EMERGENCY SERVICE IN WI-FI NETWORKS WITHOUT ACCESS POINT ASSOCIATION

by

Manav Seth

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

August 2011

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Manav Seth**

has been approved by the following supervisory committee members:

Sneha Kasera	, Chair	**6/17/2011** Date Approved
Robert Ricci	, Member	**5/9/2011** Date Approved
Neal Patwari	, Member	**5/9/2011** Date Approved

and by **Alan Davis** , Chair of

the Department of **School of Computing**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Emergency "911" service is a critical function provided in the Public Switched Telephone Network (PSTN), cellular and Voice over Internet Protocol (VoIP) networks. Wi-Fi, despite its growing importance, has no such service. In this thesis, we develop a 911-like service for Wi-Fi-capable devices, enabling them to send emergency messages through any available hotspot or access point. Our service makes use of existing 802.11 management frames and does not require the client device to associate or authenticate with the access point; this makes it available even on protected networks to which the client would not normally have access, even encrypted ones. This design ensures maximum potential reach and usability, and helps to increase public safety.

CONTENTS

ABSTRACT . . . . . **iii**

LIST OF FIGURES . . . . . **vi**

LIST OF TABLES . . . . . **vii**

ACKNOWLEDGEMENTS . . . . . **viii**

CHAPTERS

1. INTRODUCTION **1**

1.1 Emergency Services: History and Importance 1
1.2 Some Issues Facing Emergency Services 1
1.3 Objectives of Research 2
1.4 Contributions of This Thesis 3
1.5 Highlights of Our Service 3
1.6 Some Use Cases of Such a Service 5
1.7 Outline of the Thesis 5

2. BACKGROUND **6**

2.1 Related Work 6
2.2 Guidelines for 911 in the United States 7

3. RESEARCH **9**

3.1 Architecture of the Service 9
3.2 IEEE 802.11 Framing in Detail 10
3.3 Emergency Message Frames 12
3.4 Implementation Details 14
3.5 Implementation on USRP 18
3.6 Current Limitations 19
3.7 Evaluation of the Service 20

4. SOFTWARE DEFINED RADIO **26**

4.1 Introduction 26
4.2 Universal Software Radio Peripheral 26
4.2.1 Motherboard 27
4.2.2 Daughterboards 27
4.3 GNURadio 27
4.3.1 GNURadio Architecture 28
4.4 GNURadio Signal Processing Block 29
4.5 Developing the Complete GNURadio Flow Graph 30
4.6 Existing IEEE 802.11 SDR Implementations 30

5. IEEE 802.11g IMPLEMENTATION ON USRP2 . . . . . **32**
5.1 Approach 33
5.2 Description of DSP Blocks 35
5.2.1 CRC-32 35
5.2.2 Scrambler and Descrambler 35
5.2.3 MAC and PLCP Header 36
5.2.4 Convolutional Encoder and Decoder 38
5.2.5 Interleaver 38
5.2.6 Symbols Mapping 39
5.2.7 Pilot Tone Insertion 39
5.2.8 OFDM Modulation 40
5.3 Some Results 40

6. CONCLUSION **43**
6.1 Conclusion 43
6.2 Future Work 43

REFERENCES **45**

LIST OF FIGURES

3.1 Basic architecture of the service 9

3.2 Generic 802.11 MAC frame 11

3.3 Frame control field 11

3.4 Role of the AP 15

3.5 Selecting an AP 16

3.6 Sending the message to an AP 17

3.7 Effect of prioritizing emergency messages on time to receive e-mail at PSAP . 23

3.8 Histogram of RTT per bit for sending emergency message on USRP without interfering traffic 25

4.1 Typical GNURadio graph [1] 28

4.2 GNURadio framework[1] 29

5.1 OFDM blocks present in GNURadio core 33

5.2 IEEE 802.11g near-commercial implementation 34

5.3 IEEE 802.11g chain implemented in GNURadio 35

5.4 IEEE 802.11g data scrambler 36

5.5 IEEE 802.11g PPDU frame format 37

5.6 Convolutional encoder K = 7 block operation 38

5.7 Time delta between successive received frames 41

LIST OF TABLES

3.1 Performance testing of various cases 21
3.2 Results obtained when not sending acknowledgements 22
3.3 Performance improvement by sending acknowledgements 22
3.4 Time taken to send an emergency message to AP 24
3.5 Time taken to send an emergency message to the receiver USRP2 25
5.1 Encoding details for different OFDM data rates 39
5.2 Some statistics of 802.11g transmission using USRP2 41
5.3 Some statistics of 802.11g conventional transmission 42

ACKNOWLEDGEMENTS

The past two years at the University of Utah has been an exciting and an awesome journey for me. Apart from the technical knowledge I gained through regular coursework, I got an exposure to different aspects of research through this thesis. This one page will not be enough to thank and express my gratitude to my advisor, Dr. Sneha Kasera. He has always encouraged and guided me since the time I met him. I clearly remember, and may well remember the one line he spoke to me on that day, all my life: "You are my student now. I will take care of you." He gave me the full freedom to explore different domains and come up with a research problem which excites me. I would also like to mention that he has been very patient in improving my writing skills, and has given me so many tips throughout this process.

I would also like to thank Dr. Rob Ricci and Dr. Neal Patwari for serving as my committee members, and for providing valuable feedback and suggestions. Rob in particular has been very cooperative and helpful by giving his thoughtful feedback at various phases of this research. Thanks also to Ann and Karen who served as my graduate advisors and helping me right through my graduate studies.

I would also like to thank my lab mates Arijit, Manas, Prathana, Saurav, Shobhit, Suchit and Vaibhav for making our work space such a wonderful place to work! Lastly, I will thank my parents for providing constant encouragement and motivation without having a clue about what I was up to. I do not think that I would have been even close to complete my graduate studies successfully without their blessings and support.

CHAPTER 1

INTRODUCTION

1.1 Emergency Services: History and Importance

Emergency "911" service is a critical function provided in the Public Switched Telephone Network (PSTN), cellular and Voice over Internet Protocol (VoIP) networks [2, 3, 4, 5, 6]. In the U.S., the first 911 system was installed in 1968. Since then, the E-911 service has undergone several improvements and has been deployed both for cellular and VoIP in addition to the fixed line phones. The E-911 automatically associates a physical address with the calling party's telephone number, and routes the call to the most appropriate Public Safety Answering Point (PSAP) for that address. PSAP is a call-center responsible for answering calls to an emergency telephone number for police, firefighting, and ambulance services. There are roughly 6100 primary and secondary PSAPs in the U.S. Currently, some of the emergency services commonly available are:

- Enhanced 911 Service [7]
- Cellular Enhanced 911 [8]
- VoIP Enhanced 911 [9, 10]
- Automatic Wireless Fire and Smoke Alarms [11, 12]

Emergency services are available to help the public in different scenarios. Some agencies like the ambulance and fire will mostly deal with ad-hoc emergencies and others can be useful in many different scenarios like the police. These days, there have been many other kinds of emergency services such as the coastguard, lifeboat, bomb disposal, emergency road services, disaster relief services, etc. to name a few. These emergency services often work together and usually maintain open lines of communication among them. Citizens should also consider it a duty for themselves to help such services when required.

1.2 Some Issues Facing Emergency Services

As mentioned, Emergency "911" service is a critical service provided in the PSTN, cellular and VoIP networks. The PSTN, though, is the phone network established and in

use for decades, and the way emergency services are offered on it also has largely remained unchanged. However, in the nineties, with the introduction and advancement of cellular phones, the implementation of emergency services and the nature of calls made to the emergency number started to change. The fact that mobile phones were not limited to a specific location made them far more useful than fixed lines in reporting emergencies, since the phones accompanied the user, and were readily accessible whenever an emergency occurred, either to the user or to someone else. The percentage of emergency calls being made from mobile phones increased, until well over half the total calls were being made from mobiles. But, there are still many places around the world including the United States where there is no cellular network though there are Wi-Fi networks available.

In such places, the only way to communicate with an emergency center is to find a PSTN phone or use a VoIP service, although VoIP is still evolving in terms of providing support for emergency services and even coverage. Hence, there is a need to develop an emergency service for Wi-Fi networks and we make an attempt to do so in this thesis.

1.3 Objectives of Research

Wi-Fi, despite its growing importance, has no E-911 type emergency service. Wi-Fi is currently used by over 700 million people and there are close to 750,000 Wireless Hotspots around the world [13]. In 2009, 800 million devices capable of accessing the Internet using Wi-Fi were sold. Furthermore, the International Telecommunication Union (ITU) estimated that mobile cellular subscriptions worldwide will reach approximately 5.3 billion by the end of 2010 [14]. Of these, nearly 1 billion will be equipped with high-speed mobile web access. Hence, since more and more people are choosing Wi-Fi as their main means of communicating, it is imperative that there be an emergency service made for Wi-Fi. One of the challenges in providing such a service for Wi-Fi networks arises from the fact that more and more Wi-Fi networks are now secure. This security requires Wi-Fi users (mobile devices) to be authenticated by Wi-Fi access points before any data, even emergency data, can be sent or received.

In this work, we build an emergency service in Wi-Fi networks that does not require any access point association or authentication. Using our service, Wi-Fi enabled mobile devices will be able to use any nearby access point or hotspot, secure or not, to send an emergency message, without going through any authentication or the association phases of the IEEE 802.11a/b/g/n protocols. At the same time, malicious users will not be able to misuse this service to access the Internet since this service will only allow a user to send an emergency

message to an appropriate destination, which will be decided by the service itself.

1.4 Contributions of This Thesis

Our thesis has the following contributions:

- **Build a new service:** This research shall enable any device equipped with a Wireless Interface Card to send an emergency or a distress message at any time to a Public-Safety Answering Point (PSAP) using any available 802.11 Wireless Access Point or a commercial hotspot having Internet access. The user will not be required to authenticate or associate with the access point and hence makes it possible to use even a protected or encrypted network to which the client would not normally have access, only for the purpose of sending an emergency message.
- We design, implement and evaluate the service on two platforms:
    1. **PC-based access point:** A PC with a conventional wireless interface card is used as an Access Point. We then implement our service on this access point.
    2. **Universal Software Radio Peripheral (USRP):** The PC-based implementation does suffer from a few limitations, which are highlighted in Section 3.6. Also, initially, while developing the prototype, we faced a few issues implementing the service on a PC-based access point primarily due to the closed-source nature of the wireless card firmware and drivers. Further, we wanted to have complete control over the MAC layer of the IEEE 802.11 protocol, which we could not achieve using conventional wireless interface cards. Hence, as a counter-measure, to solve these issues, we decided to use the open-source software defined radio, the USRP, to implement our service.

1.5 Highlights of Our Service

The main highlights of our design of the Wi-Fi emergency service are as follows:

- We minimize the time to send the emergency message, especially in the presence of high load in the Wi-Fi network. Our system chooses an access point for emergency message transmission based on the following factors - strength of the signal received (RSS) from the access points, the number of current associations at the access points, and the past history of failures of access points in transmitting the emergency message.

- We build simple reliability by requiring the access points to acknowledge, positively or negatively, the transmission of an emergency message from a mobile device to the PSAP.
- The mobile device wanting to send an emergency message is able to check if the access point indeed has Internet connectivity before attempting to send the message.
- Our service also supports sending text messages and attachments to the nearest PSAP.
- Our service is also capable of finding the approximate location of the user and reports it to the PSAP along with details like the MAC address and type of device used by the sender.
- To aid our implementation on the USRP, we also build an IEEE 802.11g compliant transmitter. Further, we have also built a partial receiver and are working on completing it as part of our future work.

We implement our emergency service on the Ubuntu Linux platform. We use laptops for mobile devices and desktops for access points. Our service requires us to inject modified management frames into the Wi-Fi network. Therefore, we require the wireless cards on the mobile devices to support the *monitor* mode[1]. We require the wireless card on the access points to support the *master* mode. In order to obtain a tighter control over our implementation, we also implement our emergency services on the Universal Software Radio Peripheral (USRP). To achieve this, we implemented an IEEE 802.11g compliant transmitter and also attempted to build the corresponding receiver for the USRP version 2. We also built a fully functional prototype of our service for the USRP2 based on BBN 802.11g code [15] modified to incorporate our service. Using our implementation, we run a variety of experiments under different settings. We find that our system can deliver an emergency message from a mobile device to a PSAP in 1.8 - 2.4 seconds, depending on the load on the selected access point and traffic in the network. We also conduct experiments in various settings using the USRPs and find that with no interfering traffic, we can achieve a Round-Trip-Time (RTT, total time to send an emergency message and receive its acknowledgement) of ≈ 2.5 seconds. We also compare the results of our service with the web-based Short Message Service (SMS) and find that, on average, we take far less time in delivering a message than the SMS service. Thus, our experiments indicate that our emergency service can be used in real settings.

