An IEEE 802.11a/g/p OFDM Receiver for GNU Radio

Bastian Bloessl*, Michele Segata*†, Christoph Sommer* and Falko Dressler*

*Institute of Computer Science, University of Innsbruck, Austria
†Dept. of Information Engineering and Computer Science, University of Trento, Italy
{bloessl,segata,sommer,dressler}@ccs-labs.org

ABSTRACT

Experimental research on wireless communication protocols frequently requires full access to all protocol layers, down to and including the physical layer. Software Defined Radio (SDR) hardware platforms, together with real-time signal processing frameworks, offer a basis to implement transceivers that can allow such experimentation and sophisticated measurements. We present a complete Orthogonal Frequency Division Multiplexing (OFDM) receiver implemented in GNU Radio and fitted for operation with an *Ettus USRP N210*. To the best of our knowledge, this is the first prototype of a GNU Radio based OFDM receiver for this technology. Our receiver comprises all layers up to parsing the MAC header and extracting the payload of IEEE 802.11a/g/p networks. It supports both WiFi with a bandwidth of 20 MHz and IEEE 802.11p DSRC with a bandwidth of 10 MHz. We validated and verified our implementation by means of interoperability tests, and present representative performance measurements. By making the code available as Open Source we provide an easy-to-access system that can be readily used for experimenting with novel signal processing algorithms.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design — *Wireless communication*; C.3 [**Special-purpose and Application-based Systems**]: Signal processing systems

Keywords

OFDM Receiver, SDR, GNU Radio, IEEE 802.11a/g/p

1. INTRODUCTION

Orthogonal Frequency Division Multiplexing (OFDM) is used in almost all current and forthcoming wireless communication standards. Besides cellular standards (like WiMAX and LTE Advanced) and digital broadcasting standards (like DVB-T), it is also used in many of the IEEE 802.11 standards, i.e., the different WiFi variants. With OFDM, data is transmitted in parallel on multiple, orthogonal subcarriers. Compared to single carrier systems, this poses a multitude of new challenges for the hardware and asks for new signal processing algorithms to cope with OFDM characteristics, like a high Peak to Average Power Ratio (PAPR) [8].

Given the wide range of applications, OFDM gained a lot of attention in the academic community: many new algorithms have been proposed for frame detection, frequency offset correction, and channel estimation [3, 6, 13]. Furthermore, OFDM and its applicability in different scenarios and channels has been studied extensively by means of analytical methods and by means of simulation [6, 9].

Yet, the possibility to conduct experimental research in that field is extremely limited.

On one end of the spectrum lies experimentation with Commercial Off-The-Shelf (COTS) hardware, but this approach is limited to Received Signal Strength (RSS) and throughput measurements: the functionality of the physical layer (and, in part, also the MAC layer) is realized in Application Specific Integrated Circuits (ASICs) and is therefore static. For new protocol standards such as IEEE 802.11p Dedicated Short Range Communications (DSRC) there is no consumer hardware available; instead, research is conducted with expensive hardware prototypes [12]. Also, new physical layer and new signal processing algorithms can not be integrated.

On the other end of the spectrum lies experimentation with custom radio prototypes [15], which are usually based on Field-Programmable Gate Arrays (FPGAs), i.e., rather complex and inflexible. Even though this approach offers high performance, investigations typically have to focus on small parts of the receive chain, as an implementation of the complete transceiver, together with the design of the hardware platform, incurs substantial effort. Furthermore, the code for custom devices can neither be reused nor verified, nor can results be reproduced by other researchers.

Generic Software Defined Radios (SDRs) such as the well-known *WARP* [7] and *Ettus USRP*[1] platforms combine the advantages of both.

In this paper, we present a complete OFDM receiver implemented based on GNU Radio and fitted for operation on an *Ettus USRP N210*. This is, to the best of our knowledge, the first prototype of such an SDR based OFDM receiver supporting channel bandwidths up to 20 MHz – its counterparts, OFDM transmitters using GNU Radio, are already available, e.g., the one developed by Fuxjäger et al. [5].

[1] `http://www.ettus.com/`

SRIF'13, August 12, 2013, Hong Kong, China.

