Table of Contents:

1. Introduction

2. Abstract

3. State of the Art

4. Development

5. Conclusion

1. Introduction:

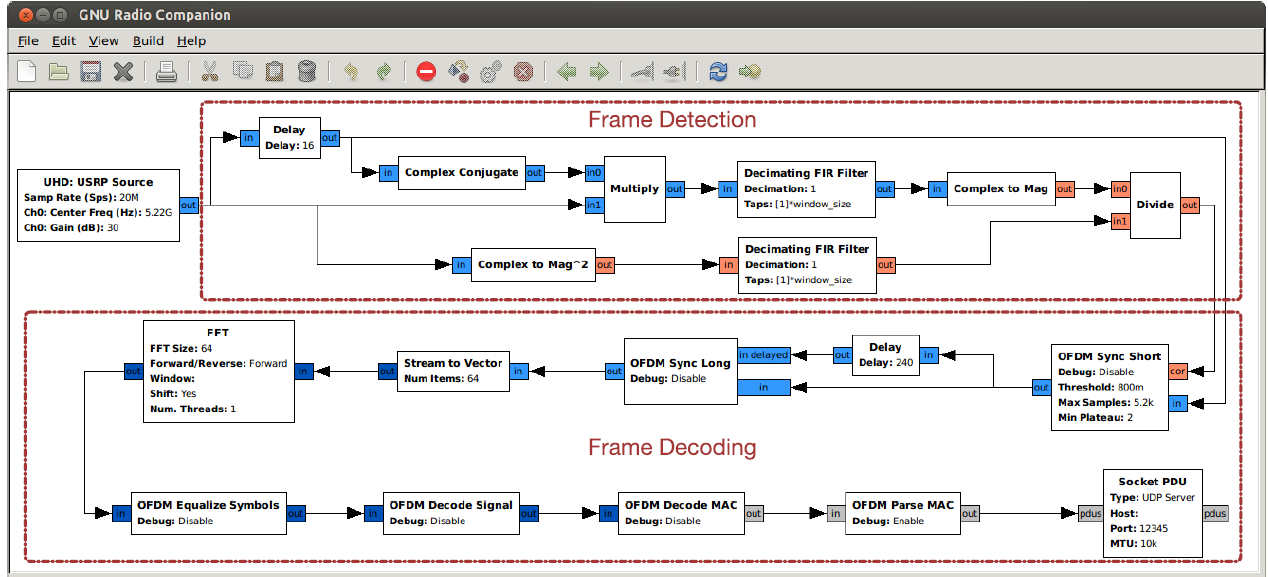
Google Summer of Code is an annual program, run by Google, for development of open source project by students during the summer each year. Since 2005 this program has been one of the major campaigns of Google to indulge more students into Open source communities. Shashank Gaur, who is a graduate student at ECE Paris, has been working under this program for GNURadio community with continuous mentoring from Ankit Kaushik from KIT. GNURadio is an open source signal processing software tool provided to implement SDRs along with hardware such as USRP provided by Ettus Research.

2. Abstract:

Being able to analyze what is working and how, is one of the strong points of any technology. GNURadio is one of those tools which gives user the ability to analyze each step in the chain of communication. From FFT plots to Constellation plots, many tools are provided by GNURadio community in order to help user to analyze each step. In this project, we(author?) is providing another great tool to analyze the real time traffic over wireless networks. Wireshark is a free and open source packet analyzer[R]. It is already used widely by most of the research community for network troubleshooting, protocol development etc[R]. Here we are trying to develop a Wireshark connector inside gnuradio which takes PDU packets as input and provide them to Wireshark software. The referenced receiver is IEEE 802.11 a/b/g receiver (Wireless receiver).

3. State of the Art

The base of this project is IEEE 802.11 a/p/g receiver developed by Bastian Bloessl at CCS-Lab, University of Innsbruck. The IEEE 802.11 a/b/g (OFDM) receiver, comprises all layers, hence it supports WiFi. This module uses some already developed tools such as VOLK, IT++ library, matched filtering etc. Its working is divided in two parts, Frame Detection and Frame Decoding respectively.



A small summary of its working is as following:

a. Frame Detection: It is worth to note that in this receiver module stream tagging is used to notify the start of an OFDM frame. Now the first thing in order to detect a frame is to know where the frame starts. It is done using an algorithm which is based on auto correlation of short training sequence. Once the algorithm detects the start of the frame, a fixed number of samples are passed to next step.

