Software Defined Radio Implementation of a Two-way Relay Network with Digital Network Coding

Dmitry Kramarev, Yi Hong, and Emanuele Viterbo
Department of Electrical and Computer Systems Engineering,
Monash University, Australia
Email: {dmitry.kramarev, yi.hong, emanuele.viterbo}@monash.edu

***Abstract*—Network coding is a technology which has the potential to increase network throughput beyond existing standards based on routing. Despite the fact, that the theoretical understanding is mature, there have been only a few papers on implementation of network coding and demonstration of a working testbed. This paper presents the implementation of a two-way relay network with digital network coding. Unlike previous work, where the testbeds are implemented on custom hardware, we implement the testbed on GNU Radio, an open-source software defined radio platform. In this paper we discuss the implementation issues and the ways to overcome the hardware imperfections and software inadequacies of the GNU Radio platform. Using our testbed we measure the throughput of the system in an indoor environment. The experimental results show that the network coding outperforms the traditional routing as predicted by the theoretical analysis.**

***Index Terms*—Software-defined radio, GNU radio, network coding, two-way relay network, testbed, network coding implementation.**

I. Introduction

In the simplest two-way relay network (TWRN), two terminals A and B exchange information via relay R. The relay is needed when the terminals are unable to communicate directly due to large distance, low SNR or major obstacles. The traditional scheduling (TS) scheme implements such communication in four time slots, as shown in Figure 1a. First, A transmits a message m_1 to relay R; second, relay R forwards m_1 to B. In the third and fourth time slots, terminal B transmits a message m_2 to relay R, and R forwards m_2 to A, respectively. With *digital network coding* (DNC), it is possible to increase the throughput of the network by 33% via reducing the number of time slots to three, as shown in (Figure 1b). In the first and second time slots, each terminal transmits its message m_1 and m_2 to the relay, then in the third slot, relay R broadcasts the XOR of the two received messages $m_1 \oplus m_2$ to both terminals. In this case, a terminal is able to recover the information based on the received combination and its own transmitted message. DNC was first proposed in [1] and [2]. Physical-layer network coding (PNC) proposed in [3] allows for further reduction of the number of time slots to two, by superposing the first and the second time slots. The design of a PNC testbed will be investigated in our future work. In [4] Louie *et al.* compare the performance

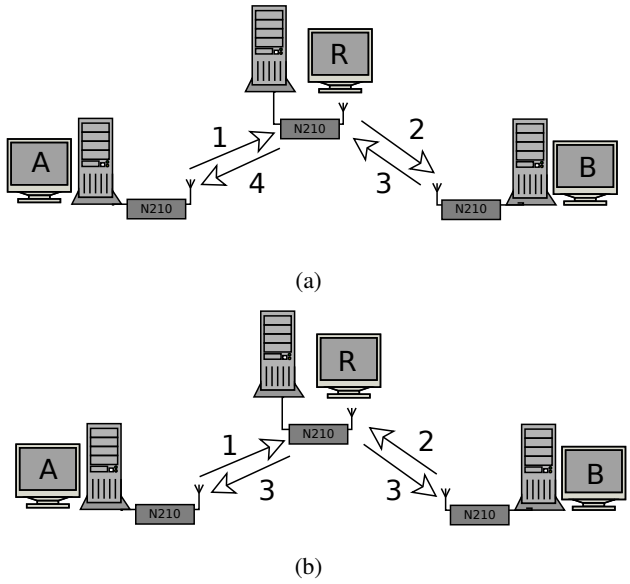

Fig. 1: Two-way relay network with (a) the four-hop TS, and (b) three-hop DNC scheme

of the two-, three-, and four-hop TWRNs. The paper also provides an extensive survey of the recent theoretical work on network coding for two-way relay networks.

