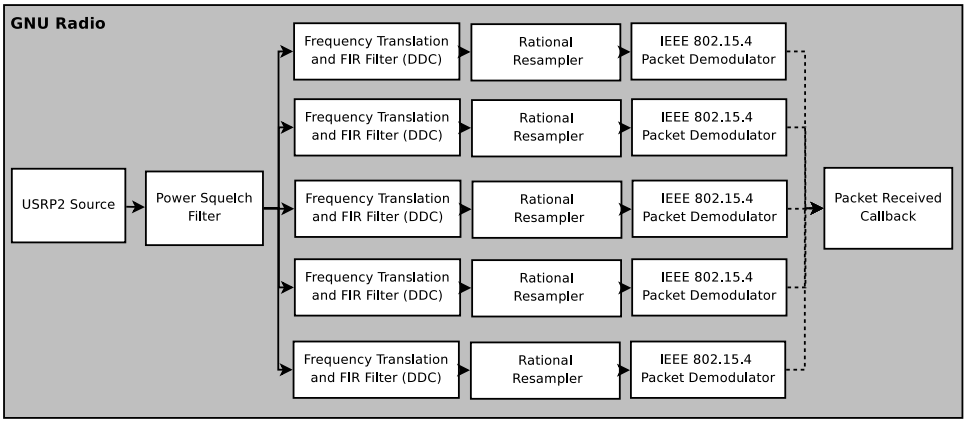

Fig. 8. Architecture of GNU Radio IEEE 802.15.4 Multi-Channel Demodulator

Since the IEEE 802.15.4 demodulation block requires 2 samples per chip and the IEEE 802.15.4 standard has a chipping rate of 2 MChips/sec we must resample the signal to the desired sampling rate of 4 MS/s. Therefore, the output of the software DDCs is sent to a rational resampler block which interpolates by a factor of 4 and decimates by a factor of 5 to produce a stream of samples at 4 MS/s.

The output of the resampler is then sent to a modified version of Schmid's IEEE 802.15.4 demodulation that was discussed before. The modifications calculate an LQI number for each packet that is demodulated. As mentioned before, LQI as specified by the IEEE 802.15.4 is just a value that ranges from 0 to 255 that reflects the quality of the signal that the packet was received from. The CC2420 implements this by averaging the chip error rate of the first 8 symbols after the SFD is detected. This result is then scaled to be between 0 - 255.

A similar system for calculating LQI in GNU Radio was implemented by summing the number of matched chips in the first 8 symbols after the SFD is detected. Since each symbol is composed of 32 chips, the maximum sum of 8 symbols is 256. The result is then put through a ceiling function of 255 inclusive and prepended to the demodulated packet.

Demodulated packets then exit the flow-based GNU Radio architecture by being forwarded to a packet received callback. This callback outputs the packet to a

buffer in the libpcap file format[Libpcap 1998]. The libpcap format is a packet capture format that can be read by different packet capture processing tools such as Wireshark. Each packet is timestamped with its receive time and the length of the packet.

Libpcap already has IEEE 802.15.4 MPDU packets as a capturable packet type. In order to support multiple channels changes were made to the libpcap format to add a PHY meta-data packet layer below the MPDU. The associated meta-data for each packet was the channel number as well as the LQI value for that packet. Wireshark was also modified to support the dissection of this new packet type. With the ability to dissect these packets, display filters can be used to limit the packets displayed to specific channels. All that is currently implemented is PHY and MAC layer dissection. Additional dissector layers could be added to analyze Zigbee packets due to the hierarchical dissection that Wireshark performs.

The existing IO Graph statistic plugin was modified to create a visualization for multi-channel data. Originally, a set of packets defined by a display filter was plotted to a graph as the number of packets received versus time. Five different filters would be plotted to the same graph. The plugin was modified so that each filter corresponded to the output of a different graph. This way, five different channels could be analyzed simultaneously over time.

