Wireless Networks In-the-Loop: Software Radio as the Enabler

Jens P. Elsner, Martin Braun, Stefan Nagel, Kshama Nagaraj and Friedrich K. Jondral
Universität Karlsruhe (TH), Germany, {elsner, braun, nagel, nagaraj, fj}@int.uni-karlsruhe.de

***Abstract*—A software architecture to rapidly develop and test radio networks in real and physical environments is proposed. Radio network terminals are developed in software and run on generic hardware to maximize reconfigurability. Due to the software nature of the radio terminals, radio networks can be simulated in a virtual environment, where physical channels are emulated by software entities. Without any changes to the code base, the same waveform can also be run in a real, physical environment. This feature is used to rapidly switch between real and virtual networks, thus bridging the gap between simulation and physical reality. Aspects of the proposed system are implemented and demonstrated with the GNU Software Radio framework.**

I. Introduction

Software radio development usually follows an iterative top-down pattern from a generic Platform Independent Model (PIM) of a waveform to platform specific model (PSM) and executable code. Such a model-based development approach leads to portable waveform design and reduces the design cycle duration of SDR applications and hence the time-to-market [1].

Traditionally, the heterogeneous RF environment which causes interference between systems is not explicitly included in the design and testing process: point-to-point influences such as noise, Doppler shift or fading are usually modeled at base band and medium access is modeled separately and for a single technology only. Radio front-end hardware design and operating environment are rarely modeled directly. It is, however, advantageous to take these influences into consideration during development of a waveform PSM and PIM. This is especially true for the development of adaptive or ”cognitive” waveforms, which need to coexist with other systems in diverse RF environments, and also for portable waveforms developed for SDR platforms with differing RF characteristics.

Radio front-end hardware design and operating environment can have a significant impact on signal processing algorithms. To quantify the effects on various protocol layers, e.g., on throughput and delay, a simulation environment which accurately models the physical realities is needed. Furthermore, the waveform should be able to run without changes to the code base in the simulator and on SDR hardware, enabling fast switching between loop and field testing of wireless networks. Such a simulation architecture is proposed and demonstrated with the GNU Radio framework.

Related work

A myriad of radio simulation tools already exist. In a recent survey [2], three commercial and eight open source network simulators are listed, as well as many more tools for wave propagation simulations. So why create another one? The wireless networks in-the-loop design discussed here differs from most available simulators in several aspects. First of all, it is not a simulator in the typical sense; a more precise description is *testbed for software radio terminals*. The main advantage of loop development is that the test code and the final software radio are identical. At the end of a development cycle, a fully functional and thoroughly tested code base has been established. This eliminates two major disadvantages of most network simulators: first, no time is lost on creating a model, as the code is reused verbatim for the final product. Next, there is no uncertainty about how well the simplifications introduced by the model affect the performance of the final implementation, since the development cycle - and, in particular, the switch from a virtual to a real environment - introduces a smooth transition from preliminary tests to the finished radio. Of course, such an approach bears disadvantages as well. For every test, the code for an entire terminal must be written. Even if this can be accomplished fairly quickly using development frameworks such as GNU Radio, it is an unnecessary task if one wants to, e.g., evaluate the bit error rate of simple point-to-point communication links in a Rayleigh channel. In simple cases like this, RF propagation tools which are parameterized by simpler means constitute a more suitable tool.

The remainder of this paper is structured as follows. In Section II, the basic concepts of loop simulation of wireless networks are introduced. In Section III, the integration of different aspects of radio front-end modeling into a loop simulator are explained. The wireless channel simulation based on a channel matrix is discussed in Section IV. The loop approach for development is then demonstrated with a simple co-channel interference analysis application in Section V. Section VI concludes.