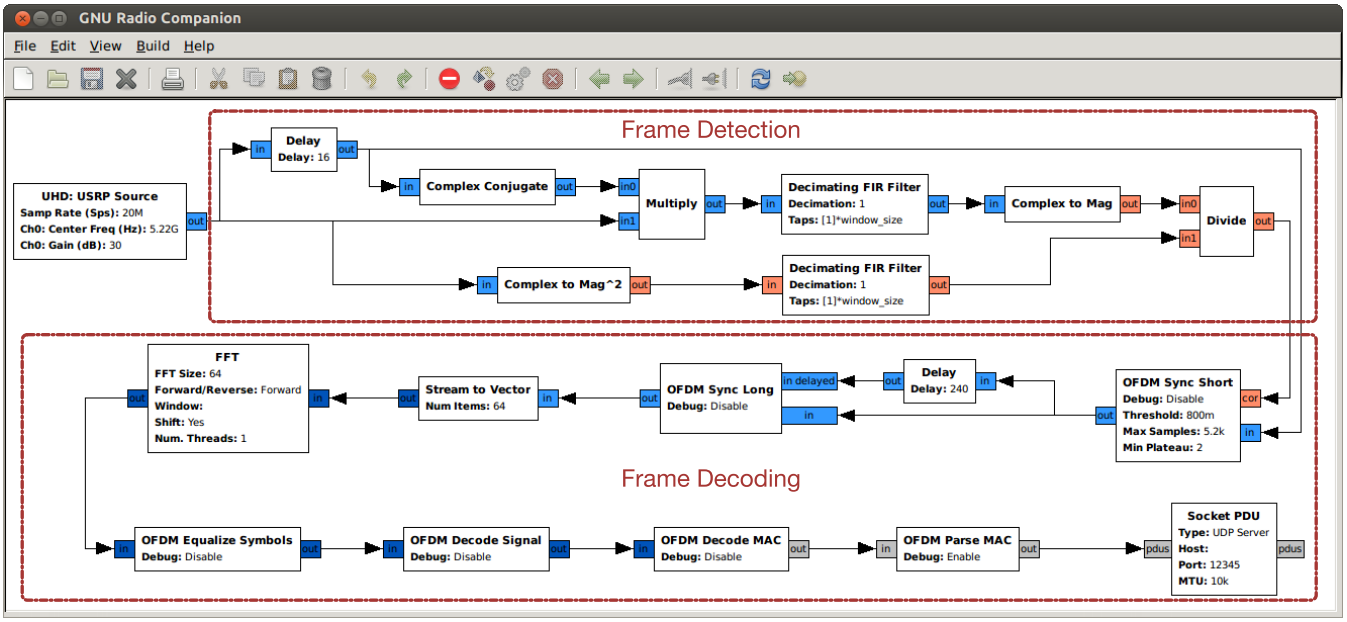

Figure 1: Overview of the blocks comprising the OFDM receiver in GNU Radio Companion.

Matt Ettus, the developer of the *USRP* series of devices, supplies an initial GNU Radio OFDM receiver using maximum likelihood estimation [14] and pseudonoise (PN) sequence correlation [11]. The applicability of this receiver to IEEE 802.11a/g/p, however, is limited as the system does not support the required bandwidth.

Moreover, our receiver comprises both the physical layer as well as the complete decoding process including the MAC layer of IEEE 802.11a/g/p networks. It supports both all WiFi variants with 20 MHz channel bandwidth as well as any IEEE 802.11p DSRC systems with 10 MHz channel bandwidth.

Our main contributions can be summarized as follows:

- We present the first OFDM receiver for the GNU Radio real-time signal processing framework supporting IEEE 802.11a/g/p.
- The receiver is able to decode the signal for up to 20 MHz channel bandwidth using a normal desktop PC and without any changes to the firmware of the FPGA.
- The receiver comprises both the physical layer as well as the complete decoding process including the MAC.
- We make the code available[2] as a modular package of completely Open Source building blocks and provide an easy-to-access system that can be readily used as a tool for experimenting with novel signal processing algorithms.

2. GNU RADIO OFDM RECEIVER

We implemented the OFDM receiver using the GNU Radio real-time signal processing framework, which is Open Source software and well-accepted in the wireless communications research community. As hardware front-end of the SDR system, we use an *Ettus USRP N210*.

[2]`http://www.ccs-labs.org/software/`

2.1 Overview

As illustrated in Figure 1 the structure of the OFDM receiver is completely exposed to GNU Radio Companion, a graphical tool to setup and configure signal processing flow graphs. The receiver is divided into two functional parts: The first part, depicted in the top half, is responsible for frame detection. The second part, shown in the bottom half, is responsible for decoding the frame. In the following, we briefly discuss some specific GNU Radio features we used, before explaining the signal processing blocks in detail.

Stream tagging: GNU Radio was initially designed for stream based signal processing. Stream tags have been introduced to annotate the sample stream with further meta data, like sampling frequency, carrier frequency, or timestamps. We employ stream tagging to signal the start of an OFDM frame, and in a later stage the length and encoding scheme of the frame.

Message passing: GNU Radio is often used to implement packet based transceiver systems, e.g., IEEE 802.11 or 802.15. As the implementation of such technologies is complicated in a stream-based environment, asynchronous message passing was introduced. Messages can, like stream tags, encapsulate arbitrary information. Thus, processing blocks can work on complete packets and switch to stream-based processing only at selected stages in the signal processing chain.

Vectorized Library of Kernels (VOLK): In order to be able to support sample rates of 20 Msps for IEEE 802.11a/g and 10 Msps for IEEE 802.11p respectively, we make use of VOLK [10], a toolkit that eases the use of Single Instruction Multiple Data (SIMD) instructions. It provides wrapper functions for the most common signal processing tasks and dynamically selects the implementation which offers the highest performance on the host system. SIMD instructions work on vectors instead of scalars, which speeds up the signal processing considerably [10]. VOLK also takes care of all platform dependent issues of vectorized instructions, allowing the user to write platform independent code.