Although a significant number of theoretical results have been verified by simulations, limited results are available on the implementation of TWRN testbeds on real-time hardware. Only a few implementations of DNC in TWRN or larger relay networks are reported in [2], [5] and [6]. For example, [5] demonstrates essential reduction of time required to exchange video files between smartphones, in case DNC is employed. In [2], Katti *et al.* present a new architecture for wireless mesh networks (COPE), which improves the throughput of wireless networks by forwarding DNC mixed packets. The testbed consists of 20 wireless nodes located on two floors of a building. In addition to demonstrating advantages of DNC, the authors also investigate the implementation issues of a large network, such as the congestions control, traffic patterns

978-1-4799-3354-9/14/$31.00 ©2014 IEEE

and optimal routing. Another example of implementation of a multi-hop relay network with MIMO network coding is provided in [6]. The MIMO TWRN implements data exchange within two time slots by transmitting messages from both the terminals at the same time. Both messages are fully recovered by exploiting multiuser antenna diversity of MIMO system with space-time coding. Then the messages are XORed and broadcasted in the second time slot.

All the testbeds, mentioned above are implemented on specific dedicated hardware. Such hardware provides only a few opportunities to easily modify the important physical layer functions and parameters such as modulation type, frequency range etc., or extend an experiment. In contrast, software-defined radio (SDR) enables to implement in software all the baseband signal processing algorithms such as modulation, demodulation, filtering, and synchronization [7] - [9]. The hardware for SDR is based on an inexpensive simple RF front-end, followed by analog-to-digital and digital-to-analog converters. SDR introduces significant flexibility and programmability, compared to traditional communication systems.

Taking into account this gap between theory and practice we see the interest for implementation of a two-way relay network testbed using SDR. As a first step towards the implementation of PNC in TWRN, we design a three-hop DNC relay network using GNU Radio, an open-source SDR platform. GNU Radio provides signal processing blocks to implement software radios using external low-cost RF hardware. GNU Radio applications, written in the Python language consist of signal processing blocks implemented in C++. Using these blocks and creating new custom blocks, we are able to implement real-time radio systems in a simple-to-use environment [10]. In this paper we discuss the implementation challenges in GNU Radio and analyze the performance of our testbed.

The rest of this paper is organized as follows. In Section II we describe some implementation aspects and ways we have used to resolve them. Section III provides detail description of two-way relay network schemes for our testbed. The experimental environment and measurements results are provided in Section IV. Finally, Section V concludes this paper and discusses directions for future work.

II. Implementation Aspects of the Half-Duplex Communication

In this section we discuss implementation aspects for the half-duplex communication used in the prototype testbed. We start with the discussion of hardware and software platforms, and we then present design aspects for half-duplex packet-switched communications in GNU Radio. Also, we summarize all the parameters of the different blocks involved in the design and describe the used packet format.

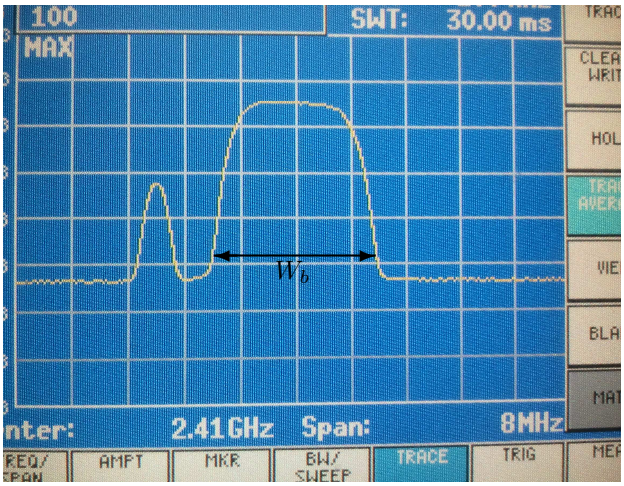

Fig. 2: Bandpass signal spectrum with IF processing

A. Hardware and Software

The hardware platform is based on Universal Software Radio Peripheral (USRP) N210 from Ettus Research [11]. Each USRP is equipped with an half-duplex XCVR2450 daughterboard operating in 2.4-2.5 GHz and 4.9-5.9 GHz dual band. The GigE interface connects the USRP and the host PC, and allows for 25 MSPS (16-bit) data transfer. We set the carrier frequency at 2.41 GHz and the sampling rate at 6.25 Msps (complex samples) at the USRP source and sink. A terminal is implemented with an USRP N210 connected to an individual PC running GNU Radio. The software is based upon GNU Radio 3.6.2.

