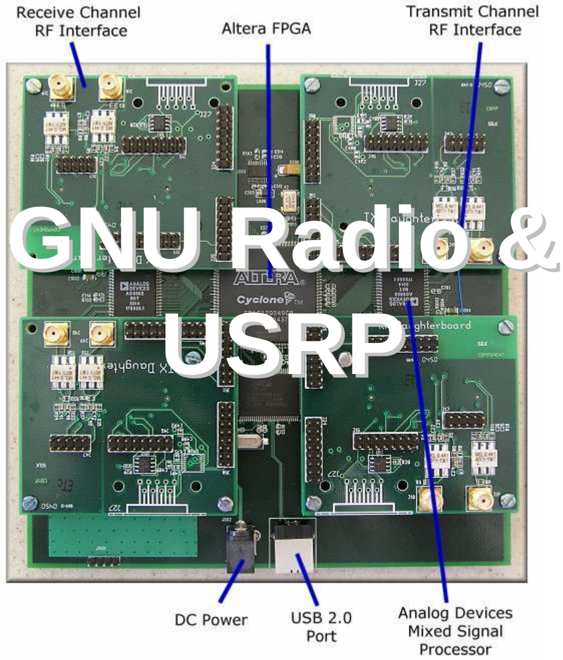
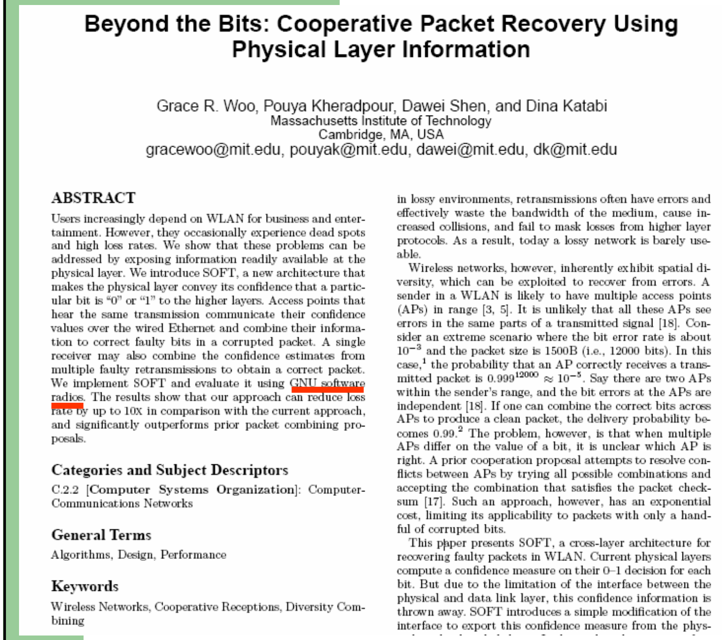
GNU Radio & USRP


Receive Channel RF Interface
Altera FPGA
Transmit Channel RF Interface
DC Power
USB 2.0 Port
Analog Devices Mixed Signal Processor



Beyond the Bits: Cooperative Packet Recovery Using Physical Layer Information

Grace R. Woo, Pouya Kheradpour, Dawei Shen, and Dina Katabi
Massachusetts Institute of Technology
Cambridge, MA, USA
gracewoo@mit.edu, pouyak@mit.edu, dawei@mit.edu, dk@mit.edu

ABSTRACT

Users increasingly depend on WLAN for business and entertainment. However, they occasionally experience dead spots and high loss rates. We show that these problems can be addressed by exposing information readily available at the physical layer. We introduce SOFT, a new architecture that makes the physical layer convey its confidence that a particular bit is "0" or "1" to the higher layers. Access points that hear the same transmission communicate their confidence values over the wired Ethernet and combine their information to correct faulty bits in a corrupted packet. A single receiver may also combine the confidence estimates from multiple faulty retransmissions to obtain a correct packet. We implement SOFT and evaluate it using GNU software radios. The results show that our approach can reduce loss rate by up to 10X in comparison with the current approach, and significantly outperforms prior packet combining proposals.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communications Networks