II. A Software Radio Network Simulator

A. *Concept*

In the following, a *software defined radio* (SDR) is a program capable of creating and processing (transmitting and receiving) digital complex baseband signals. An SDR can transmit and receive radio frequency (RF) signals by accessing a radio front-end device via a standardized API, such as, e.g., the SDR Forum Transceiver Facility [3]. The API separates the

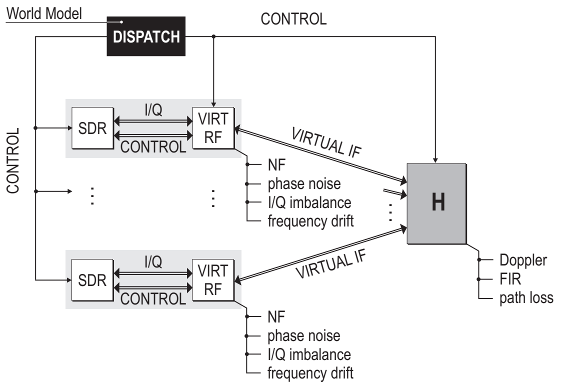

Fig. 1. Overview of a generic software radio network simulator

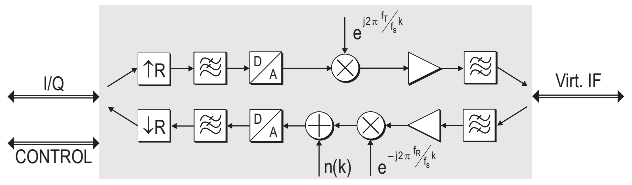

Fig. 3. Schematic overview of a virtual RF front-end

RF front-end hardware from the actual waveform application and allows to abstract both.

Figure 1 gives an overview of the proposed simulator design. The schematic SDR block stands for a waveform model at any design stage which accesses a virtual RF front-end device. Access to the virtual RF front-end is controlled through the same standardized transceiver API. The virtual RF front-end simulates analog domain signal processing and outputs a digital sample stream to a channel matrix $\mathbf{H}$. The channel matrix operates synchronously in a single simulation bandwidth. Thus, several SDRs forming a software radio network can be simulated.

Such a *software radio network* consists of several SDRs interacting in some way. The network can consist of subnetworks, which can be identical or different, and which can interfere or cooperate. While theoretical analysis of such systems is necessary for an initial parameterization, the high complexity makes detailed theoretical analysis impossible. Loop-testing and verification on physical hardware are necessary during design. With a standardized transceiver API, appropriate abstraction of the RF front-end and channel simulation, switching between testing in real and virtual mode is seamless. Figure 2 shows this development loop.

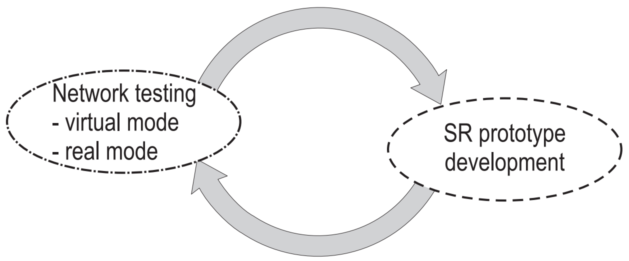

Fig. 2. SR development loop

B. Implementation

The implementation of a loop development platform presented here uses standard hardware and free and open source software (FOSS). SR development is done using the GNU Radio framework [4], which allows for rapid development by combining the advantages of the Python and C++ programming languages. All software is designed to run on standard PCs. The authors' choice of RF hardware for real mode are the Universal Software Radio Peripheral (USRP) I and II by Ettus Research LLC [5]. The virtual IF connection between the GNU Radio SDR and channel matrix processes is implemented using *named pipes*, a type of FIFO buffer provided by the operating system.

III. Virtual RF hardware

For any given RF hardware, the RF front-end has to be abstracted to a level that allows seamless transition from virtual to real mode. The modeling process is similar for all RF hardware and demonstrated here for the USRP front-end.

A virtual RF front-end has to emulate all signal processing needed to transform the time discrete I/Q baseband signal to an analog pass band signal and vice versa. The analog pass band is simulated digitally in a common *simulation bandwidth* for all nodes.

In the transmit path, a virtual RF front-end has to implement the following functionalities:

1) interpolation to digital/analog converter sampling rate,
2) digital to analog conversion (DAC) and interpolation to simulation bandwidth,
3) mixing to pass band,
4) amplification,
5) and analog RF filtering.