2.2 Frame Detection

The first task in the receive chain is to actually detect the start of an OFDM frame. Each IEEE 802.11a/g/p frame starts with a short preamble sequence, which consists of a pattern that spans 16 samples and repeats ten times. The employed frame detection algorithm has been introduced in [3]. It is based on the autocorrelation[3] of the short training sequence.Following [3, Algorithm 1], we exploit this cyclic property and calculate the autocorrelation value a of the incoming sample stream s with lag 16 by summing up the autocorrelation coefficients over an adjustable window N_{win} (here, $\bar{\text{s}}$ denotes the complex conjugate of s):

$$\text{a}[n] = \sum_{k=0}^{N_{\text{win}}-1} \text{s}[n+k]\,\bar{\text{s}}[n+k+16]. \tag{1}$$

The summation over the window (which finally results in the calculation of a moving average) acts as a low-pass filter. We experimented with different window sizes and found 48 to work well. Due to the cyclic property of the short training sequence, the autocorrelation is high at the start of an IEEE 802.11a/g/p frame.

In order to be independent of the absolute level of incoming samples, we normalize the autocorrelation with the average power p and calculate the autocorrelation coefficient c as

$$\text{p}[n] = \sum_{k=0}^{N_{\text{win}}-1} \text{s}[n+k]\,\bar{\text{s}}[n+k]; \tag{2}$$

$$\text{c}[n] = \frac{|\text{a}[n]|}{\text{p}[n]}. \tag{3}$$

Here, $|\text{a}[n]|$ denotes the magnitude of $\text{a}[n]$. A typical graph of c during frame reception is depicted in Figure 2. It clearly shows the plateau of high autocorrelation coefficients during the short training sequence. In our OFDM receiver, we consider that there is a plateau if three consecutive samples are over a configurable threshold. For every detected frame, we then pipe a fixed number of samples to subsequent blocks in the flow graph.

An annotated overview of the frame detection blocks in GNU Radio companion is depicted in Figure 1. It can be seen that we split the calculation of the autocorrelation coefficient in eight blocks and realize all operations with standard operations in GNU Radio. All the involved blocks make use of the already mentioned VOLK library. The gained speedup is crucial for the receiver, as the blocks involved in frame detection have to process the sample stream from the *USRP* at full speed.

We implemented the *OFDM sync short* block to act like a valve. Its inputs are the samples from the *USRP* and the normalized autocorrelation coefficient. If it detects a plateau in the autocorrelation stream, it pipes a fixed number of samples into the rest of the signal processing pipeline; otherwise it drops the samples.

Of course, this approach comes with some limitations. First, the size of the frames that can be decoded is limited to a configurable number of OFDM symbols, and secondly, if another frame arrives shortly after the first one, it will not be detected. If we would set the maximum number of samples that we stream into the rest of the flow graph according to the maximum number of OFDM symbols per frame, we could circumvent the size limitation. Further, we could monitor the autocorrelation also during copying of the frame and mark the start of a new frame when another plateau is detected. This way, we would not miss frames with a short time-lag, for example, a Clear To Send (CTS) following a Request To Send (RTS).

[3]Matched filtering would be more robust in order to detect a known sequence, however it would require 16 complex multiplications per input sample instead of only one. We therefore use matched filtering only at a later stage of the OFDM receiver, which needs to process less data.

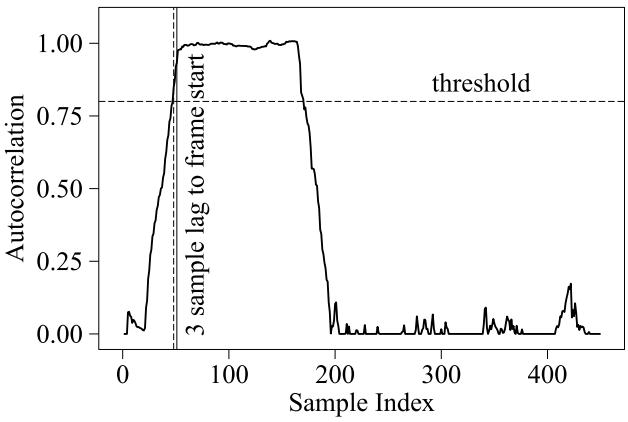

Figure 2: Characteristic behavior of the autocorrelation function as calculated in the frame detection part of the receiver during frame reception.

2.3 Frequency Offset Correction

The next block in the receive chain is *OFDM Sync Long*, which applies frequency offset correction and symbol alignment. Frequency offset correction is required, as the local oscillators of sender and receiver might work on slightly different frequencies.

