MIMO-OFDM Spatial Multiplexing Technique Implementation for GNU Radio

Francesca Martelli, Alexander Kocian, Paolo Santi, Vanessa Gardellin
Institute of Informatics and Telematics
Italian National Research Council
Via G. Moruzzi 1, Pisa, Italy
{f.martelli}{a.kocian}{p.santi}{v.gardellin}@iit.cnr.it

ABSTRACT

Multiple-Input Multiple-Output (MIMO) is a wireless technology allowing a significant increasing of the throughput without the extension of the bandwidth but by means of the use of multiple antennas both in transmission and reception. Currently, Orthogonal Frequency Division Multiplexing (OFDM) is used in conjuction with MIMO to achieve better performance. With OFDM, instead of a single carrier, the main signal is split in a sub-set of independently modulated signals on orthogonal sub-carriers.

In this paper, we provide a description of our MIMONet SDR platform for network-level exploitation of MIMO technology. We present the implementation of our OFDM transceiver, developed following the structure of IEEE 802.11n standard and implementing one of the most powerful MIMO technique: *spatial multiplexing*. In this technique, two (or more) different data streams are simultaneously transmitted over two (or more) antennas. Starting from the original GNU Radio code, we modified and added blocks to achieve a complete implementation of MIMO-OFDM spatial multiplexing. We added some features, such as the concatenation of Forward Error Correction (FEC) in the packet construction, and the use of pilot sub-carriers for channel estimation. We also developed a new synchronization algorithm derived by extending the Van de Beek algorithm to the MIMO setting. We build the framework of the standard IEEE 802.11n. In particular, we put all the preambles needed for synchronization and channel estimation. We have also designed and implemented a fine grained signal-to-noise ratio (SNR) estimation, bit error rate (BER) and packet error rate (PER) computations, that allow us to evaluate the channel conditions and validate performance of the software implementation.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; B.4.1 [**Data Communications Devices**]: Receivers,Transmitters

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SRIF'14, August 18, 2014, Chicago, IL, USA.
Copyright 2014 ACM 978-1-4503-2995-8/14/08 ...$15.00.
http://dx.doi.org/10.1145/2627788.2627795.

Keywords

Software Defined Radios; MIMO testbed; GNU Radio; Spatial Multiplexing; OFDM.

1. INTRODUCTION

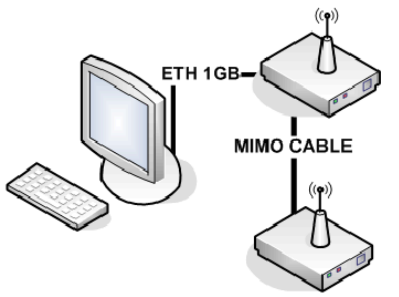
The increasing use of wireless technologies to support communication needs of individuals, companies, organizations, communities, etc., and the consequent increasing wireless traffic demand, will pose the problem of efficient bandwidth utilization at the forefront. Advanced wireless communication techniques such as cognitive radios [1] and multi-antenna systems [2] are then expected to be increasingly used by wireless network designers to improve bandwidth utilization.

A challenge to face when designing network-level approaches that make use of advanced communication technologies is their evaluation in a real environment. In fact, cross-layer design (down to the PHY layer) is often the key to fully exploit the potential of these technologies at network-level. Thus, the wireless network designer should be able to control and modify not only the upper layers of the network architecture, but also the PHY layer. Typically, commodity wireless networking cards do not allow access to the PHY layer, since the low level functionalities realized at the PHY layer such as digital signal processing (DSP) are implemented in hardware. Thus, specially designed hardware/software platforms such as software defined radios (SDRs) [3] must be used.

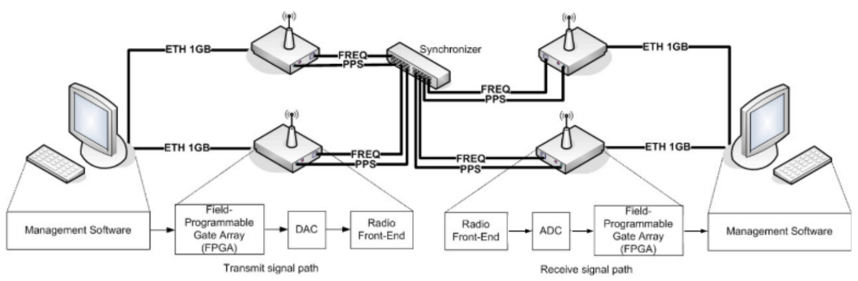
In SDRs, the entire transmitter and receiver chains are implemented in software, while a specialized hardware connected to the software modules is used as the radio front-end. This way, the wireless network designer gains full control of the PHY layer functionalities, enabling implementation and testing of the designed cross-layer, network-level protocols.

In this paper we present our implementation of MIMO-OFDM spatial multiplexing over GNU Radio, by introducing the framework for a complete IEEE 802.11n-compliant transceiver. A major contribution is the implementation of a new synchronization algorithm, derived by adapting the Van der Beek algorithm [4], originally proposed for SISO systems, to the MIMO setting at hand.

The paper is organized as follows: in Section 2 we present some related work about SDR implementations; Section 3 presents the details of the MIMO-OFDM spatial multiplexing transmitter and receiver. In Section 4 we show the results of some experiments we conducted, while Section 5 concludes the paper.