B. Mitigation of the USRP Transmission noise

Measurements with a spectrum analyzer indicate, that RFs of the daughterboards have a significant DC component, volume of which largely varies from one daughterboard to another. This phenomenon causes TX gain control parameters to be unrepresentative and the TX SNR uncontrollable. As such, swapping of two terminals can cause significant degradation of the system throughput, especially with lower TX gains.

In order to mitigate this TX noise impact we have employed additional digital up/down-conversion of the baseband signal to a small intermediate frequency (IF). The intermediate frequency f_I is chosen such that $f_I > W_b/2$, where W_b is the bandwidth of the transmitted signal. The baseband signal is first up-converted to IF on the TX side using the onboard FPGA. Then, we choose the carrier frequency f_c in the USRP sink such that:

$$f_c = f_d - f_I,$$

where f_d is the actual transmitted carrier frequency. The spectrum of the resulting bandpass signal after IF processing is illustrated in Figure 2. Similarly, on the receiver side, the bandpass signal is first down-converted to IF f_I by the USRP's daughterboard and further digital down-conversion to baseband is performed by the FPGA on the motherboard. As a result, the unwanted noisy component is shifted to frequency

$f = -f_I < W_b/2$ and suppressed by the pulse-shaping filter before demodulation. Thus, our IF processing improves the TX SNR, and in turn increases the effective range of TX gains. This makes manual calibration of the TX power unnecessary.

C. Half-Duplex Packet Switching in GNU Radio

The current GNU Radio software architecture is primarily aimed at processing continuous data streams. Packet switching is therefore difficult to implement in the GNU Radio receiver due to the bursty nature of the incoming stream. In the current version of GNU Radio blocks are not clocked and operate under the control of a scheduler, which attempts to provide a steady stream of input data to each signal processing block. To the best of our knowledge there is no half-duplex packet switching implementations which would be satisfactory to adopt in our project in GNU Radio.

We employ the transmit-on-receive approach, i.e., a terminal transmits its packet upon reception from the other terminal. This approach identifies the terminal starting communication as the *master* and the other as the *slave*.