[1]Monitor mode, or RFMON (Radio Frequency Monitor) mode, allows a computer with a wireless network interface card (NIC) to monitor all traffic received from the wireless network, as well as inject MAC frames in the wireless network.

1.6 Some Use Cases of Such a Service

An emergency service which could be used by the general public using any available wireless access point will have a lot of potential. Such a service will be useful when:

- Sending an emergency message at a crowded place, without the need of specialized emergency kiosks (e.g., at Airports).
- Sending an emergency message when there are no cellular signals (e.g., Movie theatres, underground rooms).
- This service can indeed be one of the fastest ways to send an emergency message or a distress signal.
- Can be used by handicapped people who are unable to communicate or call an emergency service.
- Also useful when the user has access to devices that are capable of Wi-Fi, but not other wireless (e.g., cellular) communication.

1.7 Outline of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 describes related work and some proposed standards for Emergency Telecommunications Services.

Chapter 3 describes the research conducted and the work done to implement the proposed Emergency Service and also some results we got from conducting experiments.

Chapter 4 discusses the emerging technology of Software Defined Radios and specifically about the USRP since we develop an IEEE 802.11g transmitter and receiver using an open source SDR: the Universal Software Radio Peripheral (USRP2).

Chapter 5 discusses the work done to implement a 802.11g transmitter and receiver on the USRP2 and GNURadio platform as a part of this thesis.

CHAPTER 2

BACKGROUND

2.1 Related Work

There have been many research studies in the past which aim at providing emergency services to users. The first 911 system was installed in Haleyville, Alabama in February 1968 as a way to quickly connect a subscriber to the local police station. This system did not identify the caller but did provide a means to access emergency services that had not previously been available. This system was quickly adapted and improved by other telephone companies to become the E911 system which provides both caller location and identification. A pioneering system was in place in Chicago by the mid-1970s, providing both police and fire departments access to the source location of emergency calls. Enhanced 911 is currently deployed in most metropolitan areas in the United States and Canada. In addition, Wireless Enhanced 911 and VIOP enhanced 911 have also been developed to provide the service to the cellular and VoIP users.

Currently, some of the emergency services commonly available are:

- Enhanced 911 Service [7]
- Cellular Enhanced 911 [8]
- VoIP Enhanced 911 [9, 10]
- Automatic Wireless Fire and Smoke Alarms [11, 12]

In cellular networks, most GSM mobile phones can dial emergency calls even when the phone keyboard is locked, the phone is without a SIM card, or an emergency number is entered instead of the PIN of the SIM card. Our service can be thought as analogous to this service. In our emergency scheme also, a user can send an emergency message across to a PSAP using an available access point even if the wireless device is not associated with that AP.

Furthermore, the FCC has advertised that it will update the current E-911 service and enable citizens to report crimes through text messages, and even allow users to send video streams from their mobile phones to emergency centers [16].

Other systems such as cellular ad-hoc relay for emergencies (CARE) [17] has been proposed which adds a functionality to relay an emergency call arising from a user outside the cellular coverage area via another user within the range of the network. There have been works like the WIISARD project [18] which describes research that explores the design of 802.11 networks enhanced to support data communications in disaster environments, especially for medical uses. In CodeBlue [2], the authors develop a wireless infrastructure for emergency purposes in medical care using low-power sensors. In a related work known as CR MAC [19], the authors propose to use cognitive radio sensors and dynamic channel assignment to come up with a mobile ad-hoc network for emergency purposes. In another work, which presents an overview of highway cooperative collision avoidance (CCA), [20] presents an idea of vehicle-to-vehicle cooperative communication protocol which aims at enhancing traffic safety on busy highways. In a novel work, AMBULANCE, [21] the authors have aimed at developing a portable device that allows telediagnosis and teleconsultation of mobile healthcare providers by expert physicians. The vital bio signals and images are transmitted from the emergency site to the consultation site using the GSM mobile telephony network. Another sensor network-based emergency service [22] provides a distributed navigation algorithm for emergency situations.

There are products like wireless smoke alarms [11], but they operate on RF and require a separate receiver. In the past, there also have been products like Wi-Fi smoke detectors and fire alarms [12] which are wire-free solutions for the traditional smoke and fire alarms. But these devices have to be first associated with an access point for their operation. For the same reason, they cannot operate as plug-and-play devices.

But almost all such systems either require new hardware support like sensor networks or are not utilizing the power and availability of the 802.11 Wi-Fi protocols. The emergency service should be easily deployable and easy to use. Also, the service should not require any expensive setup and should be an ubiquitous service. The service we are proposing does not require any additional hardware support and requires no association with an access point. The service will be available at all times and can be operated at all places where wireless Internet access is available.

2.2 Guidelines for 911 in the United States

As mentioned, 911 is the emergency telephone number for the United States. Since its deployment in 1968, it has undergone several changes in its operation. New standards such as the E-911 (Enhanced 911), Wireless enhanced 911 and recently 911 for Internet

telephony have been developed and deployed.

Since the emergency number 911 is very critical, the US government has defined certain rules and metrics governing its operation. For instance, the current model for the 911 services in the United States specifies that [23]:

- The numbers should be touch-tone generated
- Incoming calls will be forwarded from the local or end switch to a PSAP (Public Safety Answering Point) responsible for the callers location area
- The callers location should be transmitted automatically to the PSAP using available resources, usually in a packet switched manner.

In addition to the above rules, the FCC also mandates certain rules specifically for the Wireless enhanced 911. Some of these are [24]:

- All 911 calls must be relayed to a call center, regardless of whether the mobile phone user is a customer of the network being used.
- 95% of a network operator's in-service phones must be E911 compliant
- Wireless network operators must provide the latitude and longitude of callers with accuracy of $\pm$300 meters and within 6 minutes

Similarly, VoIP enhanced 911 also mandates that the VoIP providers should make sure that the user of their service should be able to reach the local 911 call center from their VoIP phone. The call should also be free of charge. There have been, though, several issues and complicated technological problems with implementing E911 with VoIP. The providers are still attempting to solve them [25].

In our research, we try to meet all these requirements set for an emergency service. The message which is sent to the PSAP using our emergency service contains "a near" accurate location of the sender. The time to send the message, as we show later in our evaluation, is also acceptable as per the requirements set.

CHAPTER 3

RESEARCH

3.1 Architecture of the Service

In this subsection, we outline the basic architecture of the emergency service.

Motivated by the E911 Service, our architecture consists of an end-user terminal or host system which communicates with a Public Safety Answering Point (PSAP). The PSAP takes responsibility for listening to or reading the emergency message, making a decision based on the type of emergency, location, etc. and sending it along to the appropriate emergency service, such as a police station, fire station, or ambulance service.

The user terminal can be any device having an 802.11 network interface card. The method by which the user activates the system may differ by the type of mobile device. On a laptop, this might mean pressing a predefined key sequence or running a particular command. On a smartphone, it might mean running a special app or pressing a "panic" button. Once activated, the user's device begins sending a message, which is relayed to a PSAP by an IEEE 802.11 compatible access point (AP) or router, as shown in Figure 3.1. This relaying is done by using a novel mechanism that does not require the user device to associate with the AP, and which is described in the remainder of this section.

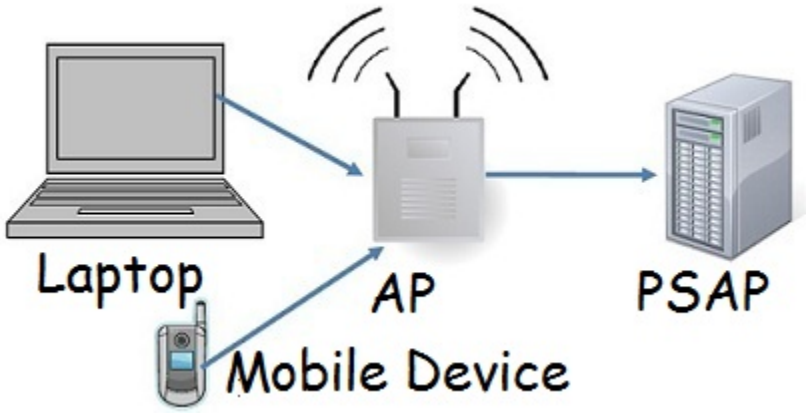

Figure 3.1. Basic architecture of the service

3.2 IEEE 802.11 Framing in Detail

A primary design consideration for our system is that it should be *universally available*. Many APs are configured to give access to a certain set of users by controlling which devices are allowed to associate, requiring a password, or encrypting the network traffic. In order to maximize the coverage of our system and its benefit to public safety, APs must be able to offer this service even to people who would not normally be able to use the AP. Our design should also make minimal changes to the 802.11 protocol, so that it is easy to add to existing AP designs (in many cases without hardware modifications.)

To achieve these goals, the communication between the user's mobile device and the access point makes use of 802.11 management frames. These frames can be sent even by client devices that are not associated with the AP, and they are always unencrypted. We make use of parts of these frames that are unused in the current 802.11 specifications. We give some background on the basics of 802.11 framing, and discuss how our design makes use of its features for emergency message transmission.

In IEEE 802.11, there are three major frame types.

1. *Data Frames* are the most common 802.11 frames, transferring data packets from one station to another.
2. *Control Frames* are used in parallel with the data frames to deliver data reliably from station to station.
3. *Management Frames* perform supervisory functions; they are used to join and leave wireless networks and assist in roaming of a station from one access point to another.

Data frames and control frames are only used after authentication and association with an AP. Therefore, we make use of management frames: this allows APs to offer emergency service to any user, without having to allow those users to authenticate and associate. To explain our modifications to 802.11, we begin with the standard 802.11 frame format.

Figure 3.2 shows the generic 802.11 MAC frame.[1] Each frame starts with a two-byte Frame Control subfield, as shown in Figure 3.3. The components of the Frame Control subfield relevant to our discussion are:

- *Type and subtype fields*: The `type` and `subtype` fields identify the type of frame used. For example, management frames have `type=00`. Probe requests, one particular type of management frame, have `subtype=0100`.

[1] All diagrams in this section follow the IEEE conventions in 802.11. Fields are transmitted from left to right.

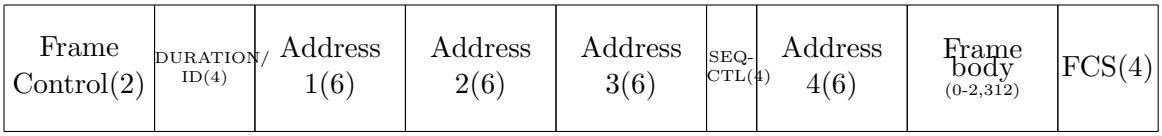

Figure 3.2. Generic 802.11 MAC frame

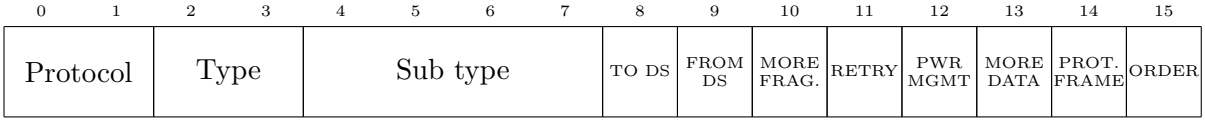

Figure 3.3. Frame control field

- *More fragments bit*: This bit functions much like the "more fragments" bit in IP
- *Protected frame bit*: If the frame is protected by link layer security (LLS) protocols like WEP, this bit is set to 1, and the frame changes slightly
- *Order bit*: Frames and fragments can be transmitted in order at the cost of additional processing by both the transmitter and the receiver. When "strict ordering" delivery is employed, this bit is set to 1.

The *Duration/ID field* follows the frame control field. Under some conditions (when its 15th bit is set to 0), it can be used to represent the Network Allocation Vector (NAV). This value represents the number of microseconds that the medium is expected to remain busy for the transmission currently in progress; as described later, we use it to increase the priority of emergency messages so that they will get through even on heavily-loaded APs.

Management frames are used for Association, Disassociation, Scanning, Authentication, etc. Whenever a wireless station wants to determine which access points are in range, it broadcasts a Probe request. All APs that receive this request respond with a Probe Response frame, containing capability information, supported data rates, etc. to the station. Hence, probe requests and probe response frames are used even when the wireless station is not associated or intending to associate with an access point. Additionally, the frame body for a probe request has very few mandatory elements and hence can be easily modified with very little change to the existing protocol. Hence, for purpose of this research, we use the probe requests and probe response management frames to transmit emergency messages.