The receive path is implemented analogously. For a usable abstraction, the virtual front-end has to take into account the most significant non-idealities. In most cases, at least the digital up and down conversion (DUC/DDC) chain, RF filtering and the RF noise figure (NF) will have to be simulated. In other cases, non-idealities such as non-linear amplification, frequency drift, phase noise or even I/Q imbalance dominate the performance and have to be included during modeling.

The abstraction follows the principal structure shown in Figure 3. In the following, the main non-idealities affecting the USRP are described. In the implemented USRP front-end abstraction, analog RF filtering and noise figure are included in the model.

A. Interpolation and decimation

The I/Q signal coming from the SDR waveform application is interpolated in RF front-ends to the sampling rate of the DA converter. In case of the USRP the DAC frequency is 128 MHz. This frequency is chosen to be the simulation bandwidth. Applications using the USRP have to provide a

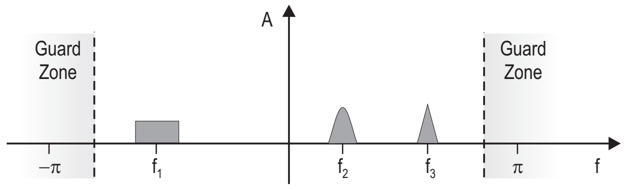

Fig. 4. Schematic overview of the simulation bandwidth with three different SDRs operating in it. The guard zone is inserted to avoid end-to-end spectral leakage.

fixed sampling rate, which is then internally interpolated to DAC rate. The interpolation factor of the USRP is limited to multiples of 4 between 4 and 512. This yields application sampling rates between 250 kHz to 32 MHz. The actual RF bandwidth and filter characteristics depend on the USRP RF daughterboard; RF bandwidths of 20 MHz are common.

The interpolation mechanism implemented in ithe virtual RF front-end mimics the USRP CIC/FIR-based interpolation stage. The USRP has a fixed FIR interpolation filter in the Analog Devices AD9862 mixed signal front-end processor, which interpolates with a fixed factor of 4. Furthermore, a tunable CIC filter is found on the FPGA which interpolates to the desired rate over 4 stages. The interpolation factor R in the CIC is tunable, multiples of 4 between 1 and 128 are possible. The filter response of the CIC filter in the FPGA is [6]:

$$H_{\mathrm{CIC}}(z) = \left[\sum_{k=0}^{R-1} z^{-k} \right]^4 \tag{1}$$

After interpolating with these filters to simulation bandwidth, a virtual RF IIR filter is applied and the signal is shifted to its final frequency in simulation bandwidth.

The converse is true for the receive path. The signal is RF filtered, mixed and decimated from the simulation bandwidth to the application sample time.

B. Digital to analog and analog to digital conversion

The upsampled signal is fed into a DAC simulator block which quantizes the signal according to a given characteristic, corresponding to the 12 bit per sample resolution offered by the USRP. To simulate analog to digital conversion, the incoming signal is quantized and then forwarded to the down-sampling block. The simulation of the DAC and ADC is based on fixed point/floating point conversion to correctly model CIC interpolation stages, as the bit resolution needed to represent a filtered result increases with the number of interpolation stages. The final result is then truncated to DAC resolution and converted to floating point.

C. Noise figure

The noise figure describes the relation of the incoming to the outgoing signal-to-noise ratio (SNR) and is hence an important qualitative parameter of the receiver front-end characterization:

$$\mathrm{NF} = \frac{\mathrm{SNR}_{in}}{\mathrm{SNR}_{out}}\,. \tag{2}$$

The noise figure quantifies Johnson thermal noise. It is simulated by adding white Gaussian noise $n(k)$ to the signal after RF filtering and before quantization. Johnson thermal noise at temperature T has the probability density [7]

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}} \tag{3}$$

with variance of the noise process given by

$$\sigma^2 = T k_b \left(10^{\frac{\mathrm{NF}}{10}} - 1\right) \tag{4}$$

where k_b is Boltzmann's constant and NF is specified in dB.

D. Phase and frequency drift

As the RF front-end is not able to reproduce the exact carrier frequency of the wanted signal, there is a phase offset ρ_o and time dependent frequency drift $\Delta f(k)$ in the received signal. The signal processing of the SR has to synchronize frequency and phase to correct these offsets. If $x(k)$ denotes input signal $y(k)$ the output signal, the drift and phase offset can be described as:

$$y(k) = x(k) e^{j(2\pi \Delta f(k) k / f_s + \rho_o)} \tag{5}$$

where f_S is the sampling rate of the signal. Frequency drift and phase offset are introduced during modulation from virtual IF to base band.

E. Phase noise

Phase noise is a serious RF impairment in communication technology due to the fact that it appears in any oscillator and any RF front-end is equipped with oscillators. High phase stability is important for long correlators found in, e.g., spread spectrum applications. Phase noise is quantified by calculating the ratio between the carrier power P_C and the noise power P_N in a unit bandwidth at a frequency offset Δf from the carrier frequency f_C [8], [9]. It is measured in dBc/Hz and provided for any oscillator on the RF part of the chosen platform. Phase noise can be simulated by filtering white Gaussian noise in a manner that the resulting power spectral density is proportional to $1/f$. Kasdin proposes to implement this filter with the frequency response [10]

$$H_{\mathrm{PN}}(z) = \frac{1}{\sqrt{1 - z^{-1}}}\,. \tag{6}$$

The transformation of (6) to the time domain can be calculated with help of the power series expansion which leads to the following recursive pulse response:

$$h_{\mathrm{PN}}(k) = 1 \qquad k = 0 \tag{7}$$

$$h_{\mathrm{PN}}(k) = \left(k - \frac{1}{2}\right) \frac{h_{\mathrm{PN}}(k-1)}{k} \qquad k > 0 \tag{8}$$

The coefficients of the IIR filter providing that impulse response can be found by taking again the iterative algorithm with the denominator taps:

$$a_0 = 1 \tag{9}$$

$$a_k = \left(k - \frac{3}{2}\right) \frac{a_{k-1}}{k} \tag{10}$$

and just one single numerator tap which provides a scaling factor of the phase noise power P_{PN} given in dBc:

$$b_0 = \sqrt{2\pi \Delta f 10^{P_{\mathrm{PN}}/10}} \quad (11)$$

After designing the filter, the output of the phase noise block $y(k)$ is given by

$$y(k) = x(k) e^{j(n(k) * h_{\mathrm{PN}}(k))}\,, \quad (12)$$

where $n(k)$ are discrete samples from a white Gaussian noise process. Phase noise needs to be modeled in the transmit and receive path. It is introduced during modulation to and from virtual IF.

IV. Simulating an RF environment: The channel matrix

A radio propagation channel between nodes k and l is completely described by its time-variant impulse response (CIR) $h_{k,l}(t,\tau)$. Any propagation path effect such as multipath propagation, Doppler spread, free space path loss etc. are modeled by such a time-variant CIR [11]. Consequently, such channels can also be modeled in a discrete-time fashion. This is a well-understood topic of radio engineering; in [11], mathematical foundations and examples of channel modeling are described. For simplicity, the CIR will be abbreviated as $h_{k,l}$ for the following section.

The entirety of channels between N SDR nodes can be summarized in the $N \times N$ channel matrix $\mathbf{H}_{\mathrm{Chan}}$:

$$\mathbf{H}_{\mathrm{Chan}} = \begin{pmatrix} h_{1,1}(t,\tau) & \cdots & h_{1,N}(t,\tau) \\ h_{2,1}(t,\tau) & & \vdots \\ \vdots & \ddots & \vdots \\ h_{N,1}(t,\tau) & \cdots & h_{N,N}(t,\tau) \end{pmatrix} \quad (13)$$

To calculate the received signal $r_k(t)$ at node k, the sum of the channel responses of the transmitted signals $s_l(t)$ must be computed:

$$r_k(t) = \sum_{l=1}^{N} s_l(t) * h_{l,k}(t,\tau) \quad (14)$$

For N SR nodes, a total of N^2 radio channels exist. For the implementation, it is useful to decrease this complexity, which can be achieved by analyzing the logical setup of the radio network. First of all, some nodes might only receive or transmit. In this case, it is not necessary to calculate the propagation of the radio waves to or from these nodes, meaning the CIR can be set to zero. The backscatter channel $h_{k,k}(t,\tau)$ is also rarely of interest, since radio systems often cannot receive and transmit at the same time. Finally, radio propagation channels are usually symmetric, meaning that if $h_{k,l}$ and $h_{l,k}$ both exist, they are equal.

Assume a radio network with three nodes: node 1 can only transmit, node 2 can both transmit and receive, but not at the same time, and node 3 can only receive. In this case, a suitable channel matrix is of the form

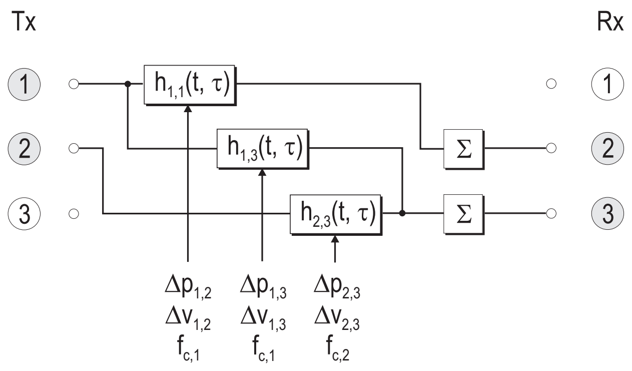

Fig. 5. The signal flow graph for the channel matrix (15)

$$\mathbf{H}_{\mathrm{Chan}} = \begin{pmatrix} 0 & h_{1,2} & h_{1,3} \\ 0 & 0 & h_{2,3} \\ 0 & 0 & 0 \end{pmatrix}. \quad (15)$$

By removing all redundant channels, the computational complexity can be reduced significantly. This feature of deactivating channels can be used for special types of testing which are difficult in real world scenarios, such as the test case in Section V: here, two nodes are only transmitting and the other two nodes are only receiving.

A. Geometrical channel matrix setup

For implementation, the channel matrix itself and the individual point-to-point channels are separate entities. During process initialization, the channel matrix loads a list of user-defined channels, parameterizes them according to a *world model* and connects them into a complete flow graph. The world model includes, e.g., a list of geometrical node positions and trajectories, and is used to adaptively change the individual point-to-point channels.

Figure 5 shows how the signal flow graph looks like for the channel matrix (15). At startup, the process reads the node and channel configuration, and creates three channel processing blocks. These blocks are then connected to the nodes' transmit ports accordingly, and summed up before sending the processed signals back to the receive ports. How the channel processing blocks are implemented is up to the end-user. Random or deterministic fading channels, multipath, Doppler etc. can all be easily implemented in GNU Radio.

B. Interfacing channel and SR processes

The channel matrix and the SR processes communicate by the means of named pipes, which are a typical method for inter-process communication. This allow for simple means to connect any kind of SR process - not necessarily written in GNU Radio. The SR nodes must transmit a continuous stream of complex baseband samples at a sampling rate equivalent to simulation bandwidth - even when not transmitting - to ensure synchronicity. The processed post-channel signals are then continuously streamed back to the SR processes through different pipes.

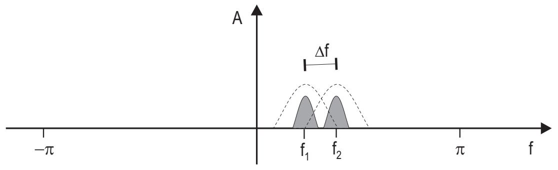

Fig. 6. Co-channel interference analysis in the simulation bandwidth. The dashed line represents the virtual RF filter. Δf is varied during tests.

V. Demonstration of Rapid Software Radio Network Development: Analysis of co-channel interference

To illustrate the concept of loop development for wireless networks, a very simple co-channel interference analysis application is considered. The following section will demonstrate how the loop concept can be applied to aid the developers create fully tested code for the individual SDR terminals.

The goal is to experimentally determine a suitable frequency spacing $\Delta f = f_2 - f_1$ for a number of point-to-point digital voice transmissions, such as depicted in Figure 6. In total, there are four SR nodes. The transmitting nodes (Tx nodes) transmit a frame-based digital voice signal on frequency f_1 and f_2. The receiving nodes (Rx nodes) decode the signal on either channel f_1 or f_2. The goal is to quantify the co-channel interference influence.

All SDR terminals shall be developed in GNU Radio. The figure of merit which will be used for evaluation is the frame error rate at the Rx nodes.

A. Initial Prototyping

In a first step, prototypes of the nodes are created using the GNU Radio software framework. GNU Radio offers a signal processing framework, where signal processing *blocks* are connected to form a GNU Radio *flow graph*. Blocks implement atomic signal processing operations or, in the case of a hierarchical block such as the channel matrix, contain another flow graph. Blocks without input are called *source blocks*, blocks without output are *sink blocks*. Signal processing blocks are written in C++. The waveform application, which sets the flow graph up and processes user input, is written in Python.

A flow graph consists of a non-recurrent chain of signal processing blocks, connected into a signal processing chain. The top-level flow graphs for the transmitting and receiving nodes are shown in the block diagram in Figure 8. Signal processing is split into the following blocks: audio I/O, Speex source encoding/decoding, transmit/receive path. The code in development is not written specifically for testing or any special case; it is created to be directly taken to the air. The only difference is that the code implements a way to feed back testing information, in this case the frame error rate.

The following GNU Radio blocks are used in the featured example:

Audio sink/source: The audio I/O blocks are used at the transmitter and receiver to send/receive voice data or audio files. They implement audio resampling.

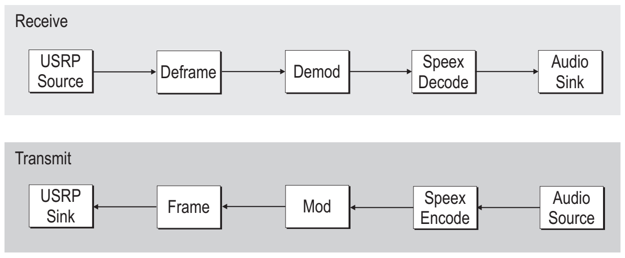

Fig. 7. Top-level GNU Radio flow graphs for digital voice transmission and reception.

Speex encoding/decoding: Speex [12] speech encoding/decoding is used to encode and compress the audio signal. Speex is an open-source/free software, patent-free audio compression format designed for speech based on code excited linear prediction [13]. Speex offers low data rate digital audio encoding with packet-loss concealment, making it an appropriate choice for a digital voice transmission.

Transmit/receive path: The transmit/receive path is a hierarchical block, which includes the three main components of the transmission/reception line. It includes the necessary functions for setting up the (virtual) USRP as source/sink, modulation/demodulation, packetization/packet deframing. Packets are made up of the preamble for synchronization and detection, the modulated symbols, and a cyclic redundancy checksum (CRC).

USRP drivers: The drivers for the USRP include function calls to choose the daughter board, select the transmit/receive centre frequency, setting gain and desired sampling rate. Unlike usual GNU Radio code, the library import directives are designed to be able to switch between virtual and real RF front-ends. This does not affect the waveform code itself.

Modulation/demodulation: The GNU Radio framework provides a number of modulation/demodulation techniques for use in applications. The modulation type is user configurable.

Packet framing/deframing: GNU Radio provides functions to build and deframe packets. The packetization function builds a packet given an access code and payload. The packet consists of a preamble for synchronization in the beginning, followed by a length indicator and a payload. It ends with a CRC-32 checksum. The deframer removes preamble and checksum, and if received without errors, converts the received packets back into stream of data.