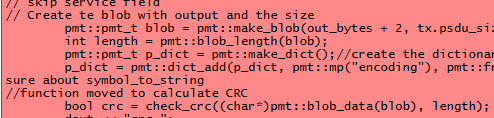
b. Frame Decoding: This involves few steps such as frequency and phase offset correction, symbol and magnitude alignment, Demodulation etc.In this step the output is provided once the descrambling is done.

The transmitter used here is the FTW 802.11 a/p/g encoder which generates frames using GNU Radio framework. It takes the payload of the mac-layer service data unit as input and prepends a static MAC header. After calculating and appending the CRC32 bits it generates the OFDM modulated frame in digital complex baseband representation, initializes the USRP2 and pushes the corresponding block of samples to the USRP2 floating-point sink. The original version of the cgran project was not developed for UHD, hence it had to be modified accordingly.

In this particular case, the generated frames are both transmitted using USRP N210 and saved in file sink (which enables use of the file as source time to time).

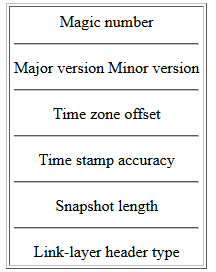
4. Development

The first major step of development is to understand the receiver written by Bastian Bloessl. With the continuous help from Bastian himself, it was pretty easy to catch up with the code and get answers of simple questions, such as what why and how. In order to provide wireshark with appropriate input, it is required to have all MAC headers. Hence some of the functionality between Parse Mac and Decode Mac blocks has to be moved around in order to accommodate wireshark connector. Stuff like CRC, Checksum extra was moved around and PDU creation was done.



PDU’s are created and a dictionary is made to save the encoding of ofdm (BPSK/QPSK/QAM16 etc). Once this is done, the PDUs are passed to the wireshark connector which evidently will create pcap headers and write it to the file (pipe). There are two type of PCAP headers, one per-file and another per-packet.

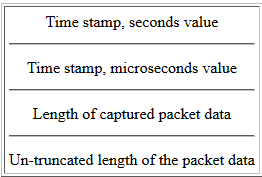
The per-file header can be described by following figure.



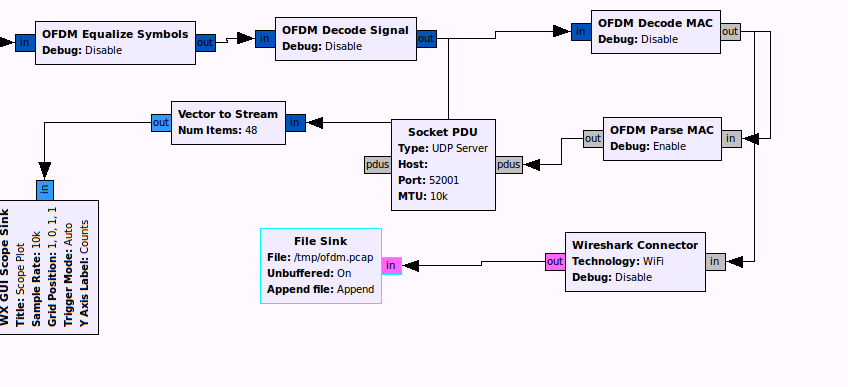
The first field in the per-file header is a 4-byte magic number, with the value 0xa1b2c3d4. Following this other values are:

* + A 2-byte file format major version number; the current version number is 2.
  + A 2-byte file format minor version number; the current version number is 4.
  + A 4-byte time zone offset; this is always 0.
  + A 4-byte number giving the accuracy of time stamps in the file; this is always 0.
  + A 4-byte number giving the "snapshot length" of the capture; packets longer than the snapshot length are truncated to the snapshot length, so that, if the snapshot length is *N*, only the first *N* bytes of a packet longer than *N* bytes will be saved in the capture. This is always 65535
  + A 4 -byte number giving the link-layer header type for packets in the capture. Here we have two possible values for ZigBee and WiFi.

After per-file header there can be packets which shall have per-packet header. The format of per-packet header is as following:



Once the pcap packets are ready they can be passed to a pipe which is connected to wireshark. Wireshark is started with using popen and pipe is created. Also using mkfifo the pcap file is used. To write into the FIFO, it is important that the File Sink is added with append option.



The IEEE 802.11 a/p/g receiver is also ported to the OFDM blocks of gnuradio in order to provide the user possibility to simply choose the option of using the rx.

5. Conclusion

To conclude, the wireshark connector is able to create pcap packets and write to pcap file which can be opened with wireshark. The