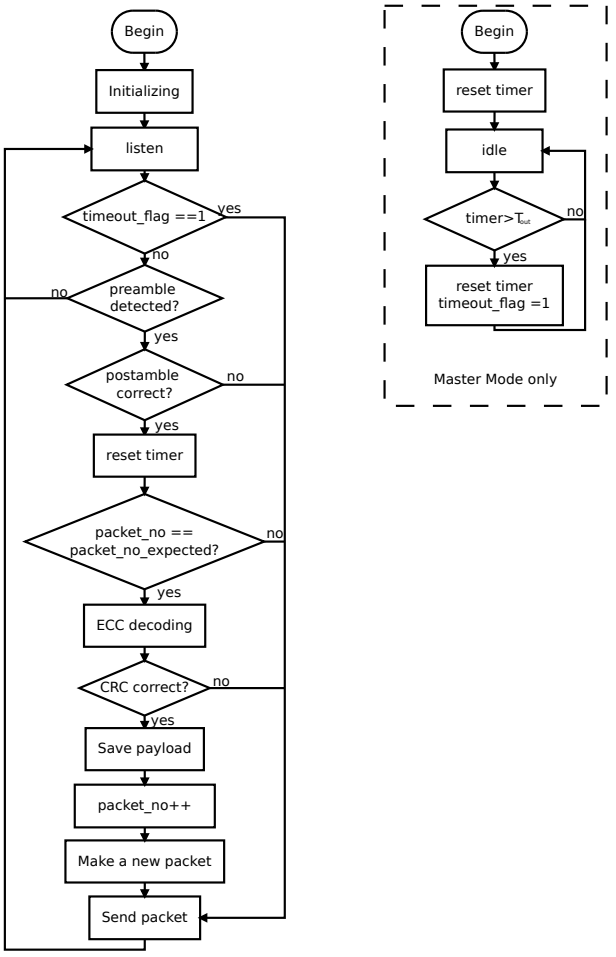

Fig. 3: A terminal operation flowchart

Packets are equipped with sequence number which is used as an ACK/NACK mechanism. The transmitter decides if the transmitted packet has been received successfully by the receiver, based on the reply packet number. Reception of a corrupted packet will cause the retransmission of the previous packet, which in turn causes the retransmission of the corrupted packet. The flowchart of this half-duplex packet switching protocol is presented in Figure 3. Upon reception from the other terminal, the terminal compares the *packet_number* of the received packet with the *expected_packet_number*. If these numbers are equal, the terminal increments its own *packet_number* and in turn transmits the next packet, otherwise re-transmits the previous packet. The master retransmits on timeout in case the communication is lost. The slave starts transmitting only after receiving a packet from the master.

We implemented this half-duplex packet switching protocol in Python and included it into GNU Radio Companion (GRC) flow graphs as a custom block, as shown in Figures 8,9. The GRC flow graphs are described in details in the Appendix.

D. Block Parameters and Packet Format

Table I contains parameters for the communication between two USRP devices.

Block	Parameter	Value
USRP	USRP Sample Rate	6.25M
	Antenna	J1
	Rx Gain	0 dB
Mod./Demod.	Modulation Scheme	DQPSK (Gray code)
	Samples per symbol	2
	Excess BW	0.3
	AGC	Default
Half-duplex	Packet Length	4096 bytes
	Correlator Tolerance	9
	Master Timeout T_{out}	0.05 s.
	Guard time T_g	0.006 s.
	ECC	(10,8) Reed-Solomon, CRC Adler-32

TABLE I: Parameters of design and experiments

The packet format is presented in Figure 4. A packet begins with 128 bytes of dummy data, followed by 8-bytes access code. The dummy data is sent at the beginning of the packet and is used for carrier, phase and timing recovery on the receiver side. An access code is needed for addressing the destination terminal and for frame synchronization. The next part of the packet consists of one byte packet

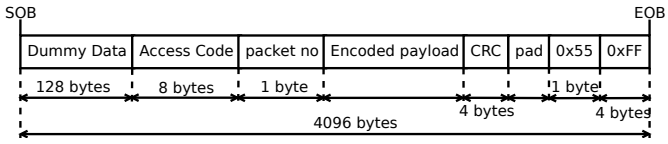

Fig. 4: Packet format

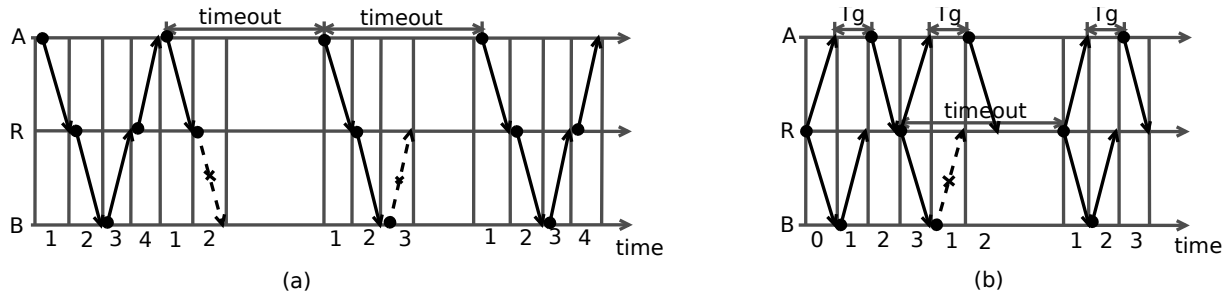

Fig. 5: Network synchronization: (a) the four-hop scheme, (b) the three-hop scheme