(a) A transmitter or receiver node with the MIMO cable



(b) A transmitter and a receiver node with the external GPS synchronizer

Figure 1: Hardware settings.

2. RELATED WORK

A number of SDR-based testbeds have been recently introduced in the literature to test network-level protocols. In terms of hardware, the most used SDR platforms are USRP produced by Ettus Research^{TM} [5], WARP [6] developed at Rice University, and Microsoft Sora [7]. One of the first SDR platforms for network MIMO systems is Hydra [8] developed at UT Austin, which is based on USRP hardware and GNU Radio software. Hydra provides a PHY layer implementation based on IEEE 802.11n, and a MAC layer based on the Distributed Coordination Function of IEEE 802.11. The Hydra platform has been used in [9] to evaluate performance of a rate adaptation technique for multi-antenna systems.

USRP hardware coupled with GNU Radio software is also used in a series of papers by Katabi et al. [10, 11, 12], where the performance of interference alignment and cancellation [10], of a random access protocol for MIMO networks [11], and of a rate adaptation technique for MIMO networks [12] are tested. WARP-based platforms have instead been used to test performance of beamforming in [13] and [14]. Finally, a Sora-based platform has been used in [15] to test performance of a spatial multiple access protocol for wireless LANs. In [16], the IEEE 802.11 a/g/p OFDM receiver implemented with GNURadio is presented.

3. MIMONET ARCHITECTURE

In this section we describe the MIMONet testbed architecture, starting with a description of the hardware platform in Section 3.1, and then presenting the software structure in Section 3.2. The transmitter and receiver implementations of the spatial multiplexing technique are illustrated in Sections 3.2.1 and 3.2.2.

3.1 Hardware

Two different hardware settings have been used to run experiments, illustrated in Figure 1. The devices common to the two settings are: (i) two PCs equipped with an Intel®-Core^{TM} i7-2600 at 3.4GHz and 8 GB of main memory; (ii) four USRPs, model N210 by Ettus Research^{TM} [5]; and (iii) Gigabit Ethernet cables to connect USRPs to the PC. USRPs are a family of computer-hosted hardware designed *with* GNU Radio. A single USRP unit consists of a motherboard and a daughterboard, where the latter is the radio front-end. The USRP N210 was chosen as the motherboard and the XCVR2450 as the daughterboard due to their computational capabilities, as well as their ability to operate in the frequency ranges of $[2.4; 2.5]$ GHz and $[4.9; 5.85]$ GHz. In fact, those are the frequencies used by the IEEE 802.11n standard for multiple antenna wireless networks [17].

In the first hardware setting (see Figure 1(a)), the connection between a PC and the two USRPs is through only one Ethernet cable, while the connection between the two USRPs is provided by a so-called MIMO cable. The MIMO cable is designed to guarantee synchronization between the two units, i.e., coherence in sampling clocks and local oscillators, which is essential for implementation of any MIMO technique. A MIMO cable can be used in a variety of ways. The so-called *shared Ethernet mode* was adopted in this instance, where one USRP is attached to the Ethernet cable and clock reference; time reference and data are communicated over the MIMO cable.

In the second setting, depicted in Figure 1(b), an external GPS synchronizer is used. Each USRP is connected to the PC by means of its own Gigabit Ethernet cable (*dual Ethernet mode*). USRPs connected to the same PC take two reference signals to synchronize their clocks and time: (i) a 10MHz reference to provide a single frequency reference for both devices; and (ii) a pulse-per-second (PPS) to synchronize the sample time across devices. The synchronizer used in our testbed is the EPSILON CLOCK EC20S by Spectracom Corporation [18] disciplined by GPS through Epsiltime$^{©}$ smart predictive slaving algorithm.

When a USRP is used as a transmitter, the management software on the PC produces a signal that is converted from digital to analog (DAC) so that it can be suitable for the radio front-end for subsequent transmission over the air. On the receiver side, the radio front-end receives the signal which is converted from analog to digital (ADC) and sent to the management software in the PC.

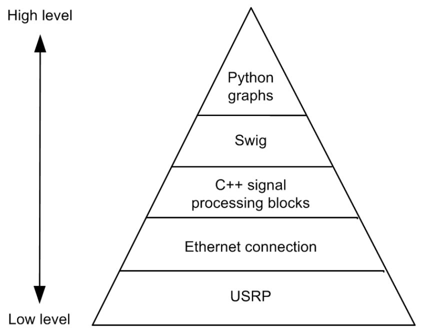


Figure 2: GNU Radio pyramid [19].

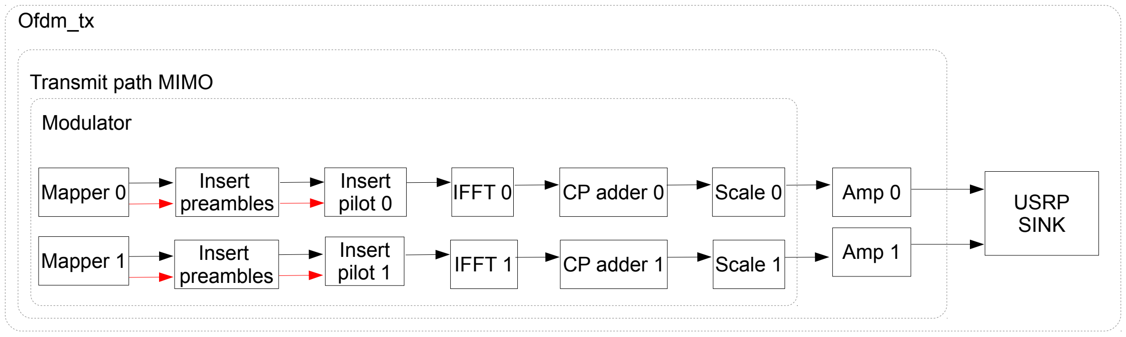


Figure 3: Transmitter architecture.

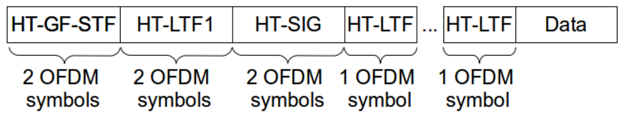


Figure 4: IEEE 802.11n preamble format, Greenfield mode.

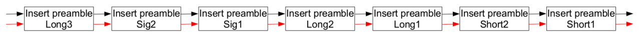


Figure 5: Insert preambles block.

3.2 Software

GNU Radio [19] is a free and open-source software for implementing SDRs. It is easy reconfigurable and hence it adapts to different network architectures and protocols. In fact, the GNU Radio architecture is based on a set of reconfigurable data sources, data sinks and signal processing blocks that are connected together to form transmitter and receiver chains. Figure 2 shows how GNU Radio works from the PHY layer up to the application layer. At the lower level are the USRP Hardware Drivers (UHDs) that provide an application programming interface (API) for USRPs and guarantee correct information exchange between USRPs and PCs through an Ethernet cable. Then, GNU Radio provides a library of signal processing blocks in C++, which implement various functionalities of the transmit and receive architectures. These C++ blocks can be used and combined by the final user through Python; in fact, GNU Radio provides a Python interface for each C++ block, as well as tools to connect the various blocks. This way, the simplified wrapper and interface generator (swig) allows C++ code to be used from Python.

The starting point of the implementation has been the original GNU Radio code, version 3.6.0. In the following sections, for each described block it is specified whether it is *new*, *modified* or *unchanged*. The developed code can be found in [20].

3.2.1 Transmitter

An overview of the implemented transmitter architecture is illustrated in Figure 3. Our approach has been to build a framework that could be easily expanded to be fully compliant to the IEEE 802.11n standard [21]. The main blocks composing the transmit path are the same already present in the original GNU Radio code, some of them with small modifications:

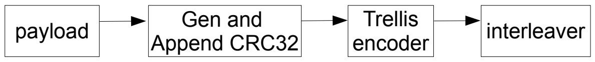


Figure 6: *make_packet* function.

- *Ofdm_tx.py* (**modified**): this is the top_block invoked to start transmission; it sets the USRP parameters by means of function *setup_usrp_sink()*; in the main function it creates the packets' payloads and calls the *send_pkt* function;
- *Transmit path mimo* (**modified**): it creates the modulator block and invokes *send_pkt*;
- *Modulator* (**modified**): it creates the transmit chain and in function *send_pkt* calls the *make_packet* function (see Figure 6). In this function, the packet's payload is passed to a function that computes and appends the CRC-32 at the end, and then the bit stream is convolutionally encoded and interleaved. With the output of the *make_packet* function, a message is built and inserted in the message queue, read from the mapper;
- *Mapper* (**modified**): it takes a string of bits belonging to a packet and maps to a vector of complex constellation points suitable for IFFT input, divided in the proper number of OFDM symbols;
- *Insert Preambles* (**modified**): it inserts a number of OFDM symbols of predetermined constellation points at the beginning of each packet and tags the first one as the first piece of the packet;
- *Insert Pilots* (**new**): it inserts pilot symbols in each data OFDM symbol and on HT-SIG preambles (not on other preambles);
- *IFFT* (unchanged): it computes reverse FFT;
- *Cyclic Prefix Adder* (**modified**): it adds a cyclic prefix before each piece of packet.

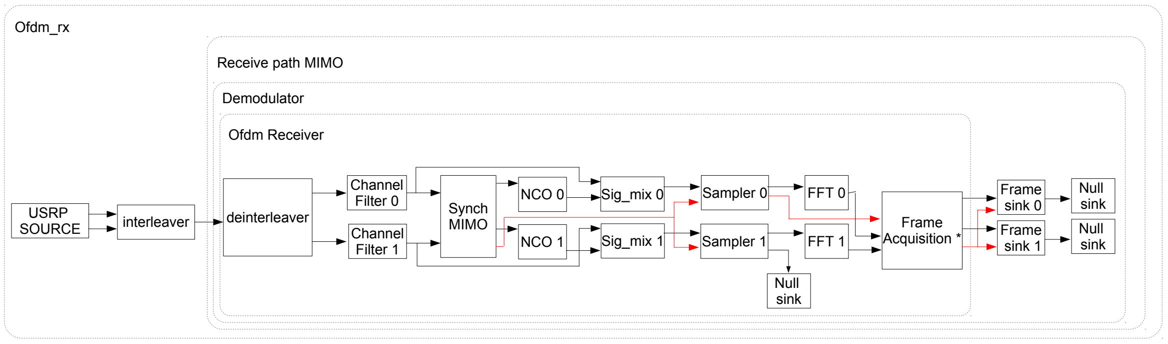


Figure 7: Receiver architecture. Lines in red represent the timing signal indicating the first symbol of a packet.

The *Insert Preamble* block is called seven times in order to accomodate all the seven preamble symbols required by the standard IEEE 802.11n. We chosen to implement the Greenfield format represented in Figure 4:

- HT-GF-STF is the short training field which takes 2 OFDM symbols; it is used for synchronization purposes;
- HT-LTF1 is the long training field which also takes 2 OFDM symbols, and the cyclic prefix in this field has a double length (32 symbols instead of 16); it is used for channel estimation;
- HT-SIG is the signal field which also takes 2 OFDM symbols, and contains information about the packet (such as modulation and coding scheme, length of the payload, type of FEC coding, etc.), since for our purposes, we used a predefined setting, we put in this field “fake” data, and the receiver will ignore what is inside these symbols. We plan to put proper data in this field in future implementations;
- HT-LTFs are additional long training fields, one for each extra dimension of the MIMO channel, in our case, since we implemented only a 2 by 2 MIMO channel, there is only one HT-LTF; as HT-LTF1, it is used by the receiver to estimate the MIMO channel.

The *Insert Pilots* block is a new block we added to put pilot symbols in predefined subcarriers. Pilots are used for channel estimation. In case of SISO transmission the pilot symbols are all “1”, while in case of 2 by 2 MIMO transmission, the second transmitter inserts the sequence “1, 1, -1, -1” in the four pilot subcarriers, as specified in [21].

The *Cyclic Prefix Adder* has been modified to produce HT-LTF1 in the proper way. As specified in [21], this preamble has a double length and also a double cyclic prefix; so we have introduced a mechanism that recognizes the two HT-LTF1 symbols coming through this block, and produces the proper double-length cyclic prefix.

3.2.2 Receiver

The receiver architecture framework is shown in Figure 7. The main blocks composing the receive path are conceptually the same of the original GNU Radio code, modified and connected to build a MIMO demodulator:

- *Ofdm_rx.py* (**modified**): this is the top_block invoked to start reception; it sets the USRP source parameters by means of function *setup_usrp_source()*; in the main function it calls the *rx_callback* function;
- *Receive path mimo* (**modified**): it creates the demodulator block;
- *Demodulator* (**modified**): it creates the receive chain and calls the *unmake_packet* function (see Figure 9) in which the packet’s payload is deinterleaved and Viterbi decoded; the decoded bit stream is passed to function *check_crc()* which extracts the last 32 bits and checks if they are equal to the CRC computed on the decoded bits; if so, the packet is marked as correct;
- *Ofdm Receiver* (**modified**): this block contains all physical functions for synchronization and equalization;
- *interleaver/deinterleaver* (unchanged) are used for multiplexing/demultiplexing the two streams coming from USRPs;
- *Channel filter* (unchanged) performs channel filtering, we have not modified this block;
- *Synch MIMO* (**modified**) performs PHY burst frame synchronization. This block exploits (i) path diversity in the MIMO channel; (ii) the periodic structure of the short training field in the IEEE 802.11n standard. The maximum-likelihood principle is used to extract the different time-offsets TO_1 and TO_2 and the common frequency-offset FO[1] based on the received signals RX_1 and RX_2, arising from $USRP_1$ and $USRP_2$, respectively. A *GNU Radio Companion* block diagram of the proposed synchronization scheme can be found in Figure 8. The implemented algorithm is an extension of Van de Beek *et al.* algorithm [4] to MIMO communications. From this block, only one timing signal is in output for subsequent blocks: given the short distances, we checked that the time offsets experienced by the two receivers are equal. The general case of different time offsets is left for future work.

[1]Notice that either MIMO cable or external synchronizer ensure identical clock frequencies in the USRPs.

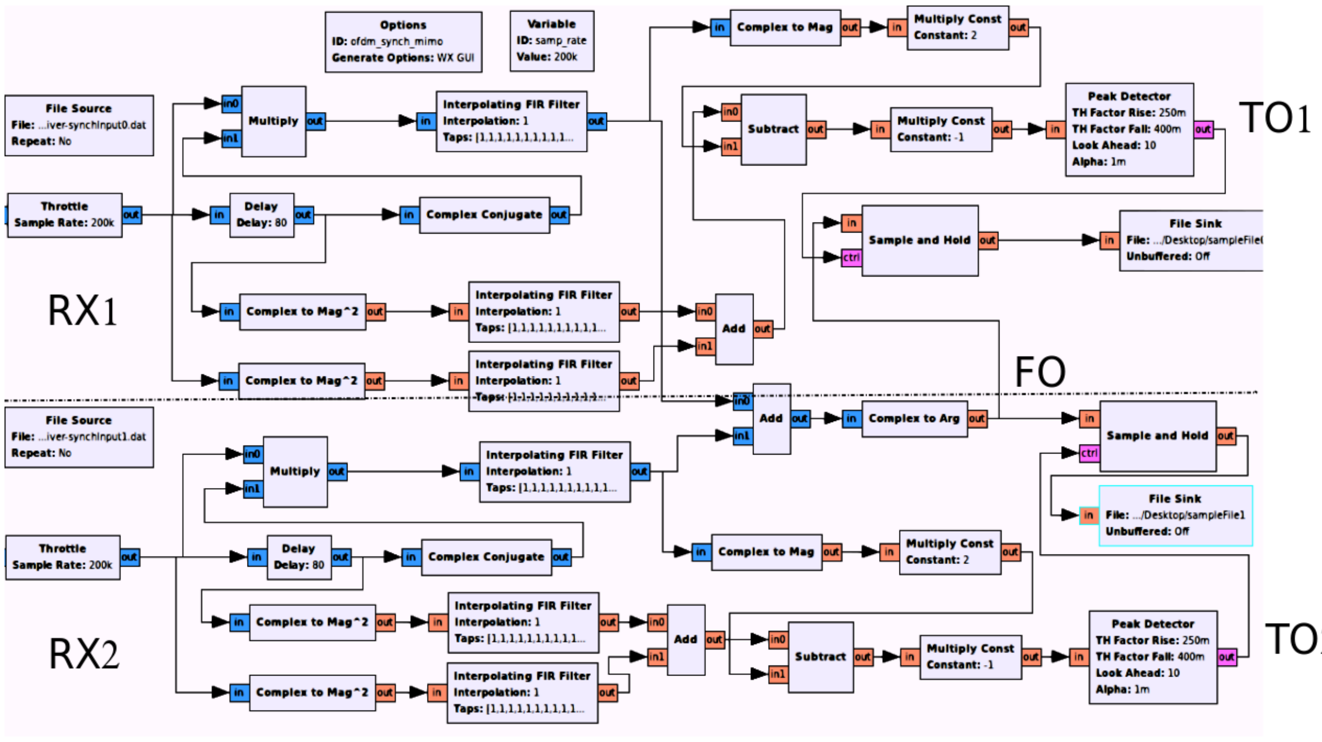


Figure 8: PHY Burst Frame Synchronization.

$$\begin{tikzcd}
\boxed{\text{deinterleaver}} \arrow[r] & \boxed{\text{Trellis Viterbi}} \arrow[r] & \boxed{\text{Check CRC32}} \arrow[r] & \boxed{\text{payload}}
\end{tikzcd}$$

Figure 9: *unmake_packet* **function.**

- *NCO* (numerically controlled oscillator) (unchanged) generates a signal proportional to the frequency error of the synchronization block;

- *Sig-mix* (unchanged) is a "multiply" block to derotate the signal;

- *Sampler* (modified) removes the cyclic prefixes and produces proper FFT-size blocks according to the timing signal output of the Synch MIMO block; we modify this block with respect to the original implementation for taking into account of the seven preambles and the proper manipulation of HT-LTF1 which has a double length cyclic prefix;

- *FFT* (unchanged) computes forward FFT;

- *Frame Acquisition* (**modified**) performs correlation, channel estimation and equalization: we implemented different versions of this block, one for SISO transmission and one for each MIMO technique; equalization is done by using pilots to correct the phase and amplitude distortion caused by the channel; the output are blocks of size equal to the number of occupied tones; it also propagates the preambles trigger so that the following blocks are able to demap only the payload OFDM symbols;

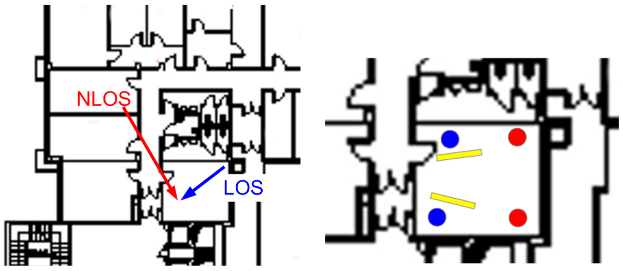
- *Frame sink* (**modified**) demaps the symbols into bits, packs them into packets and sends packets to the upper level queue sink where they are Viterbi decoded. In this block we inserted two functions to compute SNR: one function computes SNR on packet basis, namely considering the constellation points belonging to the same packet; the other function computes SNR on subcarrier basis, by producing a value every 100 constellation points (see Section 4.2).

4. PERFORMANCE EVALUATION

4.1 Experiments setup

To evaluate the performance of our implementation we conducted several experiments. We used three different scenarios, two of them by using the MIMO cable, while the third one with the external GPS synchronizer. The first two, depicted in Figure 10(a), are with the MIMO cable: one is in line-of-sight (LOS) and the other one in non-line-of-sight (NLOS). The third scenario, that will call close-non-line-of-sight (CNLOS) makes use of the external GPS synchronizer: it is depicted in Figure 10(b) where red dots represent the transmitters, and the blue ones represent the receivers: in this case, each receiver antenna is in line-of-sight with only one transmitter antenna, and obstacles (shown as yellow stripes in the figure) were inserted to have non-line-of-sight with the other transmitting antenna.