To compensate that, we utilize the algorithm suggested in [13]. Currently, we use only the short training sequence for estimating the frequency offset between sender and receiver. The intuition behind this algorithm is the following: Ideally, during the short sequence a sample $s[n]$ should correspond to the sample $s[n+16]$ due to its cyclic property. However, if noise and a frequency offset are introduced, this is no longer the case, and $s[n]\,\overline{s}[n+16]$ is not a real number, as in the idealized case. Neglecting noise, the argument of that product corresponds to 16 times the rotation that is introduced by the frequency offset between samples. To estimate the final value, averaging is applied and the final value for the frequency offset $\text{d}f$ is calculated as

$$\text{d}f = \frac{1}{16} \arg\left(\sum_{n=0}^{N_{\text{short}}-1-16} s[n]\,\overline{s}[n+16] \right), \tag{4}$$

where N_{short} is the length of the short training sequence.

Using the argument of the sum of the products (instead of considering the mean argument of the products) is much more robust against noise, as samples with small magnitudes which are more affected by noise are weighted less. Finally, the frequency offset is applied to each sample as

$$s[n] \leftarrow s[n]\, e^{\,\text{i}\,(n\,\text{d}f)}. \tag{5}$$

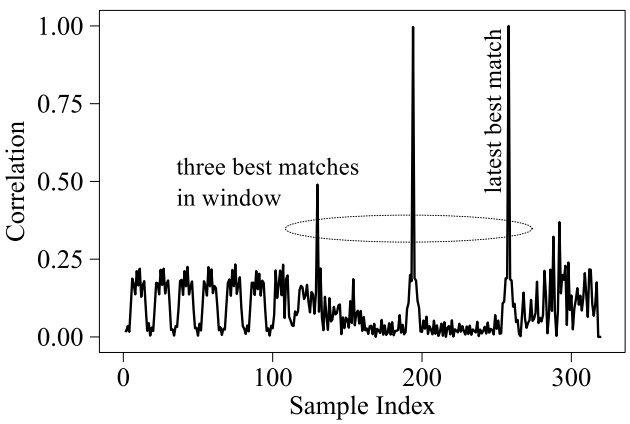

Figure 3: Characteristic behavior of the correlation of the input stream with the known sequence calculated in the *OFDM Sync Long* block.

2.4 Symbol Alignment

The *OFDM Sync Long* block is also responsible for symbol alignment. Each OFDM symbol spans 80 samples, consisting of 16 samples of cyclic prefix and 64 data samples. The task of symbol alignment is to calculate where the symbol starts, to extract the data symbols, and to feed them to an algorithm doing a Fast Fourier Transformation (FFT). This alignment is done with the help of the long training sequence, which is composed of a 64 sample long pattern that repeats 2.5 times. As this block needs only act on a subset of the incoming sample stream, and as the alignment has to be very precise, we employ matched filtering for this operation.

In Figure 3, a typical graph showing the correlation of the input stream with the known sequence is reproduced. The two characteristic peaks are very dominant and narrow, thus allowing very precise symbol alignment.

We calculate the indices of the highest three peaks as

$$N_{\mathcal{P}} = \underset{n\in\{0,\ldots,N_{\text{preamble}}\}}{\arg\max_3} \sum_{k=0}^{63} \mathrm{s}[n+k]\,\overline{\mathrm{LT}}[k], \tag{6}$$

where N_{preamble} corresponds to the added length of the short and long preambles, LT is the repeating pattern of the long training sequence spanning 64 samples, and $\arg\max_3$ returns the top 3 indices maximizing the expression.

The first data symbol thus starts at sample index

$$n_{\mathcal{P}} = \max\left(N_{\mathcal{P}}\right) + 64, \tag{7}$$

as the latest peak of the matched filter output is 64 samples before the end of the long training sequence.

With the relative position of the first data symbol known, this block can extract the data symbols, then pass chunks of data samples that correspond to one symbol to subsequent blocks in the flow graph. The first symbol of each OFDM frame is tagged, so that the following blocks are able to recognize the frame start.

Knowing the start of the data symbols, we can remove the cyclic prefix by subsetting the data stream and grouping the samples that correspond to individual data symbols as

$$\mathrm{s} \leftarrow \Big(\underbrace{\mathrm{s}[n_{\mathcal{P}}+16],\ldots,\mathrm{s}[n_{\mathcal{P}}+79]}_{\text{first symbol}},\underbrace{\mathrm{s}[n_{\mathcal{P}}+80+16],\ldots}_{\text{second symbol}}\Big). \tag{8}$$

2.5 Phase Offset Correction

The next step is the transition from time to frequency domain, which is done by the FFT block.

Following the FFT, the *OFDM Equalize Symbols* block is the first one in frequency domain and is responsible for phase offset correction and channel estimation. As the sampling times of sender and receiver are not synchronized and as the symbol alignment is not perfect, a phase offset is introduced. This phase offset is linear with frequency and can be corrected with the help of pilot subcarriers. IEEE 802.11 mandates four pilot subcarriers that encode a predefined BPSK constellation which is the same for each frame, but changes from symbol to symbol. Thus, the symbol index within the frame has to be known; it is signaled by a tag in the sample stream that is added by the *OFDM Sync Long* block. Based on the four pilots the phase offset is estimated by a linear regression and compensated.