sequence number followed by the encoded payload, CRC checksum, the pad and the postamble. For ECC we employ the (10, 8) Reed-Solomon code over $GF(2^8)$, available in the Python extension module [12]. CRC32 error detection algorithm is also added for post-ECC data verification. The postamble indicates the end of the packet and is represented by one byte with value $0x55$. In the presence of timing recovery errors, an incorrectly received postamble indicates insertion/deletion errors. The length of the pad varies according to the codeword length of the used ECC in order to keep the required packet size. Finally, four extra bytes of $0xFF$ are added to the tail of the packet as a guard interval.

III. Two-Way Relay Network Schemes

In this section we provide the detailed description of the two-way relay network schemes implemented in the testbed, namely the four-hop scheme and three-hop scheme based on DNC. The communication between a terminal and the relay is based on the half-duplex packet switching protocol described in the previous section.

A. The Four-hop Scheme Based on the Traditional Scheduling

In the four-hop scheme each terminal has its own *tx_access_code* and *rx_access_code*. The relay has two pairs of *tx_access_code* and *rx_access_code*, such that:

$$\begin{aligned} tx\_access\_code_{R0} &= rx\_access\_code_A, \\ tx\_access\_code_{R1} &= rx\_access\_code_B, \\ rx\_access\_code_{R0} &= tx\_access\_code_A, \\ rx\_access\_code_{R1} &= tx\_access\_code_B. \end{aligned}$$

We set one of the terminals into the master mode to perform the network synchronization. Figure 5 shows the network synchronization scheme when terminal A is the master. Terminal A initiates the communication by sending the first message m_0 addressed to R. Upon receiving m_0, R verifies the packet. If the packet is correct, the preamble of m_0 is changed from $tx\_access\_code_A$ to $rx\_access\_code_B$ and this modified message m_0' is forwarded to terminal B. After being received m_0' is verified and processed according to the procedure described in Figure 3. If the reception is successful, B sends its message m_1 to R and this message is forwarded to A in the same manner. Upon successful reception the master transmits the next message and so on. In the case, a packet is lost or corrupted in any hop, time-out occurs, and the master retransmits the previous message.

B. The Three-hop Scheme Based on DNC

In the three-hop scheme each terminal has its own *tx_access_code* but shares the *rx_access_code*. The relay has one *tx_access_code* and two *rx_access_code*, such that:

$$\begin{aligned} tx\_access\_code_{R0} = rx\_access\_code_A = \\ rx\_access\_code_B, \\ rx\_access\_code_{R0} = tx\_access\_code_A, \\ rx\_access\_code_{R1} = tx\_access\_code_B; \end{aligned}$$

which allows relay R to communicate with both terminals at the same time. Unlike the four-hop scheme, the network synchronization here is performed by the relay, as shown in Figure 5. At the beginning R broadcasts a packet containing dummy data. Upon reception terminal A transmits its message immediately (hop 1), while B transmits after a delay T_g (hop 2). If the messages from both the terminals are received successfully, R performs XOR of the two messages payload data and broadcasts the resulting message (hop 3). After successful reception of the broadcasted message, terminals A and B transmit the next messages and so on. In case any packet is lost at any hop, time-out at R happens and R re-transmits the previous XORed packet.