General Terms

Algorithms, Design, Performance

Keywords

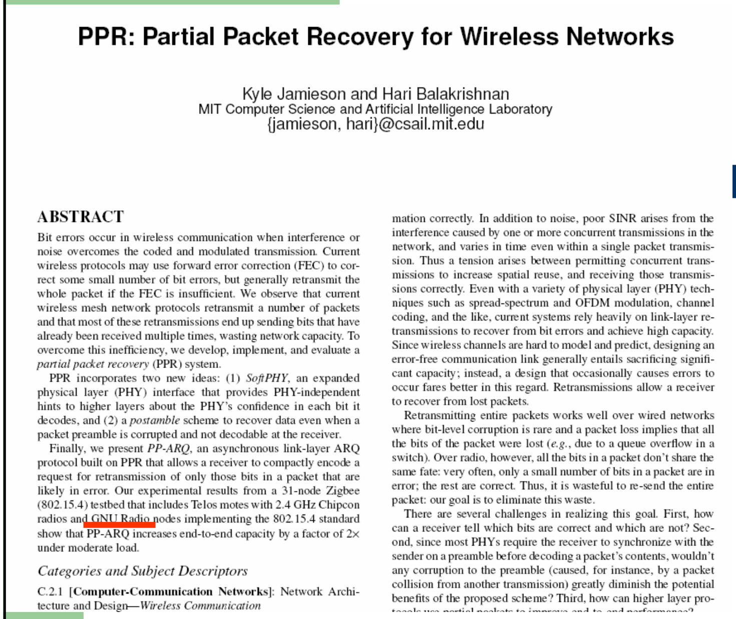
Wireless Networks, Cooperative Receptions, Diversity Combining

in lossy environments, retransmissions often have errors and effectively waste the bandwidth of the medium, cause increased collisions, and fail to mask losses from higher layer protocols. As a result, today a lossy network is barely useable.

Wireless networks, however, inherently exhibit spatial diversity, which can be exploited to recover from errors. A sender in a WLAN is likely to have multiple access points (APs) in range [3, 5]. It is unlikely that all these APs see errors in the same parts of a transmitted signal [18]. Consider an extreme scenario where the bit error rate is about 10^{-3} and the packet size is 1500B (i.e., 12000 bits). In this case,[1] the probability that an AP correctly receives a transmitted packet is $0.999^{12000} \approx 10^{-5}$. Say there are two APs within the sender's range, and the bit errors at the APs are independent [18]. If one can combine the correct bits across APs to produce a clean packet, the delivery probability becomes 0.99.[2] The problem, however, is that when multiple APs differ on the value of a bit, it is unclear which AP is right. A prior cooperation proposal attempts to resolve conflicts between APs by trying all possible combinations and accepting the combination that satisfies the packet checksum [17]. Such an approach, however, has an exponential cost, limiting its applicability to packets with only a handful of corrupted bits.

This paper presents SOFT, a cross-layer architecture for recovering faulty packets in WLAN. Current physical layers compute a confidence measure on their 0–1 decision for each bit. But due to the limitation of the interface between the physical and data link layer, this confidence information is thrown away. SOFT introduces a simple modification of the interface to export this confidence measure from the phys-

ACM MobiCom'07



PPR: Partial Packet Recovery for Wireless Networks

Kyle Jamieson and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Laboratory
{jamieson, hari}@csail.mit.edu

ACM SigComm’07

ABSTRACT

Bit errors occur in wireless communication when interference or noise overcomes the coded and modulated transmission. Current wireless protocols may use forward error correction (FEC) to correct some small number of bit errors, but generally retransmit the whole packet if the FEC is insufficient. We observe that current wireless mesh network protocols retransmit a number of packets and that most of these retransmissions end up sending bits that have already been received multiple times, wasting network capacity. To overcome this inefficiency, we develop, implement, and evaluate a *partial packet recovery* (PPR) system.

PPR incorporates two new ideas: (1) *SoftPHY*, an expanded physical layer (PHY) interface that provides PHY-independent hints to higher layers about the PHY’s confidence in each bit it decodes, and (2) a *postamble* scheme to recover data even when a packet preamble is corrupted and not decodable at the receiver.