2.6 Channel Estimation

Besides the phase, also the magnitude of the carriers has to be corrected, which is also performed by the *OFDM Equalize Symbols* block. This is especially important if QAM-16 or QAM-64 encoding is utilized, where also the magnitude carries information. Non-linearities in the magnitude might be caused by imperfect channel filters in the hardware. The current implementation of our block assumes the magnitude of the carriers to be sinc-shaped and corrects based on that assumption. However, this shape could be seen to depend also on the sender, as we experienced differences when we using different transmitters. Thus, this equalization needs some further improvement, as it currently restricts our receiver to BPSK and QPSK modulations.

This block also removes DC, guard and pilot subcarriers and thus subsets the 64 symbol input vector into 48 symbols.

2.7 Signal Field Decoding

The next block in the chain is called *OFDM Decode Signal*. In each frame, the short and long training sequences are followed by the signal field, which is a BPSK modulated OFDM symbol encoded with a rate of $1/2$ that carries information about the length and encoding of the following symbols. Again, the start of the frame and, thus, the position of the signal field is tagged in the sample stream. For decoding of the convolutional code, the IT++ library is used.

If the signal field is decoded successfully, i.e., if the rate field contains a valid value and if the parity bit is correct, *OFDM Decode Signal* annotates the sample stream with a tag, carrying a tuple of encoding and length of the frame. This tag is used by the following block to decode the payload.

2.8 Frame Decoding

The final step in the receiver is the decoding of the actual payload. It is performed in multiple sub-steps, as follows.

Demodulation: The *OFDM Decode MAC* block receives vectors of 48 constellation points in the complex plane, corresponding to the 48 data subcarriers per OFDM symbol. According to the used modulation scheme, these constellations are mapped to floating point values, representing the soft-bits of the employed modulation.

Deinterleaving: Dependent on the Modulation and Coding Scheme (MCS), the bits of a symbol are permuted. The permutation is the same for all symbols of a frame.

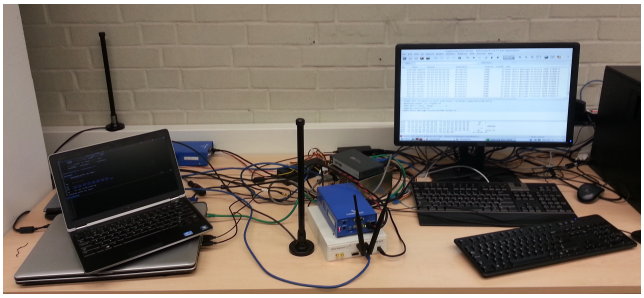

Figure 4: Setup of our interoperability test: the laptops on the left as well as the blue MK2 were used to transmit IEEE 802.11a/g/p frames, which have been received by the gray *Ettus USRP N210* in the middle.

Convolutional Decoding and Puncturing: For decoding of the convolutional code and puncturing, the IT++ library is again utilized.

Descrambling: The final step in the decoding process is descrambling. In the encoder the initial state of the scrambler is set to a pseudo random value. As the scrambler is implemented with a seven bit feedback shift register, $2^7 = 128$ initial states are possible. The first 7 bit of the payload are part of the service field and always set to zero, in order to allow the receiver to deduce the initial state of the scrambler. The mapping from these first bits to the initial state is implemented via a lookup table.

Output: After the decoding process, the payload is packed into a GNU Radio message and passed to subsequent blocks in the flow graph.

As final endpoint of the flow graph, we use a *Socket PDU* block of type *UDP Server* that sends the payload reencapsulated in a User Datagram Protocol (UDP) datagram. A user can then receive the datagrams, e.g., with *netcat* and see the payload appearing in their terminal. Therefore, the flowgraph can be easily extended with custom applications.

3. INTEROPERABILITY

For all tests and experiments, we used an *Ettus USRP N210* with an *XCVR2450* daughterboard as RF frontend, which allows us to tune to the Industrial Scientific and Medical (ISM) bands at 2.4 GHz and 5 GHz as well as to the DSRC band at 5.9 GHz.

As an initial evaluation step, we tested the interoperability of our GNU Radio OFDM receiver for different technologies. The aim was to verify basic compliance of our receiver to the IEEE 802.11a/g/p standards – not yet measuring quantitative performances metrics, such as Packet Delivery Ratio (PDR). All the mentioned standards use OFDM at the physical layer but differ in parameters like carrier frequency and bandwidth.

Table 1 summarizes the results from all interoperability tests using different IEEE 802.11a/g/p transmitters conducted in our testbed (Figure 4).