The following physical set up has been used: (i) center frequency is 2.45 GHz; (ii) sample rate is set to 200 KHz. (iii) FFT size is 64 and cyclic-prefix length is 16; (iv) number of occupied tones is 52 plus 4 pilots; (v) modulation is BPSK. We tuned the following physical parameters in order to have different SNR values: (i) USRP transmitter gain is



(a) LOS/NLOS (b) CNLOS

Figure 10: Experiments' scenarios.

in [10..40] in steps of 10; (ii) USRP receiver gain is in [10..70] in steps of 10; (iii) transmitter amplitude: 0.2 in LOS and CNLOS, and 0.5 in NLOS. Each pair (transmitter gain, receiver gain) corresponds to a single test run in which 2000 packets are sent. From the results, we removed some pairs, since with those no packet is recognized.

At application level, the tunable parameter is the packet payload size, which can be set to 96, 512 or 1032 bytes. With these packet sizes, after CRC computation and convolutional encoding, exactly all subcarriers of the packet are filled with useful data. Other settings are the following: (i) coding rate $R = 1/2$; (ii) FEC generator polynomial $G = (171, 133)_8$ (according to IEEE 802.11n standard).

To evaluate our testbed, two performance metrics are measured: SNR vs. BER (Bit Error Rate), and SNR vs. PER (Packet Error Rate).

4.2 SNR Profiles

By definition, SNR is given by the variance of the noise, say var($\mathbf{n}$). Hence, the symbol SNR is computed by estimating the noise variance around the constellation points and normalizing to the received signal power, namely $\gamma_i = \frac{1}{\text{var}(\mathbf{n})R}$ where i represents a sub-carrier of an OFDM symbol and R is the code rate.

The SNR per sub-carrier, instead, is given by $\gamma_{\text{bit}} = \frac{\gamma_i}{\log_2(A)}$ where A is the size of the alphabet of the used modulation (in BPSK $\log_2(A) = 1$).

The SNR is computed in two ways: on a packet basis, and on a sub-carrier basis. In the former, SNR is computed by considering all symbols belonging to the same packet; in the latter, instead, for each sub-carrier i, the average symbol SNR is computed across multiple packets. In particular, an SNR value is produced every 100 constellation points.

To build the profile of our channel, SNR values are divided into bins: each bin is 0.5 dB wide and the range considered is $[-2, 30]$ dB (i.e. 65 bins), where all values less than -2 dB (greater than 30 dB) are grouped in the first (last) bin. For each sub-carrier, every time a new SNR value is computed, it is assigned to the proper bin. At the end of each experiment run, we count the occurrences in each bin, say j, and in each sub-carrier, say i. Their probabilities of occurrences are

$$p_{i,j} = \frac{\theta_{i,j}}{\theta_i}$$

where θ_i represents the total number of SNR values computed for sub-carrier i, while $\theta_{i,j}$ is the number of occurrences in bin j. Then, the average SNR value observed in each sub-carrier is computed as

$$\gamma_i = \sum_{j=1}^{65} \overline{\gamma}_j p_{i,j}$$

where $\overline{\gamma}_j$ is the central SNR value of bin j.

Channel profiles are reported in Figure 11 for LOS, NLOS and CNLOS scenarios, respectively. The similarity of the shapes between the two users is proving that the two receivers experience a correlated channel, which is not the ideal setting for MIMO communications.

4.3 BER and PER

Bit Error Rate is computed in two ways: the classical *information* BER and the *transmission* BER. In the first case, bit errors are computed on the information data after Viterbi decoding; in the latter case, instead, bit errors are computed on the received data stream before decoding. Packet Error Rate is computed by counting the correct packets with respect to the received ones.

In these plots, SNR is computed on a packet-basis, and also in this case SNR values are divided in bins. The BER related to one bin is computed as the number of wrong bits divided by the total number of bits of the packets in that bin. Similarly, the PER related to one bin is computed as the number of wrong packets divided the number of packets in that bin.

In figures 12–14, we report the measured BER and PER with respect to the estimated SNR. In the BER plots, it is worth noticing that BER values, which in principle should be independent of packet size being a bit-level metric, are instead slightly affected by the packet size. This is due to the fact that SNR is computed on a per-packet basis, hence BER values derived from relatively small packets are computed based on relatively small sample sets, leading to a slight over-estimation of the experienced BER value. Notice that the relatively more spread results of BER vs. SNR obtained for relatively higher SNR values are due to the fact that the number of occurrences for relatively high SNR values are much less than for medium and low values – see Figure 15.

On the other hand, PER vs. SNR plots clearly show that longer packets lead to lower PER values, in accordance with intuition: for instance, referring to Figure 12, a PER of 0.8 is achieved at about 1.8dB for 96 bytes packets, while a 4 dB SNR is needed to achieve the same PER with 1032 bytes packet. However, when channel quality is good enough (≥ 5 dB), the difference in PER performance for different packet sizes becomes negligible.

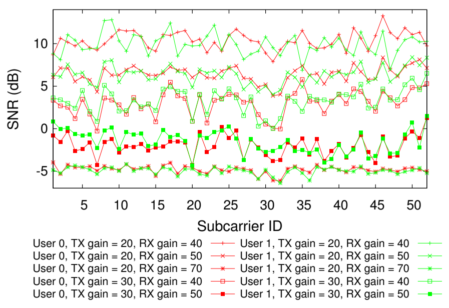
5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an implementation of MIMO-OFDM spatial multiplexing technique based on GNU Radio. We conducted some experiments to assess the expected behavior in terms of BER and PER.