9. EVALUATION METHOD

The implementation was developed and tested on a Macbook with a 2.4 GHz Intel Core 2 Duo processor with 2 GB of RAM running Ubuntu 8.10 with the 2.6.27-11 Linux Kernel. It also has a Marvell 88E8058 GigE chipset for interfacing with the USRP2. Telos motes running TinyOS were configured to send 120 packets at a rate of 4 Hz with a variable packet size from 14 bytes to the maximum 127 bytes. The Telos motes use the Texas Instrument CC2420 chip discussed above to modulate and demodulate packet data. The motes have an MPDU overhead of 11 bytes. The IEEE 802.15.4 channels used were channels 15 to 19 which is equivalent to 2.425 GHz to 2.445 GHz.

The GNU Radio multi-channel capture program was given the highest system priority and captured packets were written to a capture file. While captures were running, system idle percentage reported by `mpstat` was recorded and averaged over 6 seconds to get CPU usage metrics. The system idle reported was the average idle of both processors in the system.

Packet Error Rate (PER) was measured by dividing the number of packets missed or with an erroneous FCS by the total number of packets sent. The number of channels captured was changed from 1 to 5 with an equivalent sampling window to assess PER. Since packets are dropped if a corresponding chip sequence is not found, a longer packet has a higher chance of being dropped. Therefore packets of different lengths were sent in order to observe this behavior.

Telos motes loaded with the Base Station application were used to compare the packet capture performance of the implementation against a hardware platform. Received packets are forwarded by the BaseStation mote to the host computer. Transmitting motes were placed 3 ft away from the USRP2 and Base Station motes unless otherwise specified.

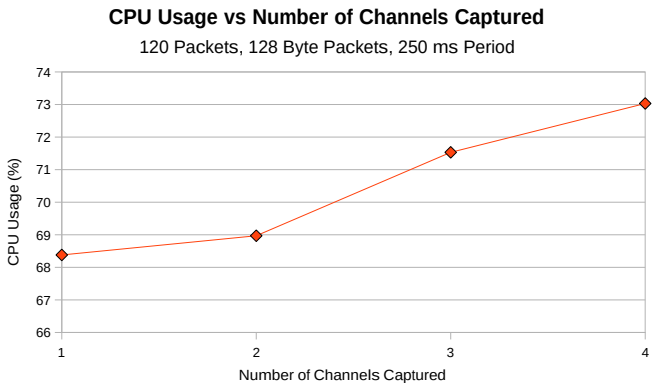

Fig. 9. CPU usage when capturing a different number of channels

10. RESULTS

When trying to decimate by a factor of 4 to get a window for 5 channels, sequencing errors were returned by GNU Radio. These sequencing errors indicate that samples sent over the GigE link were being dropped. This can occur if the host computer cannot process the data fast enough or if the network chipset cannot keep up with the data rate. USRP2 users have found that not all GigE chipsets are able to keep up with the 100 MB/s data rate that the USRP2 is capable of generating. However, when testing other sample GNU Radio applications that use a decimation rate of 4, no sequencing errors occurred. This implies that our host platform was unable to keep up with the computational load of demodulating 5 channels.

Since the host computer was unable to capture 5 channels the maximum number of channels to capture was lowered to 4. This was easy to do given the software based nature of GNU Radio. Samples were decimated by 5 to give a sampling window of 20 MHz. An almost identical architecture was used as above except there was no need for a rational resampler as the sampling rate after the DDC was at 4 MS/s already.

CPU usage was assessed for capturing a different number of channels. Figure 9 shows the results of these measurements. The CPU usage appears to scale linearly with the number of channels captured which is the expected behavior. if it does scale linearly, then there appears to be enough processing power to process 5 channels. However, the 4 channel demodulator does not use the rational resampler block. It is possible that the addition of the rational resampler increased the processing load enough to prevent the computer from demodulating packets without sequencing errors.

Next, Figure 10 shows the relation of PER with respect to the number of channels captured. Once again PER scales fairly linearly to the number of channels captured. This is expected since the packet demodulation is done in a very parallel fashion with the same demodulation code path for each channel. Therefore, as more channels are added the PER should increase at a linear rate.