First, we investigated the IEEE 802.11g standard, which, like IEEE 802.11b, uses the 2.4 GHz ISM band. It defines 14 channels with a bandwidth of 20 MHz each [2]. Since IEEE 802.11g is a high data rate extension to IEEE 802.11b, it is designed to be backward compatible. For that reason an IEEE 802.11g network usually uses the same preamble and physical header as IEEE 802.11b networks. However,

NIC	Standard	Bandwidth	
MacBook Pro	802.11a/g	20 MHz	✓
Intel Ultimate-N 6300	802.11a/g	20 MHz	✓
Air Live X.USB	802.11a/g	20 MHz	✓
Cohda MK2	802.11p	10 MHz	✓
Unex DCMA-86P2	802.11a/p	10/20 MHz	✓

Table 1: Selection of WiFi and IEEE 802.11p devices we verified to be interoperable with the receiver.

an IEEE 802.11g Network Interface Card (NIC) can also be switched to a pure OFDM mode, called Extended Rate PHY (ERP) OFDM in the standard, when all devices in the network support IEEE 802.11g.

Since our receiver only supports pure OFDM, we used that mode for our tests in the 2.4 GHz band. In particular, by setting up an ad hoc network between a MacBook Pro and a laptop with an Intel Centrino Ultimate-N 6300 WiFi card, we were able to verify that all frames were decoded correctly by the SDR. More precisely, we overhear all kinds of frames in that network, ranging from management frames (i.e., beacons), data frames, and control frames, like an RTS.

As stated in Section 2, a current limitation of the receiver is that the frame detection block pipes a fixed number of samples into the rest of the flow graph. For that reason, we miss most of the CTS frames as the receiver is still synced on the corresponding RTS that precedes the CTS frame.

In a second experiment, still using the same devices, we investigated the compatibility with IEEE 802.11a networks. IEEE 802.11a networks also have a channel bandwidth of 20 MHz and work exactly like IEEE 802.11g networks in ERP mode, but at 5 GHz. We executed the same test as for 2.4 MHz and, again, were able decode all radio packets we captured.

As stated in Section 2, we support BPSK and QPSK modulation, each with coding rate $1/2$ and $3/4$, leading to four modulation and coding schemes. We were able to verify that all four currently supported encoding schemes work.

We were also able to verify that the frame detection blocks of the receiver are able to keep up with a sample rate of 20 Msps, as we did not experience any overruns of the input buffer during the tests. Thus, we can conclude that the basic decoding algorithm works, works fast enough, and is implemented correctly.

A final experiment has been conducted in order to assess the compatibility with IEEE 802.11p. The IEEE 802.11p DSRC standard is a version of the IEEE 802.11 standard designed for Inter-Vehicle Communication (IVC) [1, 4]. It is based on an OFDM physical layer operating in the dedicated 5.9 GHz frequency band, which has been specifically reserved for vehicular communications. Essentially, it is based on IEEE 802.11a with slightly changed timings to cope with Doppler effects, the channel bandwidth having been reduced to 10 MHz, halving all bitrates, now ranging from 3 Mbit/s to 27 Mbit/s.

To perform the IEEE 802.11p test, we used two different DSRC prototype devices. The first device is a Cohda Wireless MK2[4], an integrated IEEE 802.11p / IEEE 1609 DSRC/WAVE solution. It is intended to operate as an On

[4] `http://www.cohdawireless.com/`

Component	Type
CPU	Intel Core i7-2600 CPU 3.40GHz
NIC	RTL-8169 Gigabit Ethernet
Operating System	Ubuntu 12.04 LTS, 64 bit
GNU Radio	Version 3.6.4
SDR	Ettus Research N210 revision 4
Daughterboard	XVCR2450
Antenna	VERT2450 (3 dBi)

Table 2: Overview of the most important components of our test system.

Board Unit (OBU) or as a stand-alone Roadside Unit (RSU). The MK2 is the basis for major field operational tests in USA, Australia, Germany, France, and Korea. The device is highly customizable, in particular it is possible to set any kind of wireless parameter, including the channel, transmit power, as well as the modulation and coding scheme.

The second type of device is a Unex DCMA-86P2 Mini PCI card, which is based on an Atheros IEEE 802.11a chip set and implements only the physical layer of IEEE 802.11p. These cards have been successfully used by many of the Grand Cooperative Driving Challenge (GCDC)[5] participants. Like other WiFi cards, the Mini PCI NICs can be installed in off-the-shelf laptops. All layers above the physical layer are provided by the Linux kernel and user space utilities, which we slightly adapted to support IEEE 1609 WAVE.

In order to assess compatibility between our SDR solution and the mentioned devices, we transmitted IEEE 802.11p frames from both devices. We were able to verify that frame decoding was successful in all experiments, showing that the GNU radio receiver is also compatible to IEEE 802.11p networks with a channel bandwidth of 10 MHz.

4. PERFORMANCE MEASUREMENTS

In a second evaluation step, we investigated the performance of the receiver quantitatively by means of PDR curves for different modulation and coding schemes. Information on the most important hard- and software components of our test system is listed in Table 2. The tests were performed by sending frames at a rate of 5 packets per second from IEEE 802.11a/g/p devices to our GNU Radio OFDM receiver using the *Ettus USRP N210*. The packet size was 63 Byte, consisting of 30 Byte MAC header including 4 Byte Frame Check Sequence (FCS), 8 Byte IEEE 802.2 Logical Link Control (LLC), and 25 Byte of data payload.