Finally, we present *PP-ARQ*, an asynchronous link-layer ARQ protocol built on PPR that allows a receiver to compactly encode a request for retransmission of only those bits in a packet that are likely in error. Our experimental results from a 31-node Zigbee (802.15.4) testbed that includes Telos motes with 2.4 GHz Chipcon radios and GNU Radio nodes implementing the 802.15.4 standard show that PP-ARQ increases end-to-end capacity by a factor of 2× under moderate load.

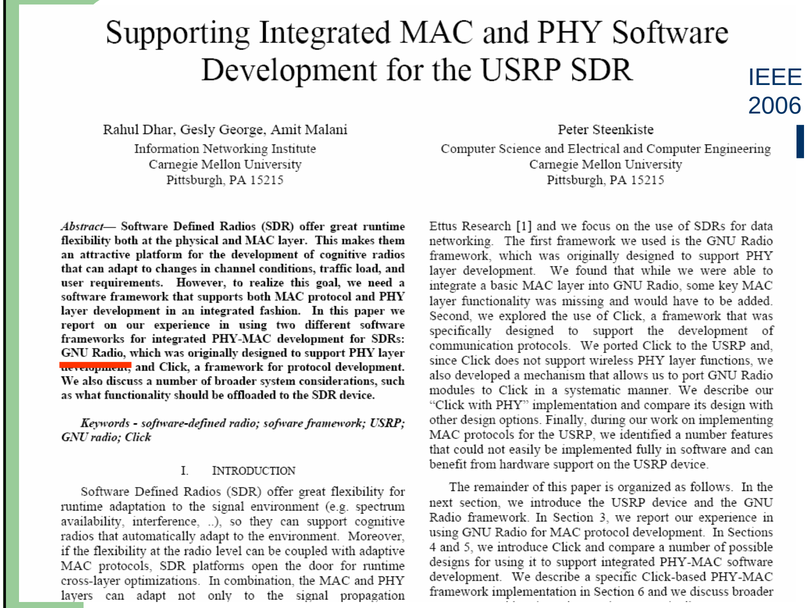
Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

mation correctly. In addition to noise, poor SINR arises from the interference caused by one or more concurrent transmissions in the network, and varies in time even within a single packet transmission. Thus a tension arises between permitting concurrent transmissions to increase spatial reuse, and receiving those transmissions correctly. Even with a variety of physical layer (PHY) techniques such as spread-spectrum and OFDM modulation, channel coding, and the like, current systems rely heavily on link-layer retransmissions to recover from bit errors and achieve high capacity. Since wireless channels are hard to model and predict, designing an error-free communication link generally entails sacrificing significant capacity; instead, a design that occasionally causes errors to occur fares better in this regard. Retransmissions allow a receiver to recover from lost packets.

Retransmitting entire packets works well over wired networks where bit-level corruption is rare and a packet loss implies that all the bits of the packet were lost (*e.g.*, due to a queue overflow in a switch). Over radio, however, all the bits in a packet don’t share the same fate: very often, only a small number of bits in a packet are in error; the rest are correct. Thus, it is wasteful to re-send the entire packet: our goal is to eliminate this waste.

There are several challenges in realizing this goal. First, how can a receiver tell which bits are correct and which are not? Second, since most PHYs require the receiver to synchronize with the sender on a preamble before decoding a packet’s contents, wouldn’t any corruption to the preamble (caused, for instance, by a packet collision from another transmission) greatly diminish the potential benefits of the proposed scheme? Third, how can higher layer protocols use partial packets to improve end-to-end performance?