IV. Experimental Results

In this section, we demonstrate the performance of the testbed via a series of experiments in the indoor environment. The locations of the terminals and relay are illustrated in Figure 6. Each terminal is located in a separate room such that the distance between terminal A and the relay is about 9 meters, the distance between terminal B and the relay is about 12 meters, and the distance between two terminals is about 18 meters. The data exchanged between the terminals are randomly generated. The payload scrambling is implemented to avoid pattern-dependencies. Figure 7 shows a measured

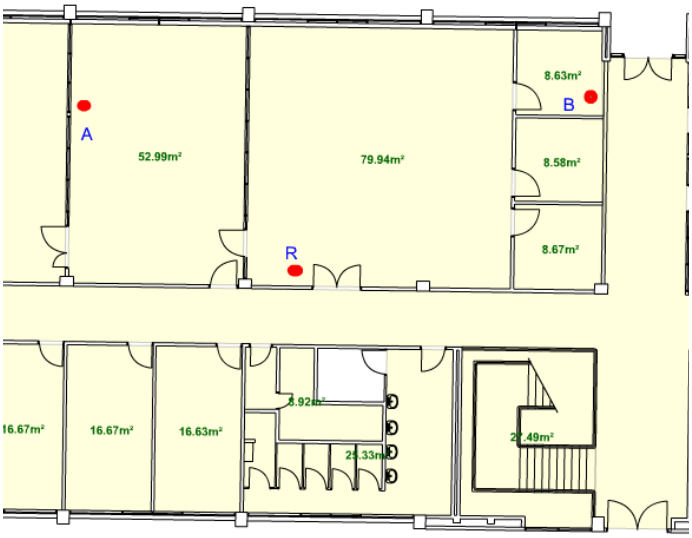

Fig. 6: Locations of the terminals in the indoor environment

average data throughput of the TWRN per direction vs. the TX gain. With TX gain $20dB$ the average data throughput per directions achieves about 504 kbps and 664 kbps for TS and DNC, respectively. The lower performance of the TS scheme at 25 dB may be due to power amplifier nonlinearity. Thus, the throughput of the DNC scheme is about 30% higher than that of the TS scheme. Also, the figure demonstrates the advantage of the TWRN communication over direct communication, when the distance between the terminals is large. Direct communication between two terminals without relay is only possible with the highest TX gain (25 dB). Despite the gain, the throughput is still lower than that of the TWRN due to presence of uncorrectable errors, insertion/deletion errors and packet losses.

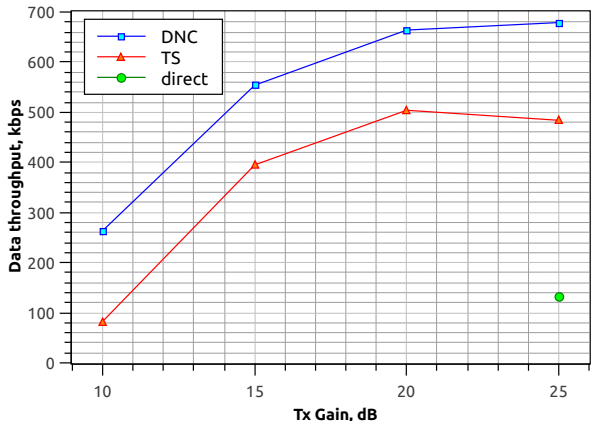

Fig. 7: Measured average data throughput per direction vs. TX gain

V. Conclusion and Future Work

In this paper we have presented a prototype of the two-way relay network implemented in GNU Radio. This prototype supports both the four-hop traditional scheduling protocol and the three-hop digital network coding. In order to overcome the negative impact of the hardware imperfections and software inadequacies we have implemented a few signal processing algorithms on the FPGA of the USRP devices, such as the intermediate frequency processing. Additionally, we have designed the GNU Radio blocks for the half-duplex packet switching.

We then have investigated the performance of the testbed in terms of throughput. The actual throughput of the testbed using digital network coding is about 30% higher than that of the traditional scheduling. This result agrees with the theoretical performance. At the same time, the throughput of the direct communication without the relay is significantly lower than that of the TWRN.

Finally, our future research will focus on implementation of the physical-layer network coding for the two-way relay network.