At the receiver, we logged the number of successfully decoded frames. If a frame was decoded correctly was decided based on the 4 Byte FCS, which is part of the MAC Protocol Data Unit (PDU).

As the experiments were carried out in our office environment and, thus, space was limited, we had to insert a 30 dB attenuator before the transmit antenna in order to decrease the signal power and to actually experience packet loss. The distance between the antennas was approximately 6 m. We performed measurements for both IEEE 802.11a networks and IEEE 802.11p networks.

For the IEEE 802.11a/g measurements with a bandwidth of 20 MHz, we decided to avoid the 2.4 GHz band due to the high amount of interference sources, which could invalidate

[5] `http://www.gcdc.net/`

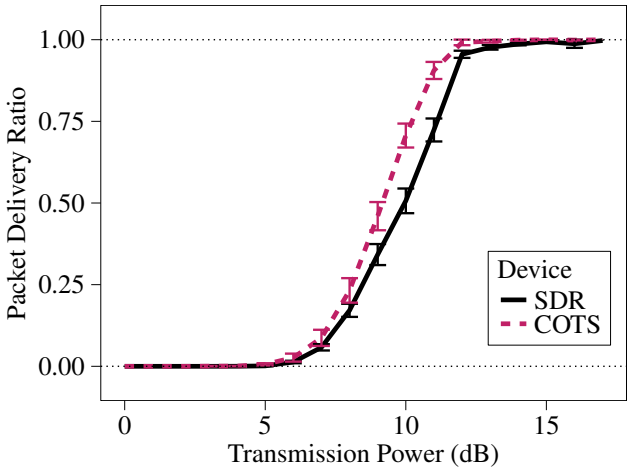

Figure 5: Packet delivery ratio of IEEE 802.11a packets, sent from a Unex device. The packet size is 95 Byte, all packets are BPSK modulated with coding rate R=1/2.

the results. Instead, we performed all measurements on the 5 GHz ISM band, which is less crowded compared to the 2.4 GHz band. The only wireless card in the lab that actually supports packet injection in the 5 GHz band using the default Linux driver is the aforementioned Unex WiFi card, which, to the best of our knowledge, cannot be forced to use an arbitrary modulation and coding scheme. For that reason, we had to stick to the lowest MCS, i.e., BPSK with a coding rate of 1/2, resulting in a bitrate of 6 Mbit/s.

At the transmitter we varied the transmission power from 0 dBm to 18 dBm in steps of 1 dBm.

Furthermore, we also started a COTS WiFi receiver and logged the overheard packets, in order to compare the performance of our implementation with consumer hardware. As COTS device, we used an *Air Live X.USB* dongle since the device has easy to access antenna connectors. The average PDR of 200 measurement runs, together with the 95 % confidence intervals are depicted in Figure 5. Note that in contrast to typical PDR curves we plot transmission power on the X-axis and not the Signal to Noise Ratio (SNR). This stems from the fact that the receiver does not log any SNR values and thus, the values on the X-axis are not to be interpreted as absolute, but relative. Furthermore, the relative power levels between both receiving devices should not deviate much, as the antenna setup is the same.

The results show that the performance of the receiver is comparable to consumer grade devices. It is especially worth noting that the PDR curve approaches one for higher transmission powers and, thus, we can conclude that we do not introduce any systematic errors in the receive chain. Furthermore, also the power interval in which the PDR curve of the SDR rises matches the interval of the COTS device very well.

For the IEEE 802.11p measurements with a bandwidth of 10 MHz, we used the *Cohda MK2* devices, since, in contrast to the Unex cards, they allow us to set all MCSs that the IEEE 802.11p standard mandates. In our measurements we made use of BPSK and QPSK modulation, each with coding rates 1/2 and 3/4, resulting in four different MCSs. We did not

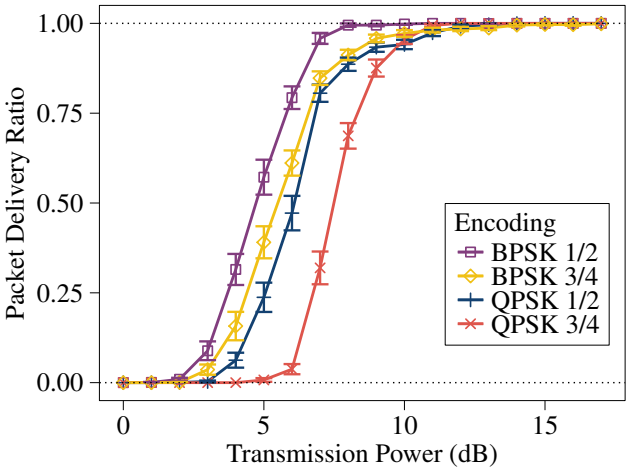

Figure 6: Packet delivery ratio of IEEE 802.11p packets, sent from a MK2 from Cohda Wireless. The packet size is 95 Byte.

employ the higher order modulations, where the magnitude of the subcarriers encodes information, since the receiver is currently limited to Phase Shift Keying (PSK) due to the lack of implementations of sophisticated channel estimation algorithms.