Supporting Integrated MAC and PHY Software Development for the USRP SDR

IEEE Workshop ..,
2006

Rahul Dhar, Gesly George, Amit Malani
Information Networking Institute
Carnegie Mellon University
Pittsburgh, PA 15215

Peter Steenkiste
Computer Science and Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15215

Abstract— **Software Defined Radios (SDR) offer great runtime flexibility both at the physical and MAC layer. This makes them an attractive platform for the development of cognitive radios that can adapt to changes in channel conditions, traffic load, and user requirements. However, to realize this goal, we need a software framework that supports both MAC protocol and PHY layer development in an integrated fashion. In this paper we report on our experience in using two different software frameworks for integrated PHY-MAC development for SDRs: GNU Radio, which was originally designed to support PHY layer development, and Click, a framework for protocol development. We also discuss a number of broader system considerations, such as what functionality should be offloaded to the SDR device.**

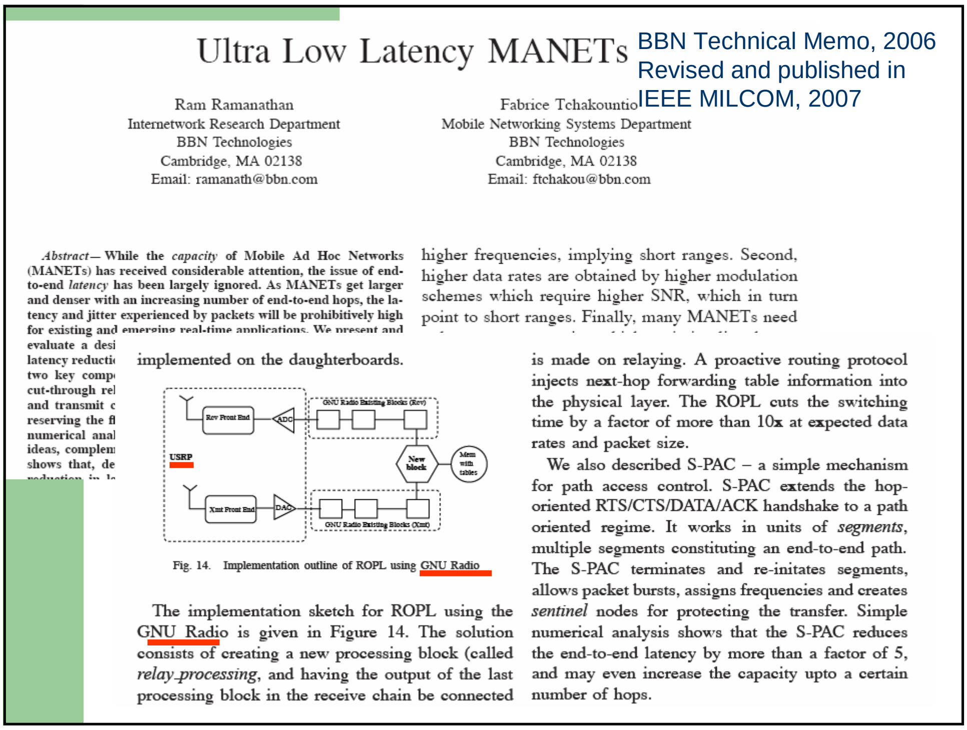
Keywords - software-defined radio; sofware framework; USRP; GNU radio; Click

I. INTRODUCTION

Software Defined Radios (SDR) offer great flexibility for runtime adaptation to the signal environment (e.g. spectrum availability, interference, ..), so they can support cognitive radios that automatically adapt to the environment. Moreover, if the flexibility at the radio level can be coupled with adaptive MAC protocols, SDR platforms open the door for runtime cross-layer optimizations. In combination, the MAC and PHY layers can adapt not only to the signal propagation

Ettus Research [1] and we focus on the use of SDRs for data networking. The first framework we used is the GNU Radio framework, which was originally designed to support PHY layer development. We found that while we were able to integrate a basic MAC layer into GNU Radio, some key MAC layer functionality was missing and would have to be added. Second, we explored the use of Click, a framework that was specifically designed to support the development of communication protocols. We ported Click to the USRP and, since Click does not support wireless PHY layer functions, we also developed a mechanism that allows us to port GNU Radio modules to Click in a systematic manner. We describe our “Click with PHY” implementation and compare its design with other design options. Finally, during our work on implementing MAC protocols for the USRP, we identified a number features that could not easily be implemented fully in software and can benefit from hardware support on the USRP device.