For the 4 channel capture, PER was a little higher than expected because a single channel did not have very good capture performance. Channel 15 was found to have

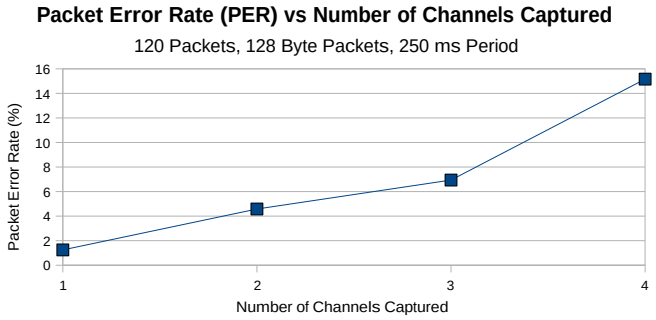

Fig. 10. Packet Error Rate when capturing a different number of channels

a lower received rate than other channels across all results. This could be due to mote variations or interference from other sources. Since the channel is divided up in parallel and the same code is used across all channels, there must have been a difference in the signal the USRP2 was receiving at that band.

The Base Station motes were able to achieve 0% PER during the test, receiving all 120 packets. The IEEE 802.15.4 demodulator matches chips by seeing if the number of wrong chips passes a certain threshold. If the number of chipping errors pass a threshold then the entire packet is discarded because of that one bad symbol. A more advanced chip correlator might result in less dropped packets. This addressed in the Section 11.

The PER was compared while varying packet size. This was tested while capturing 4 channels and while capturing a single channel. The result is shown in Figure 11. Of particular note is that the single channel performance is better compared to Schmid's results using the USRP. Schmid's paper did not specify the distance the transmitting mote was from the USRP and so results may vary. The USRP2 does have a 14-bit ADC compared to the 12-bit ADC in the USRP, which could contribute to a higher accuracy. For a single channel, the number of dropped packets was around 1% for the longest possible packet. Four-channel capture performance had a PER of 15% for the largest packet. Once again, channel 15 had a very high PER of 52%. This affected the PER average by a lot. Creating a more robust chip correlator would decrease the number of dropped packets.

Next the relation of the LQI metric to PER and distance was compared. A single channel demodulator was used in these tests as performance of more channels would scale similarly. First the PER was obtained when changing the distance of the transmitting mote and the USRP2 / Base Station. Figure 12 shows the result of that experiment. A large increase in PER occurs as distance increases. The Base Station mote was able to capture almost all packets, dropping just 2 total. The poor performance of the USRP2 is hard to track down. Since the mote is further away, the received power may be too close to the ambient noise and is being squelched. However, when removing the squelch filter for the single channel, no noticeable decrease in PER was observed. This poor distance performance is discussed further in the next section.

Figure 13 shows the relation of the calculated LQI value to distance. It shows

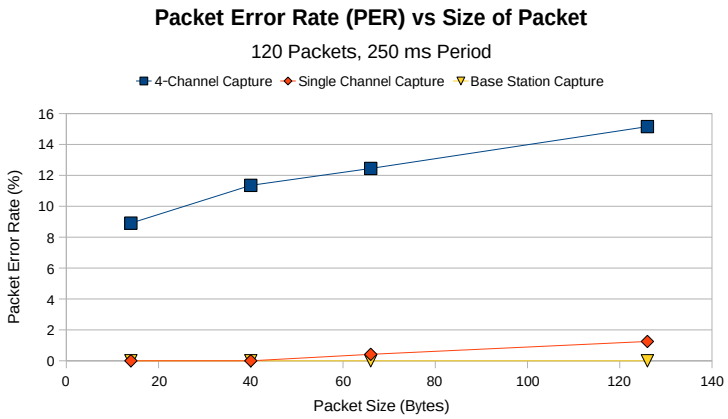

Fig. 11. Packet Error Rate when varying packet size

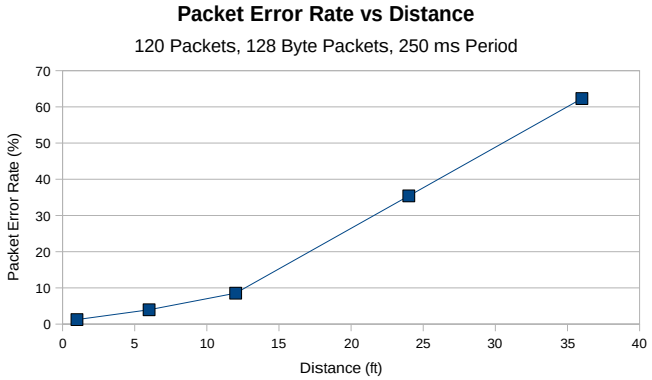

Fig. 12. Packet Error Rate when varying transmitting mote distance from receiver

an inverse relationship compared to the PER analyzed before. This is expected as LQI should decrease as the mote gets further away and signal quality deteriorates. However, the range of LQI is not very wide, going from 244 to 255. Better scaling and a more accurate chip correlator would lead to an LQI with a better range of values.