Now let us look specifically at these two frames in detail. For both these management frames, all bits of the Frame Control field other than the *Sub type* subfield are always set to 0. Our design tweaks the Frame Control subfield for these two management frames. Specifically, an emergency message of this service always has its *Order bit* set as 1. This

enables us to make the frames of this service distinguishable from the "normal" 802.11 traffic. This is because normal 802.11 management frames always set the *Order bit*, *More fragments bit* and *Protected frame bit* to 0.

3.3 Emergency Message Frames

IEEE 802.11 Management frames are quite flexible. Most of the data contained in the frame body use fixed-length fields called *fixed fields* and variable-length fields called *information elements*. To embed the emergency service into the protocol, we use this flexibility of the management frames. We use the frame body of the management frames to store the emergency message, and set the *Order bit* to 1 to distinguish the emergency frames from other 802.11 traffic.

Sending an emergency message is divided into 3 parts:

1. Selection of an AP
2. Forming and sending of the emergency packet by the mobile device
3. Processing of the received emergency packet by the AP

For the purpose of selecting an AP, a process running on the wireless station scans all the available wireless networks and prioritizes them based on the signal strength values and the number of connections the access point is serving. This is similar to standard AP scanning performed by client devices. Because sending a message to the AP does no good if the AP is not connected to the Internet (such as an ad-hoc wireless network, or one which has experienced a temporary loss of connectivity) we have also added the ability to check whether the chosen AP has Internet. For this purpose, the node sends a modified probe request packet with the *Protected frame bit* and *Order bit* set to 1. When the AP receives this probe request from a mobile device with these 2 bits set, it determines:

1. This is a frame part of the emergency service, so it should be forwarded to the daemon managing this service.
2. The mobile device is requesting lookup of Internet connectivity.

The AP then responds with a modified probe response frame. If it has Internet connectivity (i.e., is able to contact the PSAP), the same two bits are also set in the response. Otherwise, the *Protected frame bit* is set to 0.

The process of selecting an AP is done at startup of the mobile device and after regular intervals. This is done to minimize the time taken to send an emergency message; the scanning process takes a nontrivial amount of time.

After the mobile device has selected an AP, it then can send an emergency message at the appropriate time. To achieve this, the station sends a modified probe request frame to its selected AP. To prepare this frame, the *Order bit* of the Frame Control field is set to 1. The emergency message is put in the frame body of the packet.

The priority of the emergency message is increased by using an approach similar to SpectraLink Voice Priority [26]. To support SVP-type mechanisms in the emergency service, access points and nodes must transmit emergency frames with zero backoff. As mentioned before, the *Duration* field in the Frame control field represents the number of microseconds that the medium is expected to remain busy for the transmission currently in progress and hence, the wireless station backoffs for this time. Hence, if we make this *Duration* field as zero, even in the presence of contention for the wireless medium because of data traffic, the emergency frames with zero backoff will certainly have a priority boost because data frames are likely to have a positive backoff slot. Since the emergency message is a directed probe request, setting the duration as 0 will also enable the frame not to be kept waiting in the queue. But, in scenarios where multiple emergency messages are being sent at the same time, this approach can lead to contention in the network and can lead to the constant collision of the emergency frames. Hence, in case of retransmission of the emergency frame, the 802.11 standard procedure of backing off will be followed. Hence, the emergency frame will be backed off by DIFS[2] amount of time. For IEEE 802.11g standard, the DIFS time is either 28 or 50 μs.

The service also takes care of fragmenting the frame body if its length is more than 2312 bytes (the maximum allowable size). In this case, the *More fragments bit* in the Frame Control is also set to 1, in accordance with the 802.11 protocol.

In order to find the location, the access point uses Skyhook's [27] Wi-Fi Positioning System (WPS). The Skyhook WPS relies on existing WLAN access points for finding the location of devices that have 802.11 wireless interfaces. In WPS, the mobile device collects information about all visible WLAN access points in its vicinity, sends this information to the Skyhook location database which replies with a position estimate based on the aggregated information. The position estimate can then be directly used by a mapping application like Google maps or can be combined with other sources of location information, such as those from GSM stations or GPS. Positioning systems by Mexens [28] and the Fraunhofer institute [29] have a similar mode of operation. The access point also appends

[2]Distributed Interframe Spacing

the MAC address and type of device used by the user to the emergency message. Then it forwards this message to the PSAP.

In our prototype, the communication from the AP to the PSAP is done via email, using a fixed address. (A deployed implementation could use a different mechanism, such as one that explicitly acknowledges receipt at the final destination.) After successfully sending the email, the AP acknowledges this to the mobile device by sending a modified probe response frame with the *Order bit* set to 1. If it fails to send the email, the AP does not send an acknowledgement. A simple flowchart of this process can is shown in Figure 3.4. The wireless mobile device, if it does not receive an acknowledgment, sends the emergency message to all APs in range except the one already tried. In this way, the user can be sure that the emergency message will be served by at least one AP. In order for the PSAP to distinguish between multiple copies of the message that may reach it through different APs, it has to keep track of the mobile devices served. Hence, the message also contains the unique MAC address of the mobile device, so that PSAP can distinguish between unique requests.

To summarize, for the purpose of this service, the following frames are used:

- Probe request frame with *Order bit* set: Used to send the emergency message to the selected access point.
- Probe request frame with both the *Order bit* and *More fragments bit* set: Used to send long emergency messages to the selected access point.
- Probe request frame with both the *Order bit* and *Protected frame bit* set: Used to determine whether the chosen AP has Internet connectivity.
- Probe response frame with the *Order bit* set: Sent by an AP as an acknowledgment of servicing the request successfully. Also sent by the AP in response to the request of checking Internet connectivity if the AP does not have Internet access.
- Probe response frame with both the *Order bit* and *Protected frame bit* set: Sent by an AP in response to a request checking if the AP has Internet connectivity.

3.4 Implementation Details

Our prototype is implemented using a Ubuntu Linux laptop with an 802.11 wireless interface as the mobile device. Another Ubuntu Linux machine is configured to act as a wireless access point (using `hostap` version 0.7.2).

To initiate the emergency message, we use a predefined key sequence (`ctrl + E`) on the wireless device. The process can also be started by entering the command at the terminal.

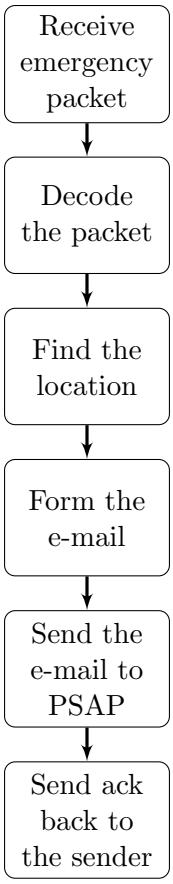

Figure 3.4. Role of the AP

The service can also be customized to the requirements of the user. For example, on a mobile phone, the emergency message could be sent by pressing a dedicated "panic" button.

The first step of this service is to select an access point. The two parameters used to quantify the value of an access point are received signal strength (RSS) and load on the AP. To determine the RSSI value (signal strength), our prototypes uses the `iwlist` command on Linux. To find the number of connections to an AP, the `tcpdump` tool is used to sniff the network of all the packets and then group the capture into groups based on the destination address.

As mentioned, the wireless station, after selecting the AP based on these two parameters, can also check whether the selected AP has Internet connectivity or not. In order to do so, it sends a modified probe request frame to the AP. The AP upon receiving this frame initiates a different thread which does a *ping* to the PSAP Mail server, in our case, *www.gmail.com*, 3 times. If it has Internet connectivity, the AP notifies the wireless station by sending the appropriate packet. A deployed implementation could more explicitly check

for connectivity to the PSAP by contacting the PSAP directly with an application-level ping. Another alternative implementation can be to provide this information in the *beacon* frames broadcasted by the access point. We were not able to implement this due to the inability to change the behavior of the Wireless card while in *Managed* mode and hence, we were unable to parse the beacon frame and find whether the `has_Internet` flag is set or not.

The following flowchart (Figure 3.5) illustrates the AP selection process.

To send any frame for this service, the wireless node injects 802.11 compatible packets, specifically the modified probe request packets. To achieve this, we put the wireless card in Monitor Mode and then inject packets using *pcap* library by manually forming the required

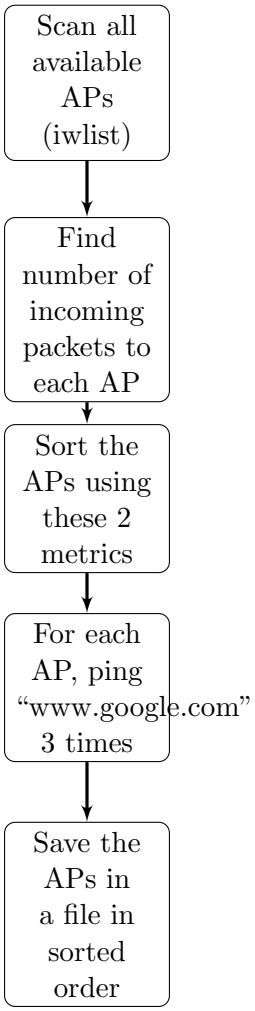

Figure 3.5. Selecting an AP

packets and use raw sockets[3] to send them via the wireless card. In our prototype, the emergency message is stored in a text file on the mobile device. The text file can be modified appropriately to accommodate the message the user wants to send to the PSAP. Further, the user has the option to attach an image with the emergency message. The emergency message, along with any attachments, forms the frame body of the packet which is injected. The entire process of sending the emergency packet can be illustrated using the following flowchart (Figure 3.6).

[3]Raw sockets allow direct sending and receiving of network packets, bypassing all encapsulation in the networking stack of the operating system.

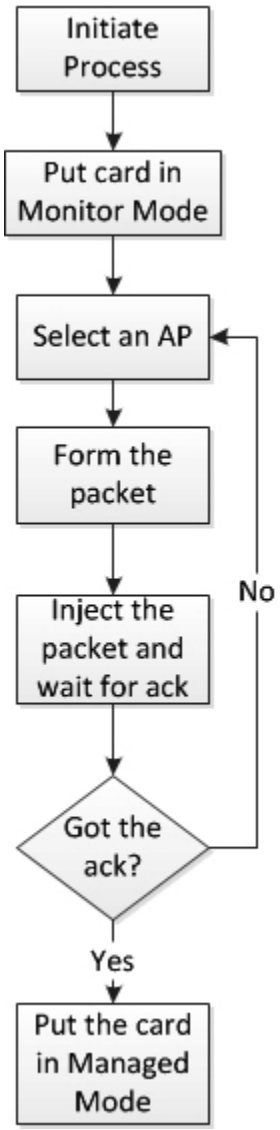

Figure 3.6. Sending the message to an AP

On the machine set up as an AP, the `hostapd` daemon process sends the frames which belong to the emergency service (probe requests having the *Order bit* set) to the emergency service daemon process running on the same machine. This is achieved using POSIX message passing APIs. The daemon process then extracts the emergency message, finds the location, relays the message to the PSAP and then sends an acknowledgment back to the host. The location, as mentioned, is found using the Skyhook WPS API. The response of the API is then formatted to this form:
Approximate Location:
latitude: 40.768348, longitude: -111.845170
Accuracy+/-171m 1
speed: 0.0km/h bearing: 0
Approximate Street Address:
12 Central Campus Dr
Salt Lake City, UT 84112
The PSAP on receiving this message takes the necessary steps of forwarding it to the appropriate destination like the police, fire department, medical hospital, etc.

The access point always creates a new thread to service an emergency request since this process should not interfere with its role as an access point and also it should be capable of servicing multiple emergency requests at the same time. To summarize, the service is comprised of the following components:

- A program to select an access point based on signal quality and strength
- A program which puts the wireless card in Monitor mode, calls the daemon process which makes sure the chosen AP has Internet connectivity.
- A program which puts the wireless card in Monitor mode, calls the daemon process which prepares the emergency message and sends to the AP.
- `hostapd` running on a Ubuntu Linux machine simulating an AP and forwarding emergency request packets to the daemon process.
- A daemon process running on both the mobile device and the AP which actually creates the message and receives the acknowledgments.

3.5 Implementation on USRP

In order to test a hardware implementation of our service, we used a USRP. This has been discussed in depth in Chapter 5.

The Universal Software Radio Peripheral, or USRP, is designed to allow general purpose computers to function as high bandwidth software radios. In essence, it serves as a digital baseband and IF section of a radio communication system. The USRP can be used for reproducing and analyzing some widespread wireless protocols, such as IEEE 802.11. Open source Software Defined Radios like the USRP can help researchers to avoid the closed source firmwares/drivers of commercial chipsets, and provides fully customizable physical and datalink layers.