The remainder of this paper is organized as follows. In the next section, we introduce the USRP device and the GNU Radio framework. In Section 3, we report our experience in using GNU Radio for MAC protocol development. In Sections 4 and 5, we introduce Click and compare a number of possible designs for using it to support integrated PHY-MAC software development. We describe a specific Click-based PHY-MAC framework implementation in Section 6 and we discuss broader



BBN Technical Memo, 2006
Revised and published in
IEEE MILCOM, 2007

Ultra Low Latency MANETs

Ram Ramanathan
Internetwork Research Department
BBN Technologies
Cambridge, MA 02138
Email: ramanath@bbn.com

Fabrice Tchakountio
Mobile Networking Systems Department
BBN Technologies
Cambridge, MA 02138
Email: ftchakou@bbn.com

Abstract— While the *capacity* of Mobile Ad Hoc Networks (MANETs) has received considerable attention, the issue of end-to-end *latency* has been largely ignored. As MANETs get larger and denser with an increasing number of end-to-end hops, the latency and jitter experienced by packets will be prohibitively high for existing and emerging real-time applications. We present and evaluate a desi latency reducti two key comp cut-through rel and transmit c reserving the fl numerical anal ideas, complem shows that, de

higher frequencies, implying short ranges. Second, higher data rates are obtained by higher modulation schemes which require higher SNR, which in turn point to short ranges. Finally, many MANETs need

implemented on the daughterboards.

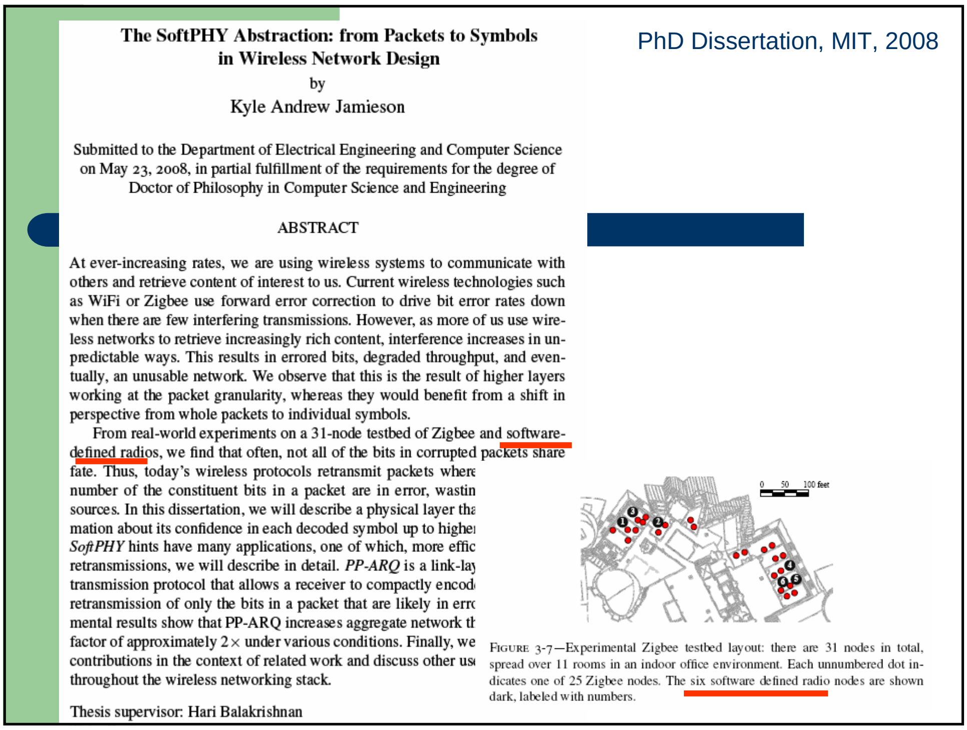
USRP
Rcv Front End
ADC
GNU Radio Existing Blocks (Rcv)
New block
Mem with tables
Xmt Front End
DAC
GNU Radio Existing Blocