MICRO COGNITIVE RADIO NETWORK TESTBED (MICRONET) FOR EDUCATION, EXPERIMENTATION, AND DEMONSTRATION

Eric Sollenberger, Vuk Marojevic, Carl Dietrich
Virginia Tech
Blacksburg, VA, US
ericps1@vt.edu, maroje@vt.edu, cdietric@vt.edu

ABSTRACT

This paper discusses the design and implementation of Micronet, a testbed that enables realistic, over-the-air software-defined radio (SDR), cognitive radio (CR), and dynamic spectrum access (DSA) experimentation and demonstration at low cost. The system can be extended to enable larger scale experimentation at moderate cost, making it well suited for quick prototyping or proof-of-concept demonstrations of experimental systems or operational concepts for limited-resource educational and research purposes. Through use of a similar interface, the testbed prepares users for remote use of more capable testbeds, such as Virginia Tech's CORNET, for large-scale and high-bandwidth applications. Micronet is a tangible alternative to large scale testbeds. It can be locally controlled and configured in a few steps for setting up popular education and research environments.

1. INTRODUCTION

Software-defined radio (SDR) enables the dynamic reconfiguration of radio transmission modes [1]. The software can make efficient use of the available hardware and the environmental conditions. A cognitive radio (CR) system makes use of an SDR platform for this purpose. A CR system can have different facets, but typically involves observing (spectrum awareness), learning (cognition /intelligence) and decision making (reconfiguration) [2]. CR can make efficient use of radio and computing resources in combination with SDR and a flexibly regulated frequency allocation system that is shared among its users, known as dynamic spectrum access (DSA).

SDR and CR research has existed for 1-2 decades already, but their potential is far from being fully exploited in practice. Several research contributions lack practical relevance, because they are either too complex to be implemented or lack the necessary development and deployment tools. Being able to implement and test a cognitive engine—the brain of the observing-learning-decision making cycle—for instance, is essential for moving from the theoretical analysis and simulations to demonstrations and prototypes. One popular approach for this is the use of wireless testbeds.

Testbeds have played an important role in the development of networking technologies since the first computer networks were developed. They help advancing wireless technology by enabling quick prototyping and testing in realistic setups.

The WINLAB *ORBIT* testbed at Rutgers University is a large-scale wireless network testbed with over 400 nodes, each consisting of a PC equipped with 802.11 a/b/g network cards and supplemented with flexible radio frequency (RF) platforms to exploit physical layer adaptations [3]. The testbed expands over a 20×20 m^2 grid. The wireless networking research group at the University of California, Riverside, has deployed a testbed consisting of fifty-eight 802.11 wireless nodes with additional multiple-input multiple-output (MIMO) cards, 15 laptops connected to Universal Software Radio Peripherals (USRPs) from Ettus Research, and 6 laptops connected to the Rice University WARP radios [4]. Like ORBIT, the majority of the nodes in this testbed are targeted to upper layer network protocol research. However, unlike ORBIT, this testbed includes both the USRP and the WARP boards, which have flexible and powerful processors onboard. The *Emulab* research facility at the University of Utah is another large-scale testbed of approximately 275 nodes that recently supplemented with USRPs [5]. Virginia Tech's CORNET consists of 48 SDR nodes—distributed USRP 2s, each connected to a server—in a research building of Virginia Tech's main campus in Blacksburg, Virginia. These nodes are at fixed indoor locations. Mobile and outdoor nodes are currently being deployed. The use of CORNET is free. Registered users can schedule nodes and login remotely to control one or several nodes and run a variety of experiments [6].

Large-scale testbeds are a valuable resource for research and education, but often lack full transparency because they are operated remotely. Whereas most researchers are familiar with this kind of access, others might prefer seeing the hardware they are operating and be able to observe the effects of physical changes to the testbed topology. The Micro Cognitive Radio Network Testbed (Micronet) provides this alternative.

A major motivation for Micronet is to combine the available hardware and software resources into a useful collection for SDR/CR. Micronet intends to be a prototype of a collection of small-scale, wireless communications testbeds that are creatively assembled using heterogeneous hardware pieces, upon availability.

Micronet is a modular design that uses PCs with modest performance, commodity digital video broadcast (DVB) tuners/receive-only SDR RF front ends, and optionally, SDR transceivers. Relatively short transmit-receive distances facilitate unlicensed operation, e.g., under FCC Part 15 rules, and use of laptop computers enables the system to be set up and taken down quickly for use during class time or demonstrations if no dedicated space is available. Use of open-source software allows packaging of system software as well as SDR toolkits and demonstration applications on live DVDs or bootable flash drives for turn-key installation on user-supplied hardware.

Micronet presents a more tangible way to get a first research experience in SDR/CR, and thus may also serve as a stepping stone for using larger testbeds, such as CORNET, for running more sophisticated cognitive radio or related experiments. Micronet is ideal for students and new researchers in the area who want to have their own testbed despite limited resources and moderate technical background. Our goal is to show how systems like Micronet can be created with much less difficulty and time commitment than might be expected.

The basic concept of Micronet is modeled after CORNET, which provides access to common SDR design and rapid prototyping tools. Some of these tools are briefly described in Section 2. In Section 3 we explain Micronet's design tradeoffs, followed by Micronet's setup and initial results in Section 4. Section 5 concludes the paper.

2. SDR DESIGN TOOLS

Research and development has been conducted in the area of SDR in the past 15-20 years and there are many resources available for research, development and education. Since SDR is by definition primarily a function of software, new capabilities can be added to the existing toolboxes, without any need to update hardware.

2.1 SDR Frameworks

SDR frameworks have emerged after the software communications architecture (SCA) was first introduced by the Department of Defense in 1999. An SDR framework facilitates the waveform development and deployment on different hardware platforms (waveform portability). Design and deployment tools for rapid prototyping and testing complement the framework. Most SDR frameworks are proprietary, developed by companies for their customers' needs, whereas others are open source. Popular open-source frameworks include Virginia Tech's Open-Source SCA Implementation-Embedded (OSSIE) [8], which evolved to REDHAWK, IRIS [9], which was developed at Trinity College Dublin, Ireland, and Abstraction Layer and Operating Environment (ALOE) [10] [11], developed at Polytechnic University of Catalonia, Spain. Interestingly, the development of all three frameworks started at about the same time, around 2004.

All three frameworks allow distributed processing. OSSIE/REDHAWK was built for education at Virginia Tech and elsewhere on SCA and SDR, in general, and has been adopted by many R&D projects. It relies on CORBA middleware as does SCA. ALOE provides a lightweight operating environment for distributed waveform processing. It enables platform independence—both waveform portability and performance—and provides hard real-time distributed processing control. IRIS has been specifically designed for cognitive radio research and development and has demonstrated DSA capability.

2.2 GNU Radio

GNU Radio is a popular open-source SDR tool for research and education [12]. It includes an extensive library of signal processing blocks and user interfaces that are connected to produce SDR waveforms or flow graphs. GNU Radio Companion (GRC) allows users to build flow graphs by assembling a block diagram of their system, connecting the blocks, and setting parameter values for the desired operation. GRC also allows the specification of a user interface with embedded controls, such as sliders and text boxes. Users can execute flow graphs developed with GRC, edit the correspond-

ing Python code, or program their own GNU Radio applications in C++/Python. In addition to the library of blocks and examples provided with GNU Radio, third party applications are available through the Comprehensive GNU Radio Archive Network (CGRAN) as well as elsewhere on the Internet. The GNU Radio user community is continuously growing and provides support to new users and solutions to specific problems.

2.3 Waveform Projects

Open source waveform packages make it possible to experiment with a variety of radio standards. There are many completed packages and many more are under development. For example, OpenBTS [16] is a software package that allows an SDR peripheral to be used as a 2G cellular basestation based on the GSM standard. Similarly, a couple of open-source LTE projects are ongoing [11] [17] and, once mature, will allow users to modulate and demodulate LTE signals. All these initiatives need a computing platform and they can be integrated as part of the testbed. Micronet may not be able to support the entire LTE processing stack due its processing limitations, but specific LTE channels or simplified LTE versions may be able to run on Micronet for educational or research purposes.

3. MICRONET DESIGN TRADEOFFS

Many decisions had to be made in terms of how to set up the testbed. We will discuss some of the options here followed by a description of what was actually implemented and why. That is, we identify the tradeoffs and rationale for our selection process from among alternative approaches so that others interested in building a testbed for similar purposes may benefit from our prior work. The critical components to be selected are the RF front end, the processing platform, and the interface among the radio front ends and the processors. Our goal was to use open source platforms that provide the necessary functionality. Using open source tools allows more flexibility in developing applications and experiments.

3.1. RF Boards

There are several alternatives for the RF front ends. The selection of which to use is a function of the testbed's purpose. For example, one of the demonstration applications planned for Micronet is an SDR positioning system. In this experiment the signal is never demodulated, and so the modulation is irrelevant as long as the envelope of the signal is constant. In this case, we can implement an extremely inexpensive system using only RTL-SDRs [7] as the sensing or receiver nodes, implementing the position locating algorithm, and a simple FM transmitter for the device to be tracked. This illustrative example shows the ability to tailor the testbed to a use case or set of use cases as required.

A more general testbed that can handle a range of experiments would need hardware capable of converting complex baseband signals to RF. The most inexpensive options with this capability are the HackRF [18] and the BladeRF [19]; two SDR transceivers that offer wide baseband and RF bandwidths. Both of these options are more expensive than all the other components of Micronet combined though. A good compromise for Micronet is to use the Raspberry Pi (discussed in the next section) as an FM transmitter, an unintended capability that was discovered recently by the Imperial College Robotics Society. With this approach it is possible to use Frequency Modulation, Frequency Shift Keying, and possibly On-Off Keying. The only additional hardware needed is a 15-20 cm wire to act as an antenna. We are in the process of developing a GNU Radio block to more readily incorporate the Raspberry Pi as a transmitter for Micronet.

3.2. Processing Boards

Regardless of the RF front end, off-board processors are often required to operate the front end. There are a range of options. The type and number of processors need to be selected depending on the users' requirements. Since the goal of Micronet is to be mobile, flexible and low-cost, an inexpensive netbook was selected as the primary processor.

The *Raspberry Pi* and *BeagleBone Black* are low-cost ARM processor-based single-board computers that can be used to capture data for RTL-SDR front ends, connected to the main network using TCP over Ethernet. Processing power is limited, but an FM signal can easily be captured and sent over TCP on the Raspberry Pi without any performance optimization. It is also possible to script the Raspberry Pi to act as a spectrum analyzer, which would be useful for spectrum sensing in DSA experiments. GNU Radio is straightforward to install, which may allow for lightweight on board DSP, such as calculating average power in a channel or demodulating narrowband signals.

3.3. Computing Topology and Interfaces

A major question is what kind of interface to use among the RF front ends and processors. CORNET employs a dedicated server for each radio, providing dedicated bandwidth and processing power for sophisticated DSP. Since the resources of Micronet are limited by design, other options need to be considered. The most straightforward method would be an all-USB interface. Every node, be it an RTL-SDR or a USRP, would be connected to a single PC via USB. The potential issues of this setup are the demand for processing power and interface capacity bottlenecks. USB extensions can be used to spatially separate the SDR nodes by up to 40 ft. (~12m) from the central PC. This distance should offer more than enough space to conduct the planned experiments for Micronet. The other option is to have multiple processors, each serving either a single or several SDR peripherals. This could prove to be a useful strategy if an application requires greater separation than USB cables allow, if the central processor is not powerful enough to handle all the peripherals at once, or if the desired location of the nodes is somehow separated, making it difficult to physically connect them. Data can be transferred between nodes via TCP over a LAN very easily using GNU Radio.

An important consideration when using a networked implementation as described above is how to divide the signal processing between nodes. For example, an FM receiver was implemented using an RTL-SDR connected to a small netbook, which was then connected to a primary PC via TCP. It was more efficient to demodulate the signal on the netbook and send an audio signal over TCP than to send the raw samples to be demodulated by the primary PC. This is certainly a function of many variables including the processing power of the PCs used and the application, but it illustrates the important concept of signal processing optimization in CR environments. Also noteworthy is the ability to set up Micronet on an ad-hoc network, allowing for demonstrations to be done regardless of available connections to an external network.

3.4. Operating System and Software Development Tools

3.4.1. Operating System

Ubuntu 12.04 LTS was selected as the operating system (OS) for Micronet due to software compatibility and long term support. Several software packages were considered for use with Micronet. Selected packages are included in the Micronet images, and documentation on how to install and use other relevant packages will be made available through the CORNET website [6].

3.4.2. SDR Development and Deployment Tools

GNU Radio was selected as the initial waveform development tool because of its open source nature, ease of use, and the functionality and community support it offers. Also, the graphical interface of GNU Radio Companion is very intuitive to use and provides readily-available digital signal processing blocks. REDHAWK [21], Octave, or Matlab are also strong candidates for future integration based on their capabilities.

Liquid DSP [22], an open-source signal processing library for SDRs, is also included in the default Micronet image as a supplementary DSP package. It offers a variety of C-language, Physical layer DSP functions that can be compiled and chained, e.g., using C or C++, without relying on any framework or external dependencies.

3.4.3. Accessibility

Other software that is useful for administrative purposes included XRDP and cluster secure shell (SSH). XRDP is a remote desktop client, which may be convenient for educational purposes as opposed to a command line interface. Cluster SSH is an application that enables a user to control multiple SSH sessions from a single terminal, which is useful for controlling a testbed that contains several nodes performing the same function. If, for instance, an array of receivers were used to characterize signal propagation through a building, Cluster SSH would allow activation of all nodes at once.

4. ANALYSIS

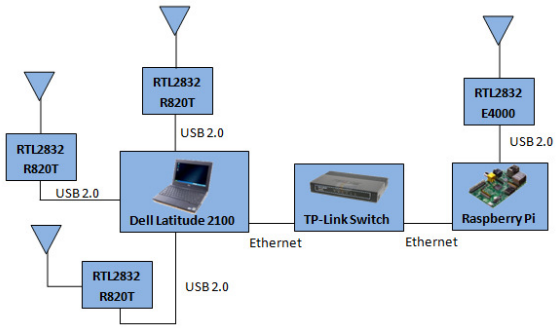
A functional testbed and quantifiable results are the primary deliverables of this initiative. Specifically we are interested in how many SDR nodes Micronet can support and what they will be capable of doing. Various experiments were conducted using Micronet in order to provide a reasonable answer to this question. The “top” utility available in Ubuntu was used to monitor CPU usage during these experiments in order to provide insight into the limit of SDR nodes that can be supported and the complexity of DSP that can be implemented.

4.1 Micronet Hardware

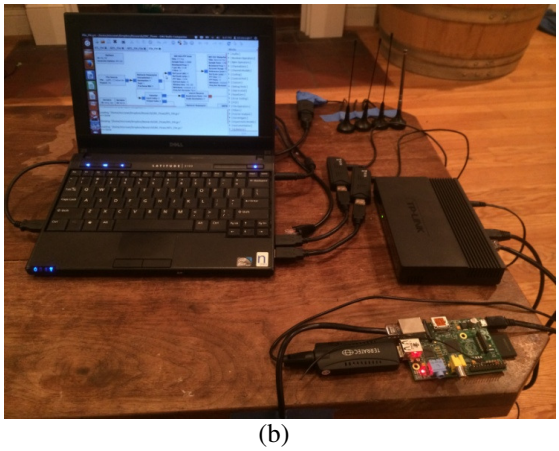
Currently, Micronet consists of a Dell Latitude 2100 netbook (dual-core, 1.6 GHz Intel Atom processor with 1 Gb RAM) [13], with three RTL2832 R820T tuners [14] connected by USB, and an additional RTL2832 E4000 tuner [15] that is connected via TCP over LAN using a Raspberry Pi (Fig. 1). The E4000 tuner was found to be more readily compatible with the Raspberry Pi. The total cost of this testbed was under $250. A bill of materials is shown below.

Table I – Bill of Materials for Micronet.

Item	Qty	Supplier
Dell Latitude 2100	1	Ebay (used)
Raspberry Pi	1	Newark
RTL2832 R820T	3	Ebay
RTL2832 E4000	1	Ebay
8 Gb SD Card	1	Best Buy
10' USB Extension	3	Sweetwater
Micro USB power cable	1	MCM Electron-ics
Ethernet cable	2	Rakuten.com
TP-Link Ethernet Switch	1	Newegg



(a)



(b)

Fig 1. Micronet hardware: (a) block diagram (b) photo

4.2 Experiments

4.2.1. FM Receivers

With the above hardware configuration we are able to run various GNU Radio signal flow graphs (Figs. 5-7 at the end of the paper). In particular, we simultaneously sampled the four RTL input data streams at 1.024 Msps and FM demodulated each signal, saving the resulting audio to .wav files. The CPU load during this process was on average 90%, indicating that this example hits Micronet's approximate maximum capacity in its current configuration.

4.2.2. Processing Loads of Different Flow Graphs

The CPU usage was measured during the execution of various flow graphs including those shown in Figures 5-7. Figure 2 below depicts these CPU loads. These flows include FM demodulation of a single RTL-SDR, data capture using one, two and three RTL-SDRs, and FM demodulation and data capture of an RTL-SDR connected via TCP using the Raspberry Pi. Each curve begins at approximately the minimum sampling rate of the RTL2832 and continues until the load on the CPU causes glitching. Note that these are averages, which is why many curves do not get very close to 100% CPU usage. CPU usage is time varying in nature due to many factors including OS overhead, file I/O, and connectivity to the LAN. As such, a processing burst can cause the application to stall and ultimately quit even though the average CPU usage was far below 100 %.

This data indicates the limits of Micronet's capabilities in terms of maximum sample rate that can be achieved for each experiment. The curves in Fig. 2 indicate that the CPU usage is roughly proportional to the sampling rate of the RTL's.

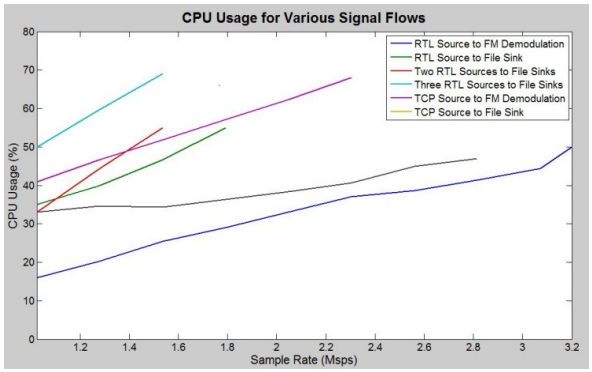


Fig 2. CPU load while executing various signal flows.

4.2.3. Spectrum Sensing

An important feature for any CR testbed is the ability to observe the radio environment. This is important in applications such as DSA, as a radio may need to sense where the spectrum is and is not being used. Micronet is capable of reading data from all four of its connected RTL dongles and saving this data as binary files for later processing or performing other light weight DSP such as FM demodulation. In other words, real-time spectrum sensing is possible for all four nodes simultaneously, yielding a total observable bandwidth of 4-10 MHz. If real-time spectrum sensing is not required, another option is to have the Raspberry Pi save waveforms or other data locally and then copy the resulting files over to the main PC afterwards for post processing.

It may also be desirable to look at sections of the radio spectrum that are wider than the bandwidth of a single RTL, or even all four RTL's. For this reason Micronet has included the RTLSDR-Scanner GUI [23] as well as a custom script which appears to be faster and more configurable. These spectrum sensing methods are not real time, as they function by sweeping the RF center frequency of the RTL across the desired band, collecting data at each step. Even so, these methods can provide useful information into the spectrum usage over large bandwidths. A plot of the FM radio band that was obtained using the custom script is included below in figure 3. In this plot, various FM stations can easily be identified, the strongest of which are 90.7, 94.9, 99.1, and 105.3 MHz.

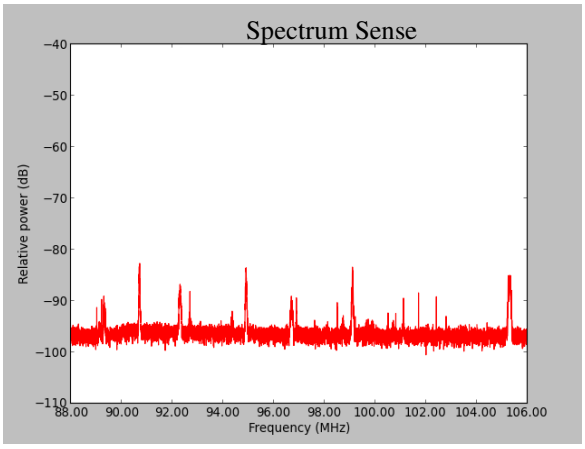


Fig 3. A plot of the FM band as sensed with an RTL SDR. Approximate sweep time is 2.5 s.

4.2.4. *Additional Experiments*

Many additional experiments will be feasible using Micronet in areas such as SDR position estimation, antenna diversity, lightweight DSA algorithms, and coexistence with CORNET. Once the Raspberry Pi has been further matured as an SDR transmitter, it will open up other possibilities for experiments involving data throughput, BER, and channel coding, for example.

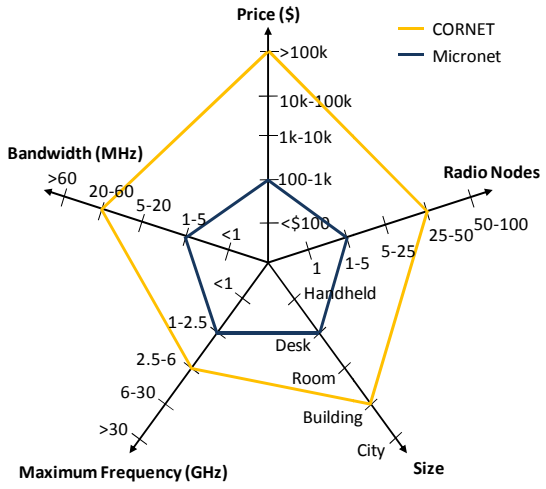


Fig 4. Micronet versus CORNET.

5. CONCLUSIONS

Micronet proves that it is possible to build a CR testbed at low cost. It demonstrates the capabilities, but also the limitations of using inexpensive SDR hardware for CR applications. These limitations include bandwidth, processing power, and spatial constraints. Several methods for circumventing these limitations were discussed and include the use of low cost processors attached to each node to reduce the processing burden on the central PC, allow for greater spatial separation of the nodes, and some degree of distributed computing between nodes. An additional objective of Micronet is to create an initial reference for leveraging CR research and education while not relying on expensive equipment. We will continue to develop Micronet by experimenting with other hardware platforms and software tools, developing additional demonstrations, and making resources and tools available as open source for others to use. Micronet is an alternative to large-scale testbeds, such as CORNET. Figure 4 quantifies the differences, considering five key attributes. Current work includes de-

veloping a GNU Radio block to utilize the Raspberry Pi as an FM transmitter along with additional experiments and demonstrations that include antenna diversity, directional antennas, and position estimation.

6. REFERENCES

[1] J. Mitola, "The software radio architecture," *IEEE Commun. Mag.*, vol. 33, no. 5, pp. 26–38, May 1995.

[2] J. Mitola, III, "Cognitive radio: An integrated agent architecture for software defined radio," Ph.D. dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.

[3] D. Raychaudhuri, et al., "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," *Proc. IEEE Wireless Communications and Networking Conf. (WCNC 2005)*, pp. 1644-1669.

[4] S. Gupta, C. Hunter, P. Murphy, and A. Sabharwal, "WARPnet: clean slate research on deployed wireless networks," *Proc. 10th ACM Int. Symp. Mobile Ad Hoc Networking (MibiHoc'09)*, pp. 31-332.

[5] M. Hibler, *et al.*, "Large-scale Virtualization in the Emulab Network Testbed," *Proc. 2008 USENIX Annual Technical Conference*, Boston, MA, 2008, pp. 113-128.

[6] VT-CORNET Web Site, http://cornet.wireless.vt.edu/trac/

[7] Michael Lustig, "RTL-SDR: Inexpensive Software Defined Radio," EE123: Digital Signal Processing, Fall 2012, Dept. Electrical Engineering and Computer Science, UC Berkeley, http://inst.eecs.berkeley.edu/~ee123/fa12/rtl_sdr.html

[8] C. R. A. Gonzalez, *et al.*, "Open-source SCA-based core framework and rapid development tools enable software-defined radio education and research," *IEEE Commun. Mag.*, Vol. 47, Iss. 10, Oct. 2009, pp. 48-55.

[9] P. D. Sutton, *et al.*, "Iris: An Architecture for Cognitive Radio Networking Testbeds," IEEE Commu. Mag., Vol. 48, Iss. 9, pp. 114-122, Sept. 2010.

[10] I. Gomez, V. Marojevic, A. Gelonch, "ALOE: an open-source SDR execution environment with cognitive computing resource management capabilities," *IEEE Commun. Mag.*, Vol. 49, Iss. 9, pp. 76-83, Sept. 2011.

[11] Distributed Real Time Framework for Software-Defined Radio (SDR), https://github.com/flexnets/aloe

[12] GNU Radio project web site, http://gnuradio.org

[13] Dell Latitude 2100 data sheet, http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/laptop-latitude-2100-specsheet.pdf

[14] R820T data sheet, http://superkuh.com/gnuradio/R820T_datasheet-Non_R-20111130_unlocked.pdf

[15] E4000 data sheet, http://www.superkuh.com/gnuradio/Elonics-E4000-Low-Power-CMOS-Multi-Band-Tunner-Datasheet.pdf

[16] OpenBTS Web Site, http://openbts.org

[17] openLTE – An Open Source 3GPP LTE Implementation, http://sourceforge.net/projects/openlte

[18] HackRF - http://greatscottgadgets.com/hackrf/

[19] BladeRF - http://nuand.com/

[20] PiFM - http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter

[21] REDHAWK - http://redhawksdr.github.io/Documentation/

[22] Liquid-dsp: software-defined radio digital signal processing library, http://liquidsdr.org/

[23] RTLSDR Scanner - http://eartoearoak.com/software/rtlsdr-scanner

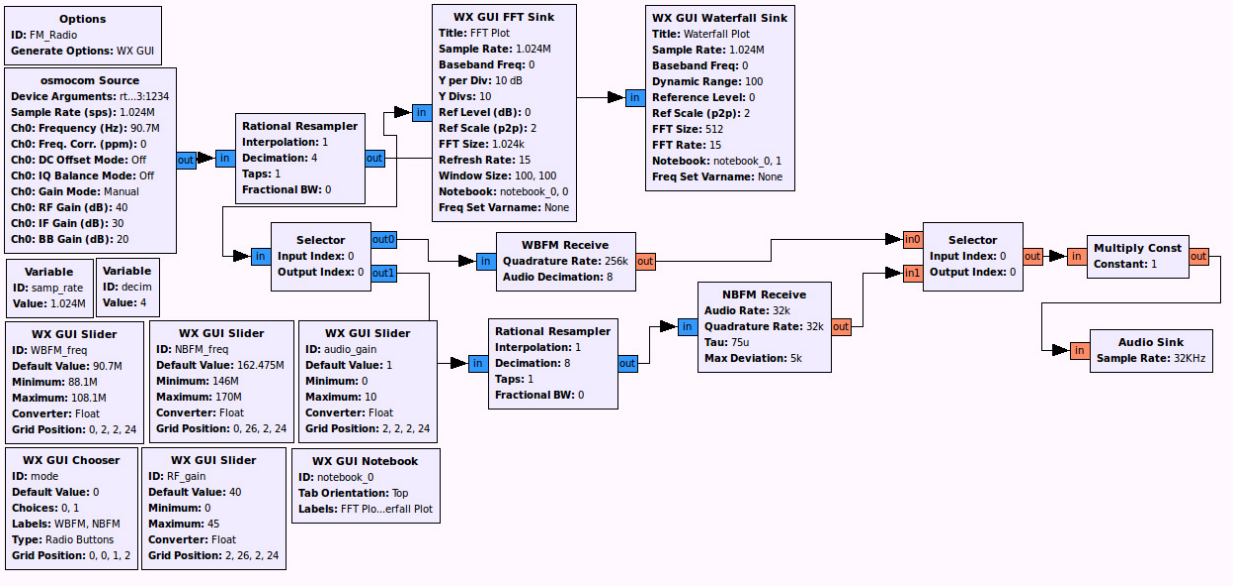


Fig. 5. Interactive RTL-SDR FM radio receiver flow graph (*RTL_FM_Radio.grc*).

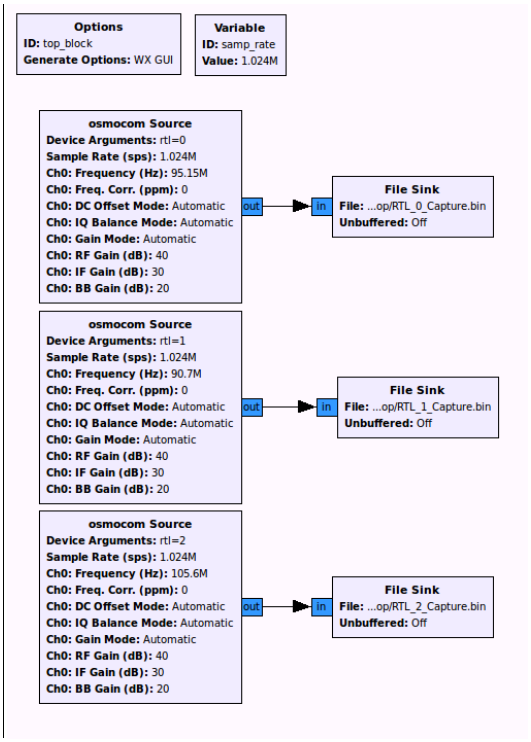


Fig. 6. 3x RTL-SDR data capture (*3x_RTL_Sense.grc*).

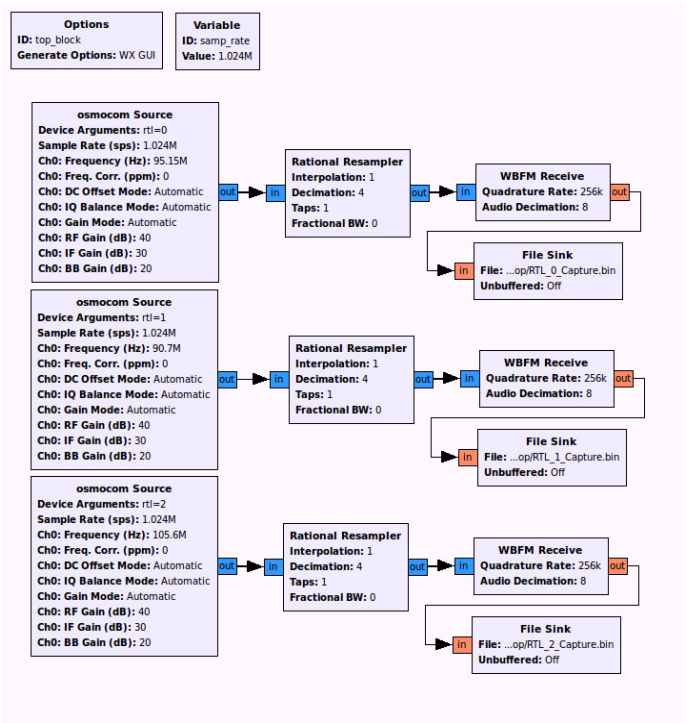


Fig. 7. 3x RTL-SDR FM receiver and data (*3x_RTL_FM_Demodulation.grc*).