We implemented our emergency service on two USRPs. Specifically, we use the second generation of USRPs, the USRP2 which has a Gigabit Ethernet interface to connect to a host machine. We use the XCVR2450, Dual-band Transceiver daughterboard which has a 100+mW output at 2.4-2.5 GHz and 50+mW output 4.9-5.85 GHz. We use this specific daughterboard since it was the only one capable of operating in the frequency range of the 802.11 Wi-Fi protocols.

In our setup, one USRP2 acts as a transmitter (node) and the other as a receiver (AP). There is a daemon process running for both the transmitter and the receiver on the respective PCs. These processes form the emergency packet at the transmitter and receive the packet on the receiver. Hence, as with our other implementation on PCs, the daemon process on the host's side forms an emergency packet by inserting a fixed predefined string at the beginning of the payload of the packet. Then, the actual emergency message is appended to the payload, with a fixed trailer text. The packet is then transmitted to the access point, represented by the other USRP2. The second USRP2, upon receiving any packet, first checks for the specific string in the payload of the packet. If it matches with the predefined string for emergency service, it then extracts the emergency message out of the packet and spawns a thread which sends an email to a PSAP with the extracted emergency text. Then, it sends an acknowledgement back to the sender which is similar to the emergency message frame but with a different payload. The protocol used between the two USRPs is IEEE 802.11b. The code for running IEEE 802.11b on the USRP was originally developed by BBN technologies [15] for the USRP. We modified this implementation work on the USRP2, and added a layer for transmitting and receiving emergency frames which adhere to the IEEE 802.11b protocol.

3.6 Current Limitations

Our implementation has a few limitations. The current implementation is not compatible with the chipsets which do not support packet injection. Many older wireless NIC cards

available on laptops and desktops do not have this capability. Also, the wireless chips present in smartphones also do not allow packet injection. As a result, our current implementation could not be tested on such devices. To enable such an emergency service in these devices, the vendor has to incorporate the changes in the wireless driver or the 802.11 subsystem running on the devices. Our design, however, is specifically intended to make this easy, since it requires very few modifications in the current implementation of the protocol. Also, sometimes the wireless adapter is unable to transmit in monitor mode and is restricted to a single wireless channel, though this is dependent on the wireless adapter's driver, its firmware, and its chipset's features.

Our implementation will not run on any OS which does not have necessary APIs and extensions for wireless monitor mode, for example, older versions of Windows (prior to Windows Vista). For such operating systems, it will be necessary to develop or modify the drivers of the wireless network interface to enable monitor mode or use an USB Wi-Fi adaptor which supports monitor mode.

Another limitation of the prototype is the inability to determine whether the PSAP has actually received the emergency email from the access point. The access point currently acknowledges the sender as soon as it is successful in sending the email across to the PSAP. This can be a problem since the sender will now not send another emergency message even though the PSAP has not received the email. This can be resolved by using an alternate protocol to contact the PSAP which explicitly acknowledges receipt of the message.

Our implementation on USRP2 using 802.11b protocol suffers from the limitation that software radios are not always capable of correctly reproducing operations previously done in the hardware domain. In our case, this is mainly because the current implementation of 802.11b receiver for the USRP is only capable of decoding low rate 802.11 packets from standard NICs over the air reliably at 1Mbps and partially at 2Mbps; it does not process packets at higher data rates. However, the USRP2 implementation is merely used as a proof of concept to show that our scheme can be implemented in hardware; it is unlikely that USRP-based access points will be deployed in production.

3.7 Evaluation of the Service

We evaluate and test our emergency service for correctness and performance. We test correctness by examining the e-mail the PSAP receives. This email should have the correct information about the sender and its approximate location. For evaluating performance, we run several experiments that measure the time taken until the PSAP receives the emergency

e-mail from the selected access point under different loads on the access points. We also evaluate how access point prioritization and use of acknowledgment helps in dealing with heavily loaded access points.

We use three laptops as mobile devices and two PCs running Ubuntu Linux 9.04 as access points using the *hostap* program. The three laptops have wireless cards from different manufacturers, namely Intel 5100, Atheros AR5413 and Intel 3945. All the laptops run Ubuntu Linux 9.04. The APs have the Atheros AR5413 chipset. One of the two access points is WPA2 protected with a random password.

We write the daemon processes, as we explain in Section 2, in C. We write the scripts, also explained in Section 2, in Python and using Linux shell commands. We write the USRP daemon processes in C++ and the front-end scripts to run the application in Python.

To send an emergency message, we use a specific key sequence `ctrl + E` on one of the mobile devices. We then measure the time duration since pressing the `ctrl + E` key sequence until the emergency e-mail is received by the PSAP. We summarize the time duration measurement under different scenarios in Table 3.1.

We also verify every time the emergency service is used that the access point selected to send the emergency message has indeed the highest signal strength.

Table 3.1. Performance testing of various cases

S.No.	Scenario	Time taken	Remarks
1	Sending the message to a preselected AP	1.2 s	AP is not under heavy load
2	Sending the message to a preselected AP	1.8 s	AP is under heavy load, increasing priority of the emergency message
3	Sending the message to a preselected AP	2.4 s	AP is under heavy load, not increasing priority of the emergency message
4	Selecting an AP and then sending the message to an AP	3.8 s	Scanning of APs take time
5	Selecting an AP and then sending the message to an AP	4.3 s	AP is under heavy load, increasing priority of the emergency message
6	Selecting an AP and then sending the message to an AP	5.1 s	AP is under heavy load, not increasing priority of the emergency message
7	Sending the message to a preselected AP	1.9 s	AP is not under heavy load, testing the AP for Internet connectivity (”ping” message)

We also perform an experiment to measure the percentage of emergency packets serviced by the access point. To achieve this, we send packets at almost the same time from different mobile devices and then count the number of received e-mails at the PSAP. For this purpose, we create a script that sends emergency packets continuously. We then start the script at the same time on all the three mobile devices in our experimental set up. Also to measure the throughput of the service, we temporarily allow rapid sending of emergency packets by the same mobile device. The results of the case when the access point does not send the acknowledgement back to the mobile device are shown in Table 3.2.

The results for the case where we enable the acknowledgements, i.e., when a mobile device retransmits the emergency message in case it does not receive an acknowledgment in 2 seconds, are shown in Table 3.3.

Clearly, the use of acknowledgments improves performance while making the emergency service more reliable.

Table 3.2. Results obtained when not sending acknowledgements

	No. of Packets sent from	No. of e-mails received by PSAP from
1	Mobile device 1: 2 Mobile device 2: 1 Mobile device 3: 2	Mobile device 1: 2 Mobile device 2: 0 Mobile device 3: 2
2	Mobile device 1: 30 Mobile device 2: 2	Mobile device 1: 24 Mobile device 2: 0
3	Mobile device 1: 30 Mobile device 2: 30 Mobile device 3: 30	Mobile device 1: 26 Mobile device 2: 22 Mobile device 3: 19
4	Mobile device 1: 100	Mobile device 1: 81

Table 3.3. Performance improvement by sending acknowledgements

	No. of Packets sent from	No. of e-mails received by PSAP from
1	Mobile device 1: 2 Mobile device 2: 1 Mobile device 3: 2	Mobile device 1: 2 Mobile device 2: 1 Mobile device 3: 2
2	Mobile device 1: 30 Mobile device 2: 2	Mobile device 1: 30 Mobile device 2: 2
3	Mobile device 1: 30 Mobile device 2: 30 Mobile device 3: 30	Mobile device 1: 30 Mobile device 2: 29 Mobile device 3: 29
4	Mobile device 1: 100	Mobile device 1: 97

We also perform experiments to show the benefits of increasing the priority of the emergency messages. To implement this, a laptop is continuously sending traffic (802.11 probe requests) to a chosen access point. The other laptop acts as the sender of the emergency requests. Results obtained are shown in Figure 3.7. The graph depicts the time it takes to receive "x" emergency messages.

Clearly, an increase in the priority of the emergency messages reduces the time required for the message to be serviced.

After running these experiments, we conclude that the acknowledgment of mobile devices' emergency messages by the access point and the prioritizing of emergency packets on the wireless network improve the overall efficiency and speed of the service for practical wireless setups.

We also conduct experiments to measure the time an emergency message takes to be received by the wireless access point after being sent by the mobile device. We use the `gettimeofday` API call to measure the time difference. The times at both the sender and the receiver are synchronised. The results are tabulated in Table 3.4. The results again show that increasing the priority of the emergency message does help in reducing the time it takes in receiving the message at the access point, and ultimately will reduce the overall time to be received by the PSAP.

We also conduct experiments comparing our emergency service with the Short Message Service (SMS) provided by the mobile operators. Specifically, we use the Google Voice [30] to send SMSs to an AT&T mobile phone. The Google voice uses the Internet to send

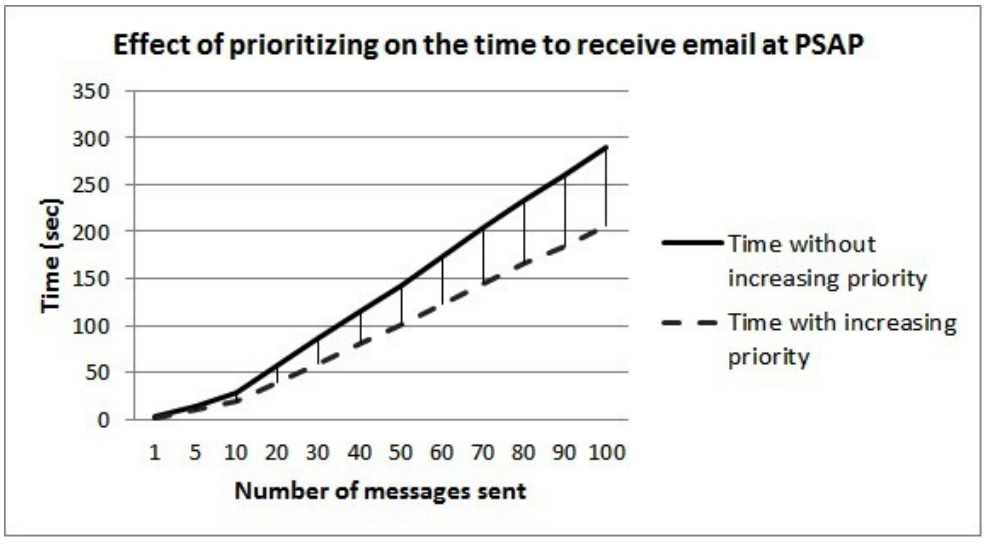

Figure 3.7. Effect of prioritizing emergency messages on time to receive e-mail at PSAP

Table 3.4. Time taken to send an emergency message to AP

S.No.	Scenario	Difference in Timestamps
1	Sending the message to a preselected AP (without n/w traffic)	67.910(ms)
2	Sending the message to a preselected AP (with n/w traffic,) (increasing the priority)	452.613(ms)
3	Sending the message to a preselected AP (with n/w traffic,) (not increasing the priority)	998.184(ms)

the short message across to the destination phone number. In our results, the average time an SMS took (we sent a total of 15 SMSs) to be delivered is around 9.8 sec. Our implementation, as seen by the results in Table 3.1, took 4.3 sec on an average for the case when we, before sending the message, scan the network for selecting the access point and also the network is heavily loaded with traffic.

We also conduct experiments by running the emergency service on USRPs. Recall from Section 2 that we want to run our service on hardware for which we have full control of the physical and the MAC layer. This control allows us to set the channel of the transmitter and receiver and various other parameters like sampling rate and the use of barker spreading[4].

In this setup, we use 2 USRP2s, with one USRP2 being the transmitter and the other being the receiver. The transmitter sends the emergency message by forming an IEEE 802.11b compatible packet with a modified payload. Like the experiment for the original version, the metric we measure is the time taken by an emergency message to reach the receiver from the transmitter. Again, we use the `gettimeofday` API call to measure the time difference.

We also simulate a busy network scenario using 4 different USRP2s. In this scenario, two USRP2s, both tuned to the same frequency channel, are constantly transmitting 802.11b frames to generate traffic. The remaining two USRP2s act as the sender and receiver of emergency messages. The combined results are shown in Table 3.5. The results clearly show that interference causes a big delay in receiving the packet on a USRP2 using the BBN 802.11b code. This is also due to the fact that no MAC layer is running on the USRP2 for the IEEE 802.11 subsystem.

In the setup with USRP2s, another performance metric is Round Trip Time (RTT). The average length of the emergency packet is $\approx$ 100 bytes. We also run the test with

[4]an autocorrelation sequence used for the 1 and 2 Mbit/sec rates in IEEE 802.11b

Table 3.5. Time taken to send an emergency message to the receiver USRP2

S.No.	Scenario	Difference in Timestamps
1	Sending the message to a preselected AP (without n/w traffic)	734.169(ms)
2	Sending the message to a preselected AP (with n/w traffic)	1521.613(ms)

a maximum packet length of 500 bytes. We measure RTT as the time that elapses from the start of sending of an emergency frame from one USRP2 to another to the complete reception of an acknowledgment frame, which is almost of the same size. The RTT results are shown by a histogram in Figure 3.8. The mean RTT of 736 emergency frames is ≈ 3.13 ms per bit. Hence, for an average emergency packet, the RTT is ≈ 2.5 seconds.

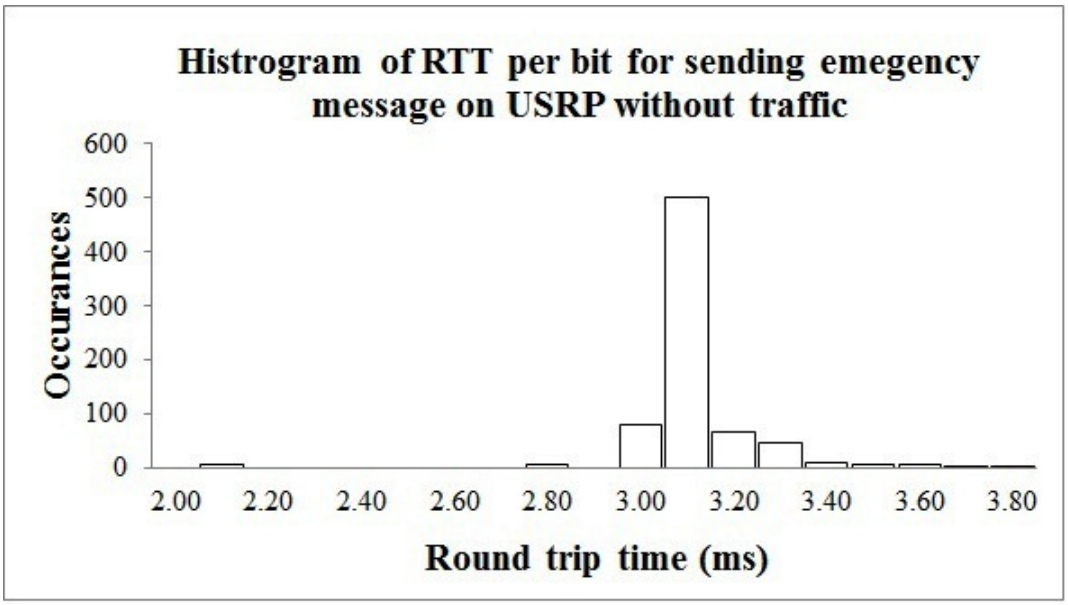

Figure 3.8. Histogram of RTT per bit for sending emergency message on USRP without interfering traffic

CHAPTER 4

SOFTWARE DEFINED RADIO

4.1 Introduction

The term "Software Radio" was coined by a team of researchers at Garland Texas Division of E-Systems Inc. (now Raytheon) in the year 1984. A software defined radio (SDR) is a radio communication system in which the digital components which generally are implemented in hardware (for e.g., filters, modulator, etc.) are implemented using software running on a personal computer or an embedded device. SDRs help in implementing and integrate the software domain as close as possible to the antenna (hardware RF) in order to guarantee a higher flexibility and reconfiguration. In addition to that, the main aim of SDR is to turn hardware problems into software problems [31]. A system built using SDR will be able to use different protocols and techniques with the press of a button! SDR technology can be used to implement a variety of radio applications. A wide range of radio applications like Bluetooth, GPS, Radar, WCDMA, GPRS, etc. can and have been implemented using SDR technology.

Currently, there are several SDR solutions available in the market. For this work, we opted for an open-source SDR platform that combines GNURadio (as software-defined subsystem) and the Universal Software Radio Peripheral, USRP2, as hardware-defined subsystem.

4.2 Universal Software Radio Peripheral

The Universal Software Radio Peripheral (USRP) was designed within the GNURadio project [32] and is currently manufactured by Ettus Research [33]. In this thesis, we use the USRP version 2 (we will refer it as USRP2). This version is a successor to the USRP1 and was primarily to overcome the low bandwidth and processing power of the USRP1. In the following sections we briefly describe the main components present in the USRP2. We have taken details from the USRP2 datasheet present in [34].

4.2.1 Motherboard

The main features of the motherboard are listed below:

- FPGA[1], which contains a 32-bit RISC microprocessor
- Two 100 MS/s 14-bit analog to digital converters
- Two 400 MS/s 16-bit digital to analog converters
- Gigabit Ethernet Interface

4.2.2 Daughterboards

The name `daughterboard` signifies that they are usually meant to be an extension or `daughter` of the motherboard. A daughterboard usually has plugs, pins or sockets which differentiates it from a standard expansion chip such as PCI[2]. In the case for the USRP, the daughterboards consists of an RF filter to discard the out band components, a LNA (Low Noise Amplifier), a mixer to move the signal from the Radio frequency (RF) to Intermediate Frequency (IF), an IF filter, and an IF amplifier[1]. The daughter board together with the motherboard let us use the USRP2 as a complete RF transceiver system. A wide variety of available daughterboards are available which allows using different frequencies for a broad range of applications. We have chosen the XCVR-2450 daughterboard, which is a transceiver board operating at 2.4-2.5 GHz and 4.9-5.9 GHz. Therefore, it is suitable for the IEEE 802.11 family of standards. A detailed datasheet and specifications of the different daughterboards available for the USRP2 is available at [35].

4.3 GNURadio

GNURadio is a free software platform and SDK licensed under the GPL for implementing software defined radios. The GNURadio project was started in 2001 by Eric Blossom. GNURadio is a signal processing package and aims to bring the software world as close to the Radio Frequency world as possible. It allows developers to basically hack the radio and electromagnetic spectrum. It is available for many flavors of Linux. Even precompiled binaries for Win32 are now available.

GNURadio applications are primarily written using the Python programming language. The supplied libraries, though, are written in C++ since they perform critical signal processing blocks and require floating point extensions.

---

[1] Field-programmable gate array

[2] Peripheral Component Interconnect

In GNURadio, the basic building blocks for any application are flow graphs and blocks. Many GNURadio applications contain only flow graphs, with the nodes of such a graph called blocks, and the data flowing along the edges. Any actual signal processing is done by these blocks. Blocks are usually written in C++, but they can be also written in Python. The data flowing through these blocks can be of any kind. The most common data types are complex and real short or long integers and floating point values. Figure 4.1 depicts a simple GNURadio flow graph. Any graph must have at least a signal source block (in the example the "Signal Generator" and the "Noise generator") and a signal sink block (in the example the "USRP2 Sink"). Additionally, the flow graph usually contains intermediate signal processing blocks (in the example the "Adder" block).

The GNURadio package provides several digital processing blocks for signal and information processing, as well as the framework to control the data flow between them. It also provides support for various signal sources and sinks. More details are present in [36].

4.3.1 GNURadio Architecture

Figure 4.2 shows the general GNURadio framework structure. In most cases, Python is used to create high level graphs. The signal processing blocks that are included natively

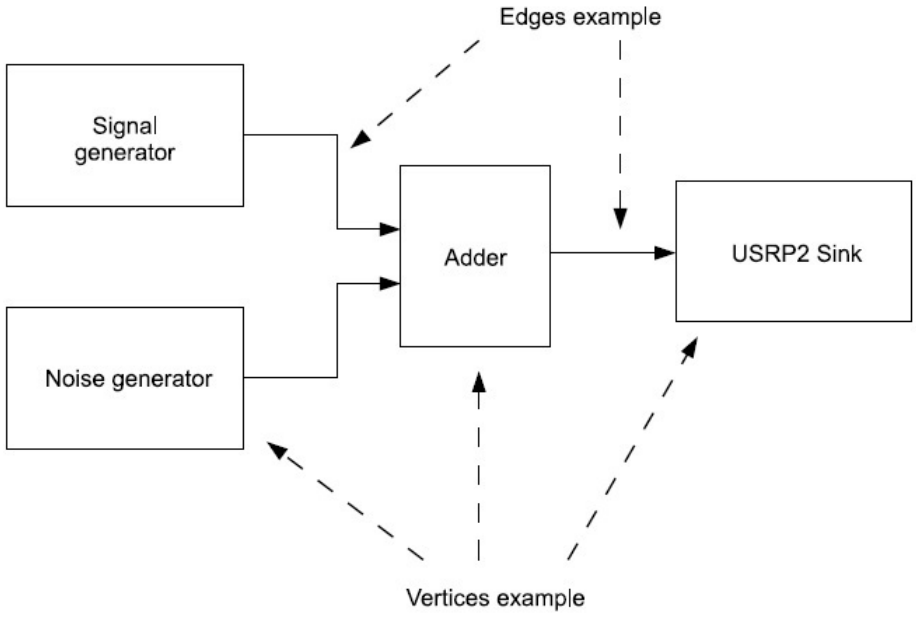

Figure 4.1. Typical GNURadio graph [1]

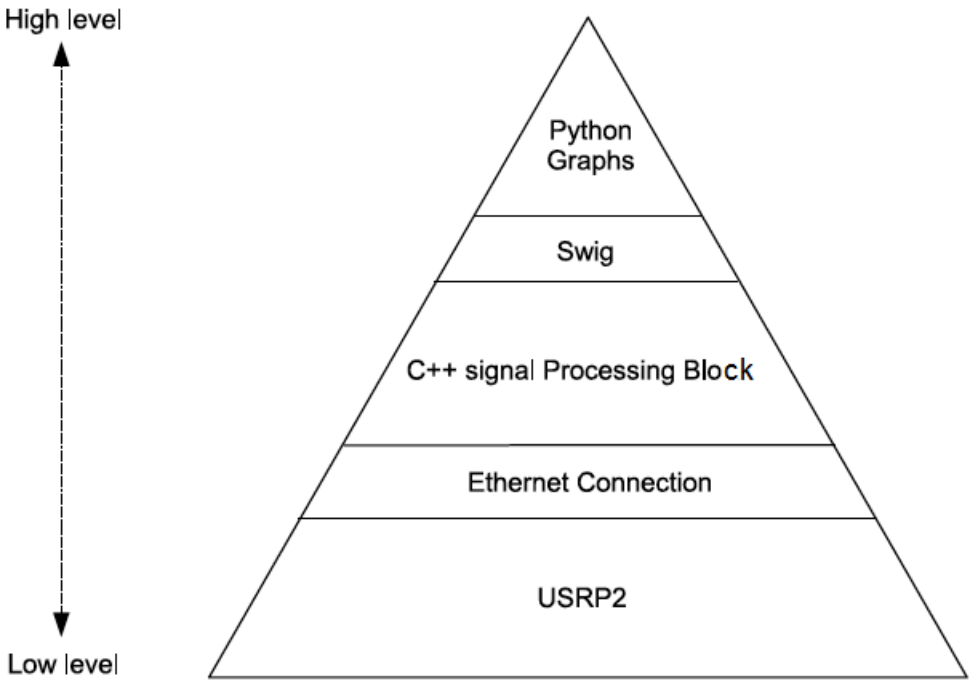

Figure 4.2. GNURadio framework[1]

are programmed in C++. To connect a Python script to a C++ library, SWIG[3][37] is used. Mostly this is done to make C++ functions available to the Python program. This approach allows the developer to use the efficiency of C++ coding along with the simplicity of Python.

Again, referring to Figure 4.1, the blocks, such as the signal generator, the noise generator, the adder, and the USRP sink are written in C++ and they represent the body of the graph. The edges, which are the connection between the blocks, are written in Python and they represent a high level view of the graph. In other words, all the signal processing, critical and real-time operations are written in a lower level language (C++), while the interfaces among them are written in a higher level language (Python).

4.4 GNURadio Signal Processing Block

To develop an existing or a new protocol on the USRP2 using the GNURadio SDK, the developer needs to write his own signal processing block. As mentioned before, all the critical and real time operations are done in C++, with the interfacing between different blocks done in Python. Hence, from a developer's point of view, the GNURadio provides a

[3]Simplified Wrapper and Interface Generator

layer of abstraction to the python interface, when we are using GNURadio's existing blocks. We consulted the online tutorial available at [38] to write our own signal processing block. The new block needs to be implemented as a shared library so that it may be imported into Python using SWIG that acts as an interface between C++ and Python. Hence, writing a new block will require at least 3 files: the `.h` and `.cpp` files that define our new block and the `.i` file which defines the SWIG interface and generates the Python code which acts as an interface to our block. All signal processing blocks should be derived from the `gr_block` class or any of its subclasses. GNURadio has defined many virtual functions, the main being the `general_work` which needs to be implemented by the new block we write. This method is the core of our block. Again, more details can be found at [38, 39].

4.5 Developing the Complete GNURadio Flow Graph

After the development of the signal processing blocks, we need to connect them to make a working application. This is done using GNURadio flow graphs which are written in Python. As mentioned before, writing a flow graph in Python does not require inner details of the signal processing block to be used. The APIs exposed by the signal processing block are available to the Python flow graph using the SWIG interface.

The first step is to `import` the signal processing block into the python script. Next, to define our own graph, we need to define our own class (usually named as `my_top_block`) which is essentially derived from a GNURadio class `gr.top_block`. This class is basically the container for our flow graph. By deriving from `gr.top_block`, we shall get all the hooks and functions needed to add the blocks and connect them. The blocks are simply connected using the `connect` method. More details and a detailed tutorial, which we also referred, are available at [40].

4.6 Existing IEEE 802.11 SDR Implementations

There have been several attempts to develop and implement the IEEE 802.11 standard with GNURadio and USRP. Most of these implementations are designed on the USRP1. The USRP1, as mentioned before, uses an USB connection instead of the Gigabit Ethernet to interface with a PC and hence, the data transfer rate is limited to the USB bandwidth of 32MB/s. The IEEE 802.11b signal spectrum is 11 MHz; therefore, the minimum sampling rate would be 22 Msamples/s. By using 16 bits samples (8 bits for each I and Q sample) we would require a bandwidth of 2*22 Ms = 44 MB/s, which is above the USB limit [1]. There have been two successful implementations that try to avoid this bottleneck.

The first well-known implementation has been developed by BBN technologies, within the ADROIT project funded by the DARPAs ACERT program[41]. In this implementation, the signal bandwidth is reduced to 4 MHz before being sent to the USB controller. In this way, the signal is basically subsampled. But due to this, the Signal to Noise Ratio (SNR) degrades. But still, it is considered a very important development in making IEEE 802.11 work on the USRP.

Hamed Firooz[42] proposed another implementation. In his architecture, despreading operation is performed in the FPGA [4] present in the USRP prior to the USB, rather than in the host CPU. In this way, the data and hence the bandwidth to be transferred through the USB connection is considerably reduced. To illustrate this, the IEEE 802.11 uses Direct-Sequence Spread Spectrum (DS-SS), with a code sequence of 11 chips per symbol. With this implementation, the signal is despreaded before the USB connection, and hence the symbol rate is reduced to 1Msymbol per second, which is easily supported by the USRP. When the signal arrives at the PC, the IEEE 802.11 demodulator and Physical Layer Convergence Protocol (PLCP) from the ADROIT BBN project is used for decoding the frame. Details on the implementation and the Verilog code for the Altera FPGA are freely available in [42].

We, on the other hand, develop the transmitter and the receiver on the USRP2 which has a Gigabit Ethernet interface with the PC and hence does not suffer from any bandwidth limitations as the case is with the USRP1. We did take reference of the BBN code for the IEEE 802.11b and also modified it to be able to run on the USRP2. It performs like expected, though as mentioned, it only works for the 1 and 2Mbps rates for IEEE 802.11b.

[4] Field-programmable gate array

CHAPTER 5

IEEE 802.11g IMPLEMENTATION ON USRP2

Our main goal to implement an IEEE 802.11g transmitter and receiver is to test our Emergency Service on the USRP2 and GNURadio platform, which is an open source and cost-effective hardware alternative to an access point. As a matter of fact, we faced some difficulties in the beginning when we were trying to implement the service on conventional wireless cards present in laptops and desktops. We then started the project to develop the service on the USRP. As a part of this project, we also decided to implement a transmitter and the corresponding receiver for the IEEE 802.11g protocol. This will enable us to modify the PHY and the MAC layers of the protocol to support the emergency service in the most efficient way. Since we are unable to develop the receiver completely, the aim to fully implement the emergency service on the USRP has not been achieved. But, due to the current lack of any open-source 802.11g chip sets available in the market, our implementation can help the research community to conduct experiments to measure different metrics and performance parameters of the 802.11g standard.

Implementing a wireless technology with GNURadio is a complex process. The reason we chose to implement it and not use the existing implementations as mentioned in Section 4.6 is that the earlier implementations are not capable of transmitting at the full 11Mbps for IEEE 802.11b, leave alone the 54Mbps for 802.11a/g protocols. This is primarily due to the fact that those implementations were developed for the USRP1 which has a USB bottleneck of 32Mbps.

Typically, the main steps involved in the development of the IEEE 802.11 transmitter and receiver are:

- Understanding the physical layer of the standard
- Creating the required signal processing blocks or using the ones provided by the GNURadio package
- Creating the flow graph to use the blocks in an application

5.1 Approach

To implement a IEEE 802.11g transmitter and receiver for USRP2, we needed to implement several signal processing blocks present in the IEEE standard. Some of them were present in the existing GNURadio repository. Figure 5.1 shows the available GNURadio OFDM chain.

As we can see, the frame generation consists only of a symbol mapping, IFFT[1], cyclic prefix insertion and training sequence insertion. The blocks implemented in the GNURadio are fundamental in nature and are far away from being IEEE standard compliant.

The full IEEE 802.11g transmitter/receiver block diagram is shown as in Figure 5.2. In order to implement it on USRP and GNURadio, we need to implement the blocks contained in it. We also took reference from an earlier research work done by Andrea Constantini [43]. The corresponding GNURadio chain which we implement can be seen in Figure 5.3. We implemented some of the blocks in Python, instead of C++, since its easier to write code in Python than in C++ due to many data structures and their corresponding operation available by default in Python. In particular, we used Python for all the blocks that perform bit-operations (e.g., bit-shifts, bit-masks, etc.). The rest of the chain, from symbols mapping to the last block before the transmission, has been done entirely in C++, due to tight real-time constraints. In the following subsections, we describe the GNURadio implementation of each block we implemented.

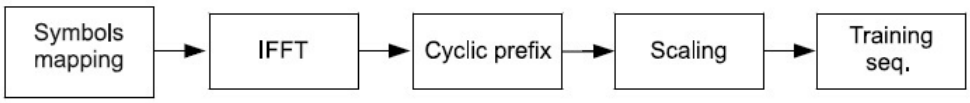

Figure 5.1. OFDM blocks present in GNURadio core

[1]Inverse Fast-Fourier Transform

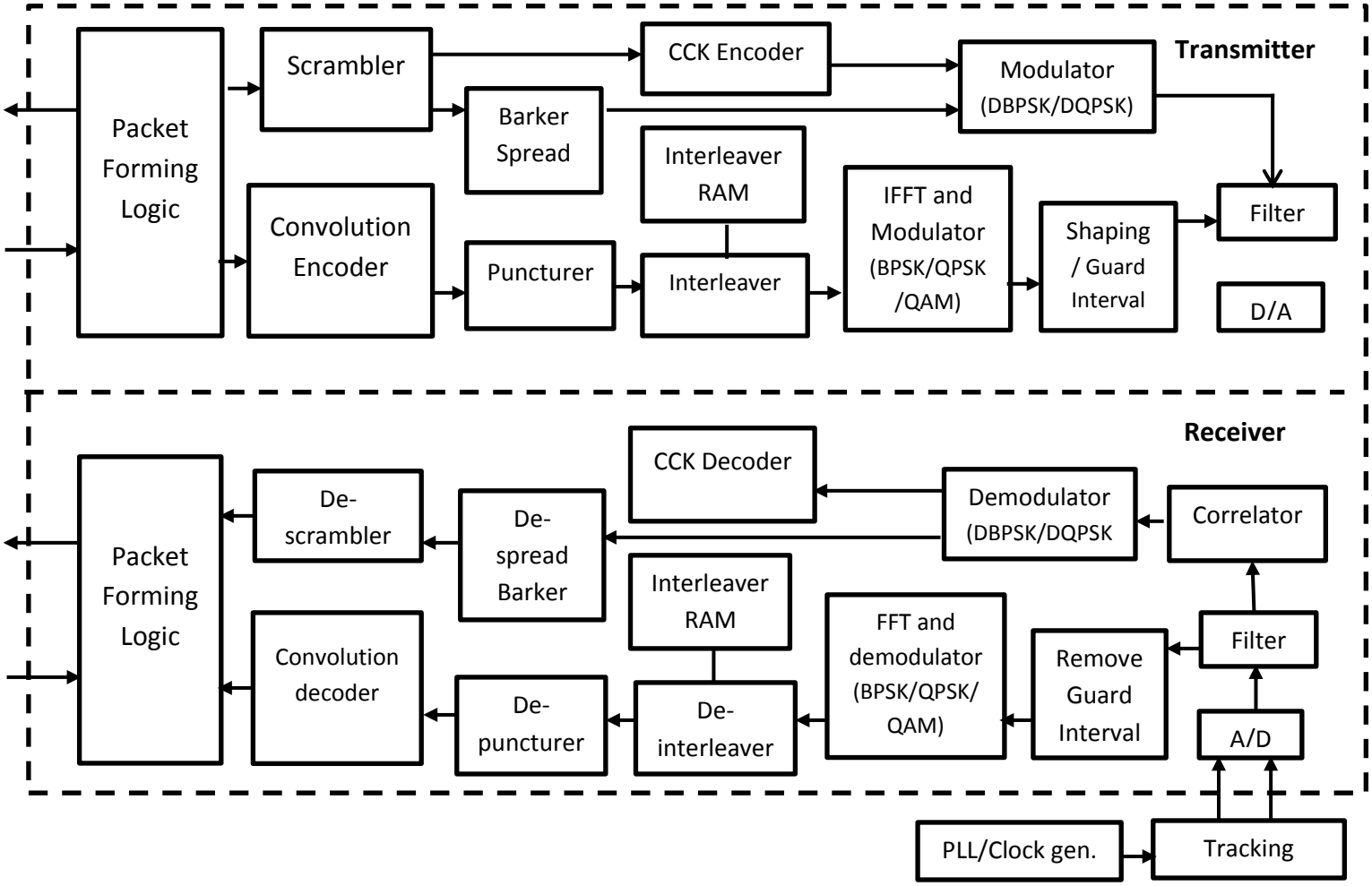

Figure 5.2. IEEE 802.11g near-commercial implementation

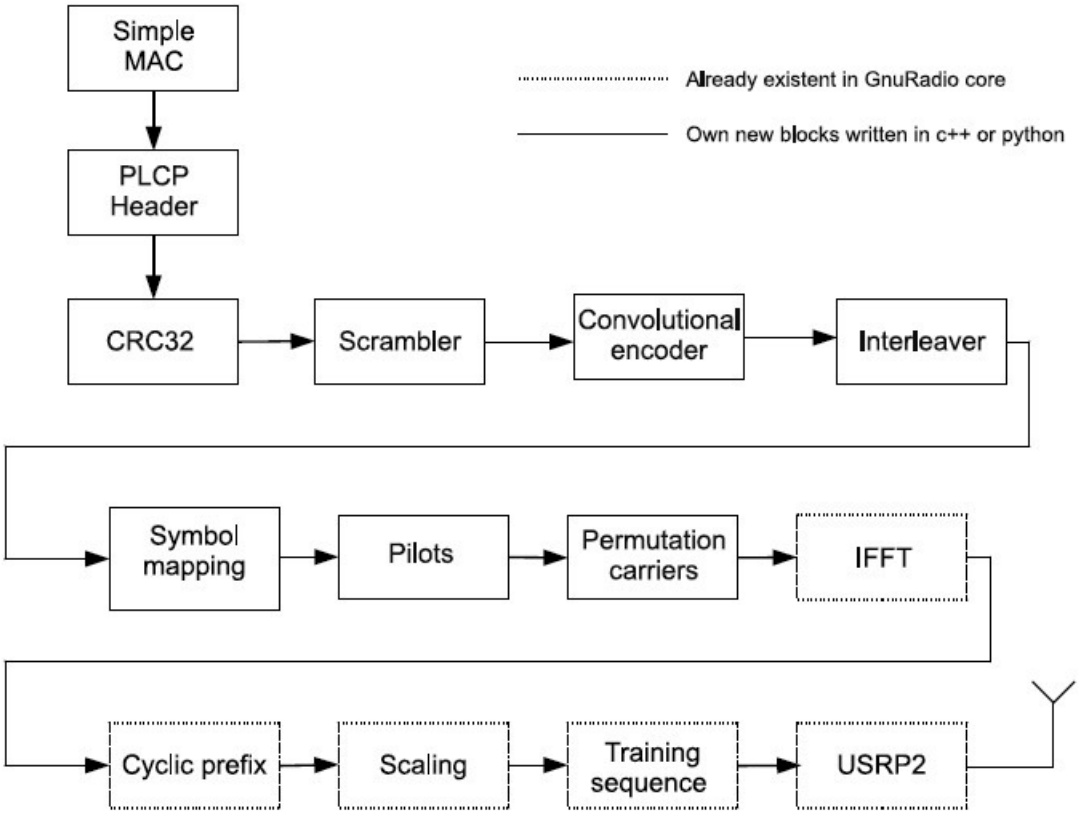

Figure 5.3. IEEE 802.11g chain implemented in GNURadio

5.2 Description of DSP Blocks

5.2.1 CRC-32

The cyclic redundancy check (CRC) is used by the receiver of the frame to check if there is an error in the received sequence of bits contained in the frame. In our system, the CRC-32 is calculated according to the algorithm in [44]. The code available in the GNURadio repository did not produce a standard compliant CRC and caused the receiver (we use Wireshark [45]) to reject the frame. We write the IEEE 802.11 compliant version of the CRC-32 code in C++, with a wrapper in Python.