Acknowledgement

This work was performed at the Monash Software Defined Telecommunications Lab. This work was partially supported by Australian Federal and Victoria State Governments and the Australian Research Council through the ICT Centre of Excellence program, National ICT Australia (NICTA).

Appendix

GNU Radio Companion (GRC) generates a signal processing Python script from the signal flow graph. We create a GRC flow graph composed of standard and custom blocks. The flow graphs representing a terminal and the relay are shown in Figures 8 - 10. Figure 8 shows a flow graph of a terminal for the four-hop scheme. Figure 9 shows a flow graph of a terminal adapted for the three-hop scheme. The skip head block is used to skip the dummy data from the first message sent for synchronization purpose. The XOR block connected with output *rx_out* of *Half-Duplex* block and the source file is used to recover the message from the network-coded received packet.

The GRC flow graph of the relay for both schemes is shown in Figure 10, where both the schemes are implemented in Python in the block *Half-Duplex-R*.

References

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204-1216, July 2000.

[2] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, pp. 497-510, June 2008.

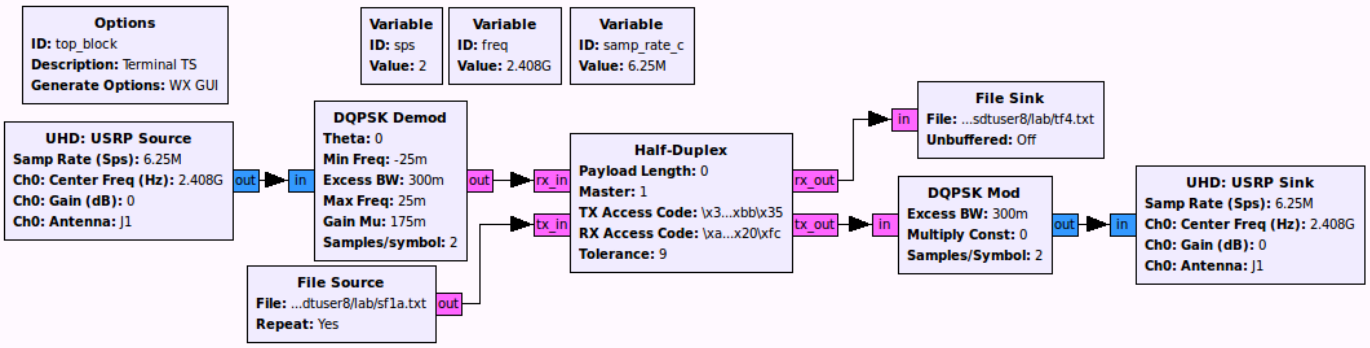

Fig. 8: GRC flow graph of the terminal with the four-hop scheme

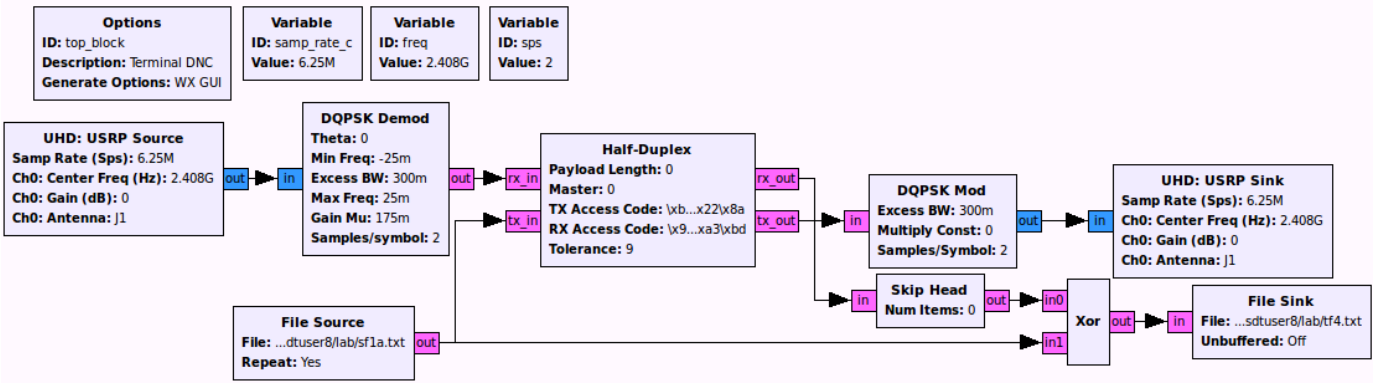

Fig. 9: GRC flow graph of the terminal with the three-hop scheme