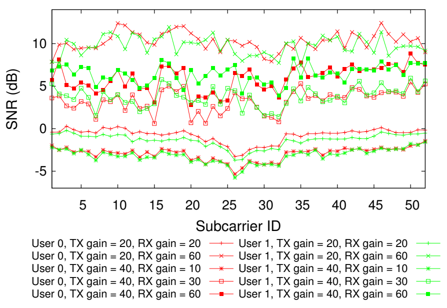
As future work, we are in the process of including support of more advanced MIMO techniques based on iterative algorithms, including distributed MIMO implementations.

6. REFERENCES

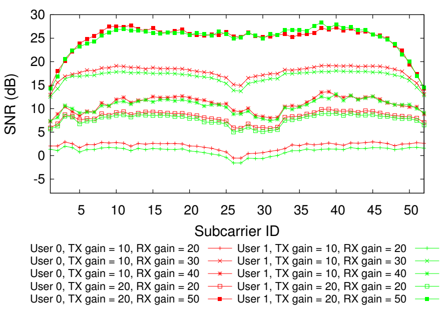
[1] I. F. Akyildiz, W. Y. Lee, M. C. Vuran, and S. Mohanty. A Survey on Spectrum Management in Cognitive Radio Networks. *IEEE Communications Magazine*, (4):40–48, 2008.



(a) LOS

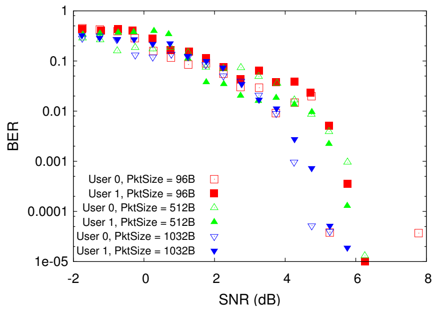


(b) NLOS

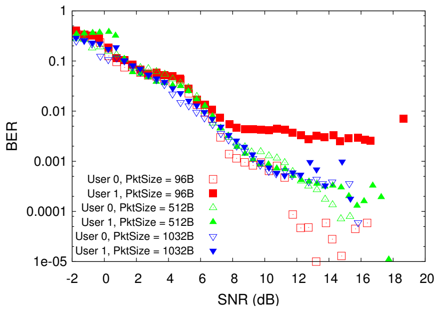


(c) CNLOS

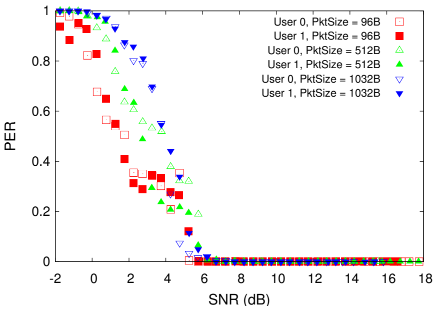
Figure 11: Channel profiles.



(a) Information BER

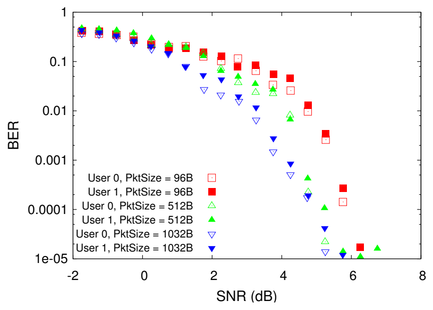


(b) Transmission BER

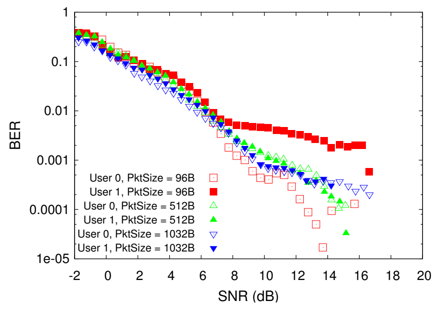


(c) PER

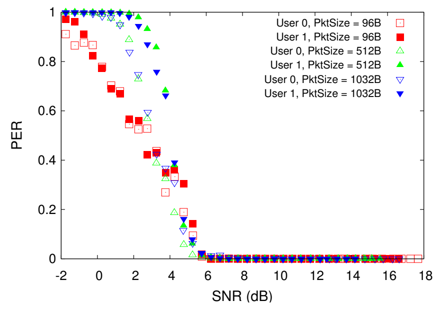
Figure 12: Computed metrics in LOS scenario.



(a) Information BER

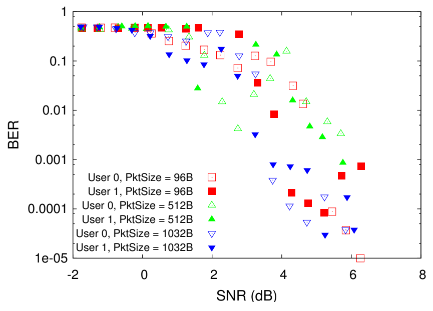


(b) Transmission BER

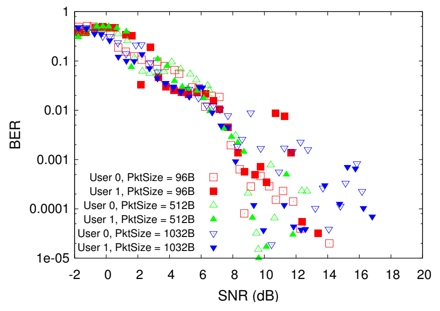


(c) PER

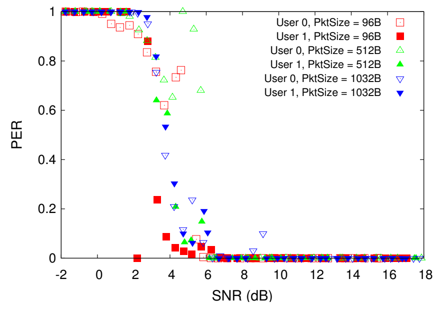
Figure 13: Computed metrics in NLOS scenario.