B. Applying the development loop

Once the prototypes have reached a state where they can be run, the testing stage begins. The channel matrix is initialized with a very simple setup, where the nodes are connected through simple flat fading channels. On startup, two separate waveform processes with a virtual RF front-end and the channel matrix process are initialized.

$$\mathbf{H}_{\text{Chan}} = \begin{pmatrix} 0 & 0 & h_{1,3} & h_{1,4} \\ 0 & 0 & h_{2,3} & h_{2,4} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{16}$$

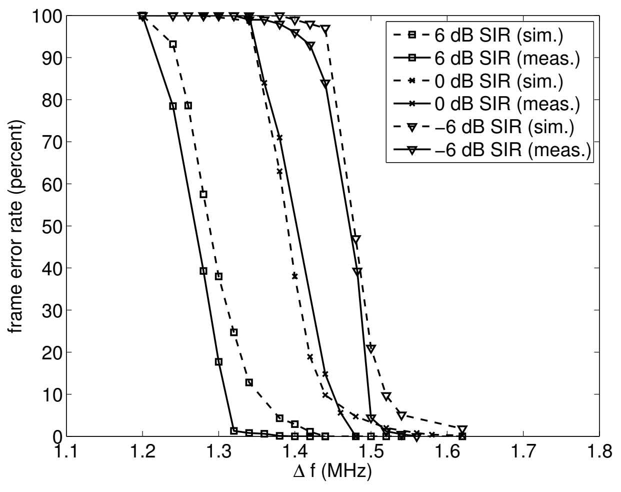

Fig. 8. Simulated and measured co-channel interference results. RF filter bandwidth is 2 MHz. The signals are GMSK modulated with $f_{3\mathrm{dB}}T = 0.35$. The signal-to-interference ratio (SIR) at the receiver is specified, the signal-to-noise ratio (SNR) is fixed at approximately 22 dB (high SNR regime).

The channel matrix is shown in (16).

For testing, frame error rates are noted for a certain Δf. In the next step, the test is re-initialized with a different channel spacing and re-run. Finally, when the entire system works well in virtual mode, the switch to real mode can be performed for validation. If the RF front-end and channel conditions are modeled with adequate detail, the frame error rates for a certain channel spacing will be similar. Figure 8 shows frame error rate results. For the co-channel interference setup described here, the experiment shows good consistency between real and virtual mode.

VI. Conclusion

For software radio networks, direct simulation of the complex interactions between nodes in wireless networks is feasible, and a software solution to do so is presented. Without code changes, the simulation results can be directly verified in real environments. Especially during the development stage of complex wireless networks - be it in the context of mobile ad hoc networks or spectrum overlay - the possibility to seamlessly move from simulation to real networks is a significant advantage.

The conceptual solution presented here based on GNU Radio is currently under development and will be released under the GNU General Public License.

References

[1] S. Nagel, V. Blaschke, J. Elsner, F. K. Jondral, and D. Symeonidis, "Certification for SDRs in new Public and Governmental Security Systems," in *Proceedings of the SDR Forum Technical Conference 2008*, Oct 2008.

[2] W. T. Kasch, J. R. Ward, and U. Andrusenko, "Wireless Network Modeling and Simulation Tools for Designers and Developers," *Communications Magazine, IEEE*, vol. 47, no. 3, pp. 120–127, March 2009.

[3] E. Nicollet and L. Pucker, "Standardizing Transceiver APIs for Software Defined and Cognitive Radio," *RF Design*, Feb 2008.

[4] Free Software Foundation, "GNU Radio," http://gnuradio.org/.

[5] "Ettus Research LLC," http://ettus.com/.

[6] E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 155–162, April 1981.

[7] J. G. Proakis, *Digital Communications*. McGraw-Hill, 2001, 4th edition.

[8] B. Razavi, *RF microelectronics*. Prentice Hall, 2005.

[9] B. Razafi, "Design considerations for direct-conversion receivers," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Jun 1997.

[10] N. J. Kasdin, "Discrete simulation of colored noise and stochastic processes and $1/f^{\alpha}$ power law noise generation," in *Proceedings of the IEEE*, vol. 83, no. 5, May 1995.

[11] M. Pätzold, *Mobile Fading Channels - modelling, analysis and simulation*. Wiley, 2001.

[12] "Speex: A Free Codec For Free Speech," http://www.speex.org/.

[13] M. Schroeder and B. Atal, "Code-excited linear prediction(celp): high-quality speech at very low bit rates," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 937–940, 1984.