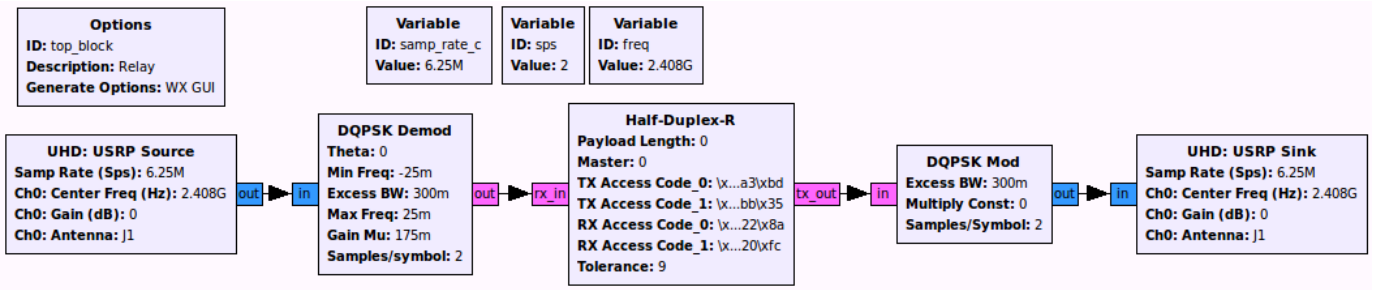

Fig. 10: GRC flow graph of the relay

[3] S. Zhang, S. C. Liew, and P. P. Lam, "Physical-Layer Network Coding," in *Proceedings of the 12th annual international conference on Mobile computing and networking (MobiCom 06)*, pp. 358-365, LA, USA, Sept. 2006

[4] R. H. Y. Louie, Y. Li, and B. Vucetic, "Practical Physical Layer Network Coding for Two-Way Relay Channels: Performance Analysis and Comparison," *IEEE Trans. on Wireless Comm.*, vol. 9, no. 2, pp. 764-777, Feb. 2010.

[5] H. Seferoglu, L. Keller, B. Cici, A. Le, and A. Markopoulou, "Cooperative Video Streaming on Smartphones," *Proceedings of the 49th Annual Allerton Conference*, pp. 220-227, Illinois, USA, 2011.

[6] K. Mizutani, Y. Kida, T. Miyamoto, K. Sakaguchi, and K. Araki, "Realization of TDD Two-way Multi-hop Relay Network with MIMO Network Coding," *Proceedings of the 6th Int. ICST Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Osaka, Japan, 2011.

[7] A. Abidi, "The Path to the Software-Defined Radio Receiver," *IEEE J. Solid-St. Circ.,* vol. 42, no. 5, pp. 954-966, May 2007.

[8] J. Mitola, "The Software Radio Architecture," *IEEE Commun. Mag.,* vol. 33, no. 5, pp. 26-38, May 1995.

[9] C. Johnson, W. Sethares, and A. Klein, "Software Receiver Design," Cambridge University Press, 2011.

[10] GNU Radio: http://gnuradio.org/

[11] Ettus Research USRP N210, https://www.ettus.com/product/details/UN210-KIT

[12] Reed-Solomon ECC Python Extension Module, http://hathawaymix.org/Software/ReedSolomon/