Since IEEE 802.11p operates on its own, dedicated frequency band at around 5.9 GHz, we do not assume that there are any considerable interference sources. As in the previous measurements we varied the transmission power between 18 values, spaced 1 dBm apart. The results of 30 measurement runs and the 95 % confidence intervals are shown in Figure 6. We can see that all four employed encodings are supported and the SDR approaches a PDR of 1 for higher transmission powers. Furthermore, the results are reasonable in the sense that higher bitrates suffer from higher packet loss as expected.

5. CONCLUSION

We presented an IEEE 802.11a/g/p receiver for GNU Radio and gave an overview of its structure and mode of operation. This is, to the best of our knowledge, the first GNU Radio receiver supporting Orthogonal Frequency Division Multiplexing (OFDM) at channel bandwidths of up to 20 MHz. The receiver is using the *Ettus USRP N210* Software Defined Radio (SDR) and does not require any change to the firmware of the Field-Programmable Gate Array (FPGA). To check the implementation and to verify its correctness, we made extensive interoperability tests with both consumer grade IEEE 802.11a/g WiFi cards as well as early IEEE 802.11p devices. Furthermore, we presented Packet Delivery Ratio (PDR) measurements, which showed that we can not just decode 20 MHz OFDM signals, but also that the performance of the receiver is reasonable. To make our work accessible to the community, we release the receiver under the GPLv3. This way, our GNU Radio OFDM receiver can serve as a basis for further experimentation, measurements, and research on signal processing algorithms. This also allows for reproducibility of conceptual studies and experiments, and all blocks of the receiver can be analyzed in more detail by fellow researchers.

6. REFERENCES

[1] Wireless Access in Vehicular Environments. Std 802.11p-2010, IEEE, July 2010.

[2] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Std 802.11-2012, IEEE, 2012.

[3] L. Chia-Horng. On the design of OFDM signal detection algorithms for hardware implementation. In *IEEE GLOBECOM 2003*, pages 596–599, San Francisco, CA, December 2003. IEEE.

[4] D. Eckhoff, C. Sommer, and F. Dressler. On the Necessity of Accurate IEEE 802.11p Models for IVC Protocol Simulation. In *IEEE VTC2012-Spring*, pages 1–5, Yokohama, Japan, May 2012. IEEE.

[5] P. Fuxjäger, A. Costantini, D. Valerio, P. Castiglione, G. Zacheo, T. Zemen, and F. Ricciato. IEEE 802.11p Transmission Using GNURadio. In *6th Karlsruhe Workshop on Software Radios (WSR)*, pages 1–4, Karlsruhe, Germany, March 2010.

[6] T. Hrycak, S. Das, G. Matz, and H. G. Feichtinger. Practical Estimation of Rapidly Varying Channels for OFDM Systems. *IEEE Transactions on Communications*, 59(11):3040–3048, November 2011.

[7] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly. WARP: A Flexible Platform for Clean-Slate Wireless Medium Access Protocol Design. *ACM SIGMOBILE Mobile Computing and Communications Review*, 12(1):56–58, January 2008.

[8] D.-W. Lim, S.-J. Heo, and J.-S. No. An Overview of Peak-to-Average Power Ratio Reduction Schemes for OFDM Signals. *Journal of Communications and Networks*, 11(3):229–239, June 2009.

[9] M. Morelli and U. Mengali. A Comparison of Pilot-Aided Channel Estimation Methods for OFDM Systems. *IEEE Transactions on Signal Processing*, 49(12):3065–3073, December 2001.

[10] T. Rondeau, N. McCarthy, and T. O'Shea. SIMD Programming in GNU Radio: Maintainable und User-Friendly Algorithm Optimization with VOLK. In *SDR 2012*, Brussels, Belgium, June 2012. Wireless Innovation Forum Europe.

[11] T. Schmidl and D. Cox. Robust frequency and timing synchronization for OFDM. *IEEE Transactions on Communications*, 45(12):1613–1621, 1997.

[12] C. Sommer, D. Eckhoff, R. German, and F. Dressler. A Computationally Inexpensive Empirical Model of IEEE 802.11p Radio Shadowing in Urban Environments. In *IEEE/IFIP WONS 2011*, pages 84–90, Bardonecchia, Italy, January 2011. IEEE.

[13] E. Sourour, H. El-Ghoroury, and D. McNeill. Frequency Offset Estimation and Correction in the IEEE 802.11a WLAN. In *IEEE VTC2004-Fall*, pages 4923–4927, Los Angeles, CA, September 2004. IEEE.

[14] J.-J. van de Beek, M. Sandell, and P. O. Borjesson. ML estimation of time and frequency offset in OFDM systems. *IEEE Transactions on Signal Processing*, 45(7):1800–1805, 1997.

[15] A. van Zelst and T. C. W. Schenk. Implementation of a MIMO OFDM-based wireless LAN system. *IEEE Transactions on Signal Processing*, 52(2):483–494, Feburary 2004.