5.2.2 Scrambler and Descrambler

The data are scrambled before they are modulated and transmitted. This is done to remove long sequences of 1s or 0s in the frame. The first 6 bits of the Service bits in the Signal field in the OFDM PLCP frame is used to initiate the scrambler. The scrambling is performed in Python. To implement the scrambler, the DATA field, composed of SERVICE, PSDU, tail and padding bits is XORed with the scrambler sequence, stored as a sequence

of bits in the Python script. The SIGNAL field is instead not scrambled. Also, unlike the 802.11g standard, in which the seed is randomized each time a frame is transmitted, we use the same stored seed for all transmissions. The polynomial $G(z) = z^7 + z^4 + 1$ is used to scramble all bits transmitted by the PHY layer. The corresponding block diagram in accordance with the IEEE 802.11g standard is shown in Figure 5.4.

The descrambler is easily implemented by calling the scrambler function on the descrambled input as the configuration of the scrambler and descrambler is self-synchronizing.

5.2.3 MAC and PLCP Header

The first step in the transmitter chain is the encapsulation of the payload into the MAC and PLCP headers. In Figure 5.5, the format of the PPDU, including the OFDM PLCP preamble, is shown. The preamble in the PLCP consists of a synchronization field followed by a start of frame delimiter. It may be either the long preamble of 144 bits or the short preamble of 72 bits. In either case, the preamble is transmitted at 1 Mbps using DBPSK modulation. Before modulation, the data are scrambled. We implemented this module such that the PLCP preamble is added from a static table. We programmed this module using the Python programming language. More details about each single field can be found in [46]. The signal field is properly set since it is used by the receiver to properly demodulate the frame using the appropriate demodulation scheme. The value of this field is, as of now, determined by a command line argument to both the transmitter and receiver. The command line argument determines the rate of transmission and in turn the modulation scheme to be used.

The receive PLCP is invoked upon detecting a portion of the preamble sync pattern followed by a valid SFD and PLCP header. The PLCP header also contains the SERVICE filed. It contains the control bits which help the receiver to decode the received frame.

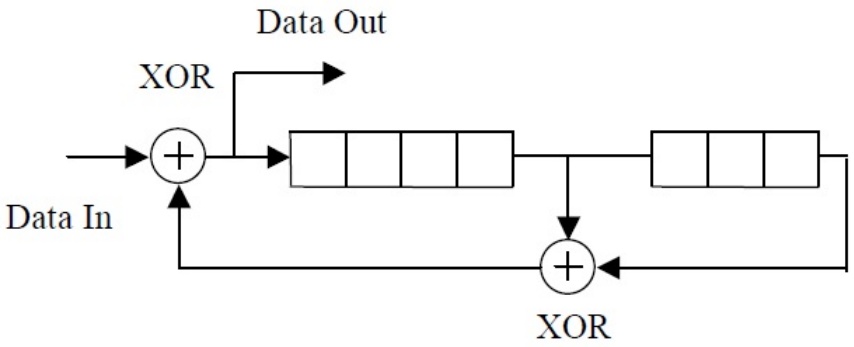

Figure 5.4. IEEE 802.11g data scrambler

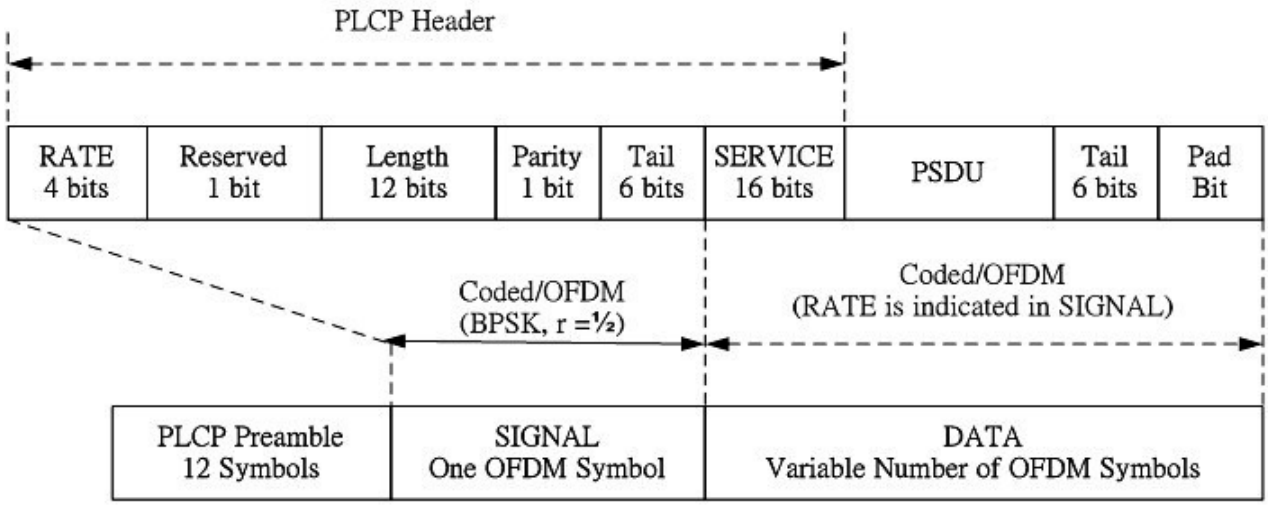

Figure 5.5. IEEE 802.11g PPDU frame format

Bits 0, 1, and 4 are reserved and are set to zero. In all 802.11g receivers, the transmit and symbol clocks are locked, so bit 2 is always set to 1. Value of bit 3, as the name suggests, is dependent on the modulation scheme. It is set when the frame body is modulated with Packet Binary Convolutional Code(PBCC), and set to zero for DSSS, CCK, and DSSS-OFDM modulations. The last three extension bits are used to assist receivers in determining the frame length in bytes from the Length field, which is expressed in terms of the number of microseconds required for transmission. In our implementation, though, the receiver determines the rate of the transmission using a command line argument since we are unable to develop a rate detector at the receiver USRP2. Hence, we just set these 3 bits to 0s.

Also, the formation of the PLCP frame involves correctly padding the data bits. The number of bits required for padding is calculated using:

$$N_{SYM} = \text{Ceil}((16 + 8 * LENGTH + 6)/N_{DBPS})$$

where N_{DBPS} is the number of data bits per OFDM symbol

$$N_{DATA} = N_{SYM} * N_{DBPS}$$

Now,

$$N_{PAD} = N_{DATA} - (16 + 8 * LENGTH + 6)$$

where N_{PAD} are the number of padding bits required.

5.2.4 Convolutional Encoder and Decoder

The convolutional encoder was developed in Python as well. In accordance with the IEEE 802.11g standard, it uses the generator coefficients as $g_0 = 133_8$, which is `1011011` in binary and $g_1 = 171_8$ (`1111001` in binary) with rate R = 1/2 and k = 7, as shown in Figure 5.6. Higher rates, i.e., for R = 2/3 or 3/4, are obtained by means of the `puncturing` operation. The Puncture block is implemented such that it does not transmit some of the encoded bits (also called stealing). At the receiver, these bits are inserted with a dummy zero value. An example of puncturing operation and further details are contained in [46].

At the receiver, Viterbi Convolution decoder is used. This has been implemented in Python and we referred the tutorial given in [47].

5.2.5 Interleaver

We implemented the interleaver in Python. The interleaver, as defined by the standard, is a two-permutation process onto the coded bits of a single OFDM block, containing a number N_{CBPS} of bits that are modulation-dependent. The first permutation ensures that adjacent bits are mapped onto nonadjacent subcarriers. The second one ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation. If we denote k as the position index of the coded bits before permutation, i as the position index after the first permutation and j the position index after the second permutation and prior to mapping, then the first permutation is as follows [48]:

$$i = (N_{CBPS}/16) \cdot (k \bmod 16) + \text{floor}(k/16)$$

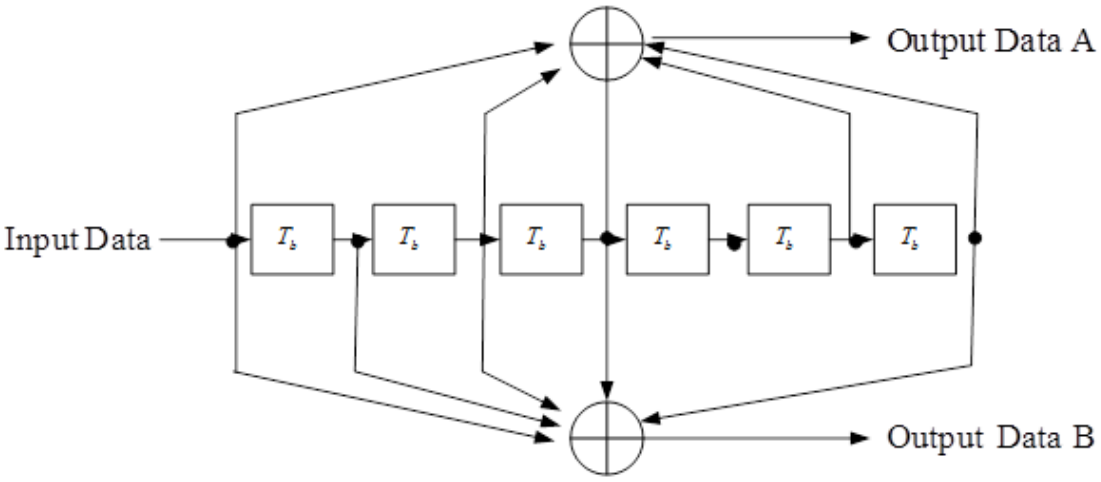

Figure 5.6. Convolutional encoder K = 7 block operation

where k = 0,1,....,N_{CBPS} and the function floor denotes the largest integer not exceeding the argument. The second permutation is given by the formula [48]:

$$j = s \cdot \text{floor}(i/s) + (i + N_{CBPS} - \text{floor}(16 \cdot i/N_{CBPS}))\text{mod}s$$

where i = 0,1,....,N_{CBPS} The value of s is determined by:

$$s = \max(N_{BPSC}/2, 1)$$

where N_{BPSC} is the number of coded bits per subcarrier. The permutation tables are given in the IEEE 802.11g standard document [48].

5.2.6 Symbols Mapping

The symbols mapping block divides the bits stream coming from the interleaver into groups of N_{CBPS} bits. In the standard 802.11g the OFDM subcarriers are modulated by using BPSK, QPSK, 16-QAM, or 64-QAM, depending on the RATE requested. The encoded and interleaved binary serial input data shall be divided into groups of N_{BPSC} (1, 2, 4, or 6) bits and converted into complex numbers representing BPSK, QPSK, 16-QAM, or 64-QAM constellation points as per the Table 5.1. We write the code for this block in C++ but we make use of static tables for storing the Grey-coded constellation mappings. The same technique is illustrated in the IEEE 802.11g standards document [48].

5.2.7 Pilot Tone Insertion

In each OFDM symbol, out of the 52 subcarriers, carriers -21, -7, 7 and 21 are reserved to the pilot signals. These pilots will be BPSK modulated. We write the code to insert the

[2]Coded bits per subchannel is a function of the modulation (BPSK, QPSK, 16-QAM, or 64-QAM).

[3]The data bits per symbol is a function of the rate of the convolutional code

Table 5.1. Encoding details for different OFDM data rates

Speed (Mbps)	Modulation and Coding Rate (R)	N_{CBPC}[2]	N_{CBPS}	N_{DBPS}[3]
6	BPSK, R=1/2	1	48	24
9	BPSK, R=3/2	1	48	36
12	QPSK, R=1/2	2	96	48
18	QPSK, R=3/4	2	96	72
24	16-QAM, R=1/2	4	192	96
36	16-QAM, R=3/4	4	192	144
48	64-QAM, R=2/3	6	288	192
54	64-QAM, R=3/4	6	288	216

pilot symbols in C++. To do so, we run a `for` loop to insert the pilot signals at the correct indexes.