11. FUTURE WORK

As mentioned previously a different method to match chipping sequences could improve PER. Currently, the demodulator has a default threshold mis-match chips value. If the number of chips that do not match the symbol is less than the threshold then that symbol is considered demodulated. However, if the chipping sequence does not match any of the symbols because the threshold is surpassed then not only is that symbol not demodulated, but the entire packet is dropped.

A different implementation could find the symbol mapping with the lowest total mis-matches and decide that is the mapped symbol. This would mean every in-

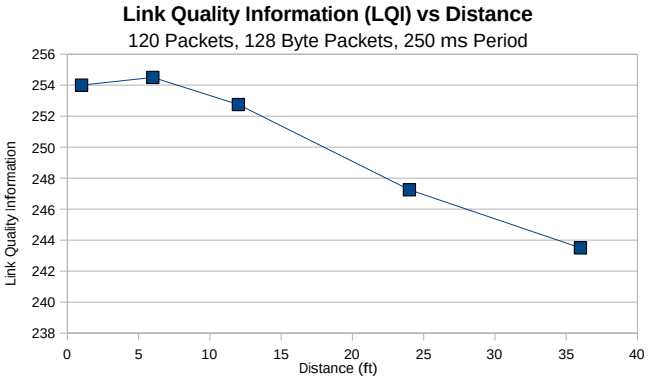

Fig. 13. Link Quality Information when varying transmitting mote distance from receiver

coming chipping sequence will get demodulated to some byte sequence. This would allow for packets with more severe errors to still be demodulated. In addition, the most likely symbol will be demodulated instead of just dropping the packet according to a hard coded threshold value.

In addition to a more robust chip correlator, the LQI value could be extended to be an average over the entire length of the packet as opposed to the first 8 symbols. Scaling the value to use the range of 0 - 255 would also make the metric more valuable in assessing link quality.

To obtain a different quality metric, GNU Radio is developing inband code that could send RSSI data from the ADC along with the samples. This would increase the amount of data sent over the link and therefore the number of channels demodulated would decrease or a different link would need to be used. Data such as RSSI would be another useful metric in measuring the quality and strength of a received channel.

It was noted that the PER increased quickly as the transmitting mote was moved further from the USRP2. Additional work investigating this behavior is definitely necessary. In order for the USRP2 to be considered a real development platform for applications using IEEE 802.15.4 this problem must be addressed. Squelching out the signal was ruled out as a possibility since the single-channel implementation was tested without the squelch filter with similar results. A possible reason for this poor performance could lie in the chip correlator. If changes were made as discussed it would be interesting to see the improvement in results.

If trying to sample a very large window, such as 80 MHz, to capture all 16 channels then a link capable of sustaining 320 MB/s would be necessary. The best option for this would be a PCI-Express interface, which is capable of 1 GB/s per lane. It is likely that future SDR platforms for wide bandwidth applications will need to move to this type of interface in order to support the amount of sample data that SDRs can produce.

GNU Radio has multi-threading support built in already. It would be interesting to see how a highly parallel application such as multi-channel capture scales across multiple cores and processors when even more channels are being demodulated.

Finally, for a true packet capture solution SDR could be eliminated completely. Using the Telos motes and the already implemented Wireshark packet analysis framework, a high performance packet capture and analysis solution could be created. This would be similar to what [Ban et al. 2007] did but with commercial Telos mote hardware. This does not preclude the development of IEEE 802.15.4 for SDR as there are many more applications that SDR and IEEE 802.15.4 could be used for.

12. CONCLUSION

Using the previous SDR work involving IEEE 802.15.4 demodulation a real-time multiple-channel packet capture solution was designed. This design was for 5 consecutive channels of the 16 channels located in the 2.4 GHz band. However, the multiple-channel architecture could be trivially scaled to demodulate all 16 channels once SDR and computational hardware is available.