(a) Information BER

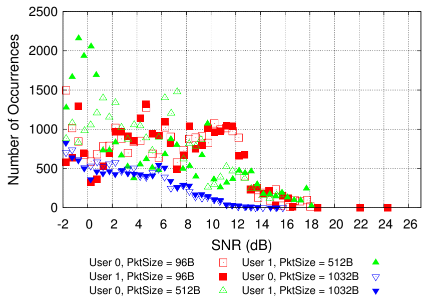


(b) Transmission BER

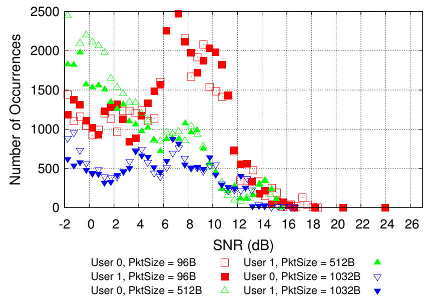


(c) PER

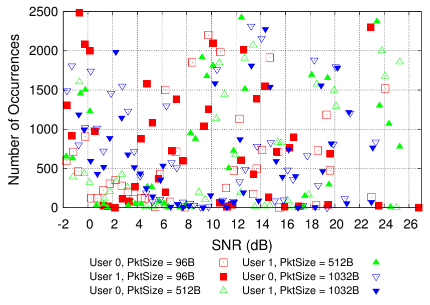
Figure 14: Computed metrics in CNLOS scenario.



(a) LOS



(b) NLOS



(c) CNLOS

Figure 15: SNR occurrences in the three scenarios.

[2] A. Sibille, C. Oestges, and A. Zanella. *MIMO: From Theory to Implementation*. Academic Press, 2010.

[3] M. Dillinger, K. Madani, and N. Alonistioti. *Software Defined Radio: Architectures, Systems and Functions*. John Wiley & Sons, 2003.

[4] 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, Jul 1997.

[5] EttusResearch. `http://www.ettus.com/`.

[6] WARP. `http://warp.rice.edu`.

[7] Microsoft Research. `http://research.microsoft.com/en-us/projects/sora/`.

[8] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. C. Daniels, W. Kim, R. W. Jr. Heath, and S. M. Nettles. *Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed*. 22–25 Apr., Dublin, Ireland 2007.

[9] W. Kim, O. Khan, K.T. Truong, S.H. Choi, R. Grant, H.K. Wright, K. Mandke, R.C. Daniels, R.W. Heath, and S.M. Nettles. An experimental evaluation of rate adaptation for multi-antenna systems. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 2313–2321, 19–25 Apr., Rio De Janeiro, Brazil, 2009.

[10] S. Gollakota, S. D. Perli, and D. Katabi. Interference Alignment and Cancellation. In *Proc. ACM SIGCOMM*, pages 159–170, 2009.

[11] K. C. Lin, S. Gollakota, and D Katabi. Random Access Heterogeneous MIMO Networks. In *Proc. ACM SIGCOMM*, pages 146–157, 2011.

[12] W.L. Shen, Y.C. Tung, K. C. Lee, K. C. Lin, S. Gollakota D., Katabi, and M. S. Chen. Rate Adaptation for 802.11 Multiuser MIMO Networks. In *Proc. ACM Mobicom*, 2012.

[13] H. Yu, L. Zhong, A. Subharwal, and D. Kao. Beamforming on Mobile Devices: A First Study. In *Proc. ACM Mobicom*, 2011.

[14] E. Aryafar, N. Anand, T. Salonidis, and E. Knightly. Design and Experimental Evaluation of Multi-User Beamforming in Wireless LANs. In *Proc. ACM Mobicom*, 2010.

[15] K. Tan, H. Liu, J. Fang, W. Wang, J. Zhang, M. Chen, and G. M. Voelker. SAM: Enabling Practical Spatial Multiple Access in Wireless LAN. In *Proc. ACM Mobicom*, 2009.

[16] Bastian Bloessl, Michele Segata, Christoph Sommer, and Falko Dressler. An IEEE 802.11a/g/p OFDM Receiver for GNU Radio. In *ACM SIGCOMM 2013, 2nd ACM SIGCOMM Workshop of Software Radio Implementation Forum (SRIF 2013)*, pages 9–16, Hong Kong, China, August 2013. ACM.

[17] 802.11-2012 - IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications, March 2012.

[18] Spectracom. `http://www.spectracomcorp.com/ProductsServices/TimingSynchronization`.

[19] GNURadio. `http://gnuradio.org/`.

[20] MIMONet Project. `http://mimonet.iit.cnr.it/`.

[21] 802.11n. IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput, 2009.