5.2.8 OFDM Modulation

After the pilots are inserted and mapped, the OFDM modulation function is called. This is an inbuilt function in the GNURadio. However, we have to define the contribution of the pilot subcarriers which is given by the Inverse Fourier Transform, also provided by GNURadio. The polarity of the pilot subcarriers is controlled by a sequence, as defined by the IEEE 802.11g standard [48], which is a cyclic sequence of scrambled bits. We store this sequence in a static array in C++ and use it to find the polarity of the pilot signals during the pilot insertion phase.

5.3 Some Results

We set up the system to verify the behavior of the IEEE 802.11g transmitter. We compile and install our code into a PC equipped with an Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz, 2 GB of RAM, and an integrated Intel Gigabit Ethernet interface. The operating system environment is GNU/Linux (Ubuntu 9.04), with Python 2.6.2 and the GNURadio stable release 3.2.2. The Intel Corporation 82571EB Gigabit Ethernet Controller is connected to a USRP2 equipped with the daughter board XCVR2450 and an antenna with 5dBi gain. At the receiver side we use:

- An additional USRP2 connected to the same PC, used to analyse the received spectrum and also act as an 802.11g partial receiver
- An Atheros Communications Inc. AR5413 802.11abg NIC card in a PC to decode the transmitted frames

We use Wireshark [45] to decode the transmitted frames using the Atheros NIC card. We start transmitting the frames repeatedly. We run experiments with different frame lengths and rates.

An example run of sending 100 IEEE 802.11g frames with Rate = 6 Mbps and length of packet as 97 bytes. One of the metrics we analyse is the time difference between successive frames received at the receiver, in our case Wireshark running on an Ubuntu PC. The results are shown in Figure 5.7.

Also, some other statistics are shown in Table 5.2.

We then conduct some experiments to measure these metrics for the conventional 802.11 chipsets available in our laptops. The laptop on which we test has an Intel 5100 NIC card.

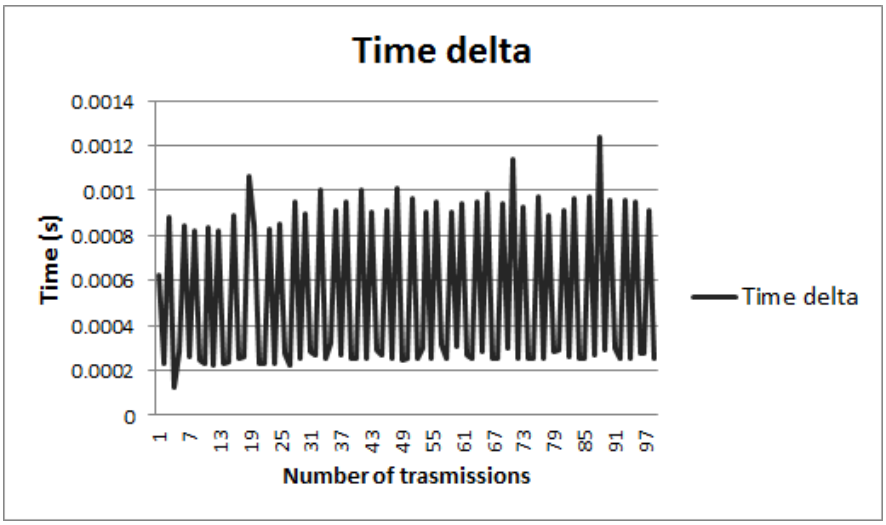

Figure 5.7. Time delta between successive received frames

Table 5.2. Some statistics of 802.11g transmission using USRP2

S.No	Metric	Value
1	Average transmission time for each packet	0.335739408(s)
2	Average delta between successive packets	0.529909 (ms)
3	Total transmission time	0.388273947 (s)

This laptop is the transmitter of 802.11g frames. The receiver is again an Ubuntu PC having an Atheros AR5413 NIC card. The results obtained are shown in Table 5.3. Please note that both the setups are identical. Both use burst transmission and also the placements of both the transmitter and the receiver are identical in both the setups. However, also note that the conventional protocol running on the Ubuntu PC and the laptop is also having the 802.11 MAC layer and hence, every packet will be backed off by a fixed (or a random in case of medium being busy) amount of time before being transmitted. However, the value of this back off time is usually of the order of ≈ 50 μs.

As seen by these results, our 802.11g implementation on the USRP2 performs very close to the industry standard though still some work needs to be done to support features like supporting different types of preambles, Super G mode and most importantly, the complete implementation of a corresponding receiver.

Table 5.3. Some statistics of 802.11g conventional transmission

S.No	Metric	Value
1	Average transmission time for each packet	0.173428(s)
2	Average delta between successive packets	0.227615 (ms)
3	Total transmission time	0.189628452 (s)

CHAPTER 6

CONCLUSION

6.1 Conclusion

Our system enables any users to send an emergency or a distress message to a Public-Safety Answering Point (PSAP) using any available 802.11 wireless access point or a commercial hotspot having Internet connectivity. The user is not required to authenticate or associate with the access point. The service is designed such that the emergency message can be given a higher priority than the existing 802.11 packets in the network. Further, the service provides an approximate location of the user to the PSAP. Hence, the service is fully capable of being a full-fledged public emergency service and can be employed in highly populated places having wireless Internet access such as airports, shopping complexes, commercial buildings, etc. Our emergency service requires minimal changes to the existing access points and can be made to work using almost any available wireless NIC cards on PCs, laptops, and mobile phones.

We have also developed and implemented an IEEE 802.11g transmitter and a partial receiver for the USRP using the GNURadio platform. This enables us to verify and test our service on a real hardware device. This also gave us the flexibility to modify the protocol to make the service more efficient.

Based on our evaluation, our service delivers the emergency message consistently in less than 5 seconds to the PSAP, which in our case, is an e-mail address.

6.2 Future Work

An emergency service must be robust against common security threats. As we point out in Section 2.6, our emergency service is vulnerable to a denial-of-service attack in which an attacker can keep on sending emergency packets to an access point while also changing the source MAC address. Genuine users cannot obtain the emergency service while this attack is going on. One of our important future goals is to address this security threat. Furthermore, we plan to deal with the problem of rogue or fake access points disrupting

the service. We will work towards a solution to this problem along the line of the research done by Jana et al. [49].

Another feature we plan to add to our emergency service is to have a provision for the PSAP to communicate with the sender of the emergency message. This can be done if the AP allows the PSAP to connect to the sender by allowing limited data connectivity for a short duration. The sender is allowed to communicate only with the PSAP and no other network nodes. However, this solution must also address the authentication of the sender and the PSAP. It must also prevent misuse of such a service.

We are currently working on completing the receiver capable of receiving and properly decoding IEEE 802.11g packets. Next, we plan to implement the access point functionality on the USRP2 so that it can act as a full-fledged access point capable of serving multiple users at the same time. This implementation will be helpful in collecting various measurements and benchmarks for the emergency service in Wi-Fi networks under different traffic and access point usage scenarios.

REFERENCES

[1] "Open source software-defined radio: A survey on gnuradio and its applications," http://userver.ftw.at/~valerio/files/sdrreport.pdf.

[2] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton, "Codeblue: An ad hoc sensor network infrastructure for emergency medical care," in *In International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[3] D. R. Oran and S. Gai, "System for discovering and maintaining geographic location information in a computer network to enable emergency services," Patent, 03, 2007.

[4] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: challenges and opportunities," *Pervasive Computing, IEEE*, vol. 3, no. 4, pp. 16 – 23, oct.-dec. 2004.

[5] J. J. Lichter, M. J. Meyer, and T. Moulos, "Enhanced emergency service for isdn based emergency services in a wireless telecommunications system," Patent, 07, 2001.

[6] S. N. Zellner, M. J. Enzmann, and R. T. M. Jr., "Multimedia emergency services," Patent, 11, 2009.

[7] "http://www.fcc.gov/pshs/services/911-services/enhanced911/Welcome.html."

[8] "http://www.fcc.gov/cgb/consumerfacts/wireless911srvc.html."

[9] A. S. Mintz-Habib, M.; Rawat and X. H., Wu, "A voip emergency services architecture and prototype," *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference*, pp. 523–528, 2005.

[10] J. Y. Kim, W. Song, and H. Schulzrinne, "Kim et al. an enhanced voip emergency services prototype an enhanced voip emergency services prototype abstract."

[11] "Commercial wireless systems international llc," http://wirelessfirealarm.com/.

[12] "http://www.fireangel.co.uk/Fire-Safety-Products.aspx."

[13] "http://en.wikipedia.org/wiki/Wi-Fi."

[14] "http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf."

[15] "Bbn technology - acert savane server: http://acert.ir.bbn.com/projects/adroitgrdevel/."

[16] "http://www.wired.com/epicenter/2010/11/fcc-911-texting/."

[17] A. Janefalkar, K. Josiam, and D. Rajan, "Cellular ad-hoc relay for emergencies (care)," in *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, vol. 4, sept 2004, pp. 2873 – 2877 Vol. 4.

[18] M. R. Arisoylu M and L. L. Rao R, "802.11 wireless infrastructure to enhance medical response to disasters," 2005.

[19] P. Pawelczak, R. Venkatesha Prasad, L. Xia, and I. Niemegeers, "Cognitive radio emergency networks - requirements and design," in *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on*, nov. 2005, pp. 601 –606.

[20] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *Communications Magazine, IEEE*, vol. 44, no. 1, pp. 74 – 82, jan 2006.

[21] S. Pavlopoulos, E. Kyriacou, A. Berler, S. Dembeyiotis, and D. Koutsouris, "A novel emergency telemedicine system based on wireless communication technology-ambulance," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 2, no. 4, pp. 261 –267, dec. 1998.

[22] Y.-C. Tseng, M.-S. Pan, and Y.-Y. Tsai, "Wireless sensor networks for emergency navigation," *Computer*, vol. 39, no. 7, pp. 55 –62, july 2006.

[23] "Design of a network independent emergency service," http://wiredspace.wits.ac.za/bitstream/handle/10539/2151/Goli%20Thesis.pdf.

[24] Consumer and Governmental Affairs Bureau, "Wireless 911 services," http://www.fcc.gov/cgb/consumerfacts/wireless911srvc.html.

[25] "VoIP :: 911 :: Regulation," http://www.cybertelecom.org/voip/911reg.htm.

[26] "Spectralink inc.: spectralink voice priority," 2005. [Online]. Available: http://www.spectralink.com/files/literature/SVPwhitepaper.pdf

[27] "Skyhook, inc." http://www.skyhookwireless.com.

[28] "Mexens llc navizon virtual gps service," http://www.navizon.com.

[29] "Fraunhofer iis autonomous wlan positioning system."

[30] "Google voice." [Online]. Available: https://www.google.com/voice

[31] E. Blossom, "Gnu radio: tools for exploring the radio frequency spectrum," *Linux J.*, vol. 2004, pp. 4–, June 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=993247.993251

[32] "Gnuradio - the gnu software radio." [Online]. Available: http://www.gnuradio.org

[33] "Universal software radio peripheral (usrp) m ettus: Ettus research llc." [Online]. Available: http://www.ettus.com

[34] "Usrp2." [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp2_v5.pdf

[35] "Tx and rx daughterboards," http://www.ettus.com/downloads/ettus_daughterboards.pdf.

[36] "Gnuradio modules," http://gnuradio.org/doc/doxygen/modules.html.

[37] "Simplified wrapper and interface generator http://www.swig.org/."

[38] "Eric blossom - how to write a signal processing block http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html."

[39] Dawei Shen, "Tutorial 10: Writing a signal processing block for gnu radio part i," 2005. [Online]. Available: http://www.snowymtn.ca/GNURadio/GNURAdioDoc-10.pdf

[40] "Eric blossom - how to write a gnuradio python application block http://gnuradio.org/trac/wiki/Tutorials/WritePythonApplications."

[41] "Bbn technology - acert savane server. http://acert.ir.bbn.com/."

[42] "Hamed firooz, implementation of full-bandwidth 802.11b receiver: http://span.ece.utah.edu/pmwiki/pmwiki.php?n=main.80211breceiver."

[43] "Implementation of an ieee 802.11p transmitter in open-source software defined radio," http://userver.ftw.at/~valerio/files/Costantini_masterthesis.pdf.

[44] "Cyclic redundancy check. http://www.hackersdelight.org/crc.pdf."

[45] "Wireshark," http://www.wireshark.org/.

[46] "Part11: Wireless lan medium access control and physical layer specifications. high-speed physical layer in the 5ghz band." [Online]. Available: http://standards.ieee.org/getieee802/

[47] "A tutorial on convolutional coding with viterbi decoding," http://home.netcom.com/~chip.f/viterbi/tutorial.html.

[48] "http://standards.ieee.org/getieee802/download/802.11g-2003.pdf."

[49] S. Jana and S. K. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," 2008.