Open-source software made it much easier to develop the packet capture solution. The signal processing for demodulation was implemented in GNU Radio and packet analysis was done in Wireshark. The produced solution showed good scaling performance as more channels were added. The flexibility of software based signal processing allowed the implementation of an initial link quality metric. More work remains to be done investigating USRP2 performance with transmitting motes placed further away.

With all signal processing and logic contained on a single platform, SDR opens up the possibility for novel protocols that observe and dynamically adapt to their environment. The IEEE 802.15.4 modulation / demodulation implementation, and the solution produced show the promise that SDR holds for the future.

ACKNOWLEDGMENTS

Thanks to Thomas Schmid for his support and work in implementing the IEEE 802.15.4 block in GNU Radio. Thanks to Dustin Torres for the many discussions regarding USRP architecture and signal processing. Finally, thanks to Mani Srivastava for his support in this project.

REFERENCES

Adhoco. 2009. Adaptive home control system. http://www.adhoco.com/.

Ban, S. J., Cho, H., and Kim, C. L. S. W. 2007. Implementation of ieee 802.15.4 packet analyzer. In *PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY*.

Blossom, E. 2009. Gnu radio. http://www.gnu.org/software/gnuradio/.

Bose, V., Ismert, M., Welborn, M., and Guttag, J. 1999. Virtual radios. *Selected Areas in Communications, IEEE Journal on 17,* 4 (Apr), 591–602.

Choong, L. and Tadjikov, M. 2008. Using usrp and gnuradio to decode 802.15.4 packets. http://www.cs.ucla.edu/ septikus/cs213/index.html.

Cook, P. and Bonser, W. 1999. Architectural overview of the speakeasy system. *Selected Areas in Communications, IEEE Journal on 17,* 4 (Apr), 650–661.

Ettus, M. 2009. Universal software radio peripheral. http://www.ettus.com/.

Ewy, B., Sparks, C., and Shanmugan, K. 1995. An overview of the rapidly deployable radio network proof of concept system.

FCC. 2009. Fcc spectrum allocation. http://www.fcc.gov/oet/spectrum/table/fcctable.pdf.

GNURadio. 2004. Exploring gnu radio and the usrp. http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html.

IEEE. 2006. Ieee 802.15.4-2006 standard. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.

Instruments, T. 2007. 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver. http://focus.ti.com/lit/ds/symlink/cc2420.pdf.

Instruments, T. 2008. Rf packet sniffer. http://focus.ti.com/docs/toolsw/folders/print/packet-sniffer.html.

JTRS. 2009. Joint tactical radio system. http://jpeojtrs.mil/.

Kang, M. S., Chong, J. W., Hyun, H., Kim, S. M., Jung, B. H., and Sung, D. K. 2007. Adaptive interference-aware multi-channel clustering algorithm in a zigbee network in the presence of wlan interference. *Wireless Pervasive Computing, 2007. ISWPC '07. 2nd International Symposium on*, –.

KOLODZY, P. 2005. Application of Cognitive Radio Technology across the Wireless Stack. *IEICE Trans Commun E88-B,* 11, 4158–4162.

Libpcap. 1998. Libpcap file format. http://wiki.wireshark.org/Development/LibpcapFileFormat.

Petrova, M., Riihijarvi, J., Mahonen, P., and Labella, S. 2006. Performance study of ieee 802.15.4 using measurements and simulations. *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE 1*, 487–492.

Pister, K. and Doherty, L. 2008. Tsmp: Time synchronized mesh protocol. In *Parallel and Distributed Computing and Systems*.

Schmid, T. 2006. Gnu radio 802.15.4 en- and decoding analyzer. In *NESL Technical Report*.

SDRForum. 2007. Use cases for cognitive applications in public safety communications systems volume 1: Review of the 7 july bombing of the london underground.

Wireshark. 2009. Wireshark packet analysis. http://www.wireshark.org/.

Yun, J., Lee, B., Li, J., and Hanr, K. 2008. A channel switching scheme for avoiding interference of between ieee 802.15.4 and other networks. *Computer and Computational Sciences, 2008. IMSCCS '08. International Multisymposiums on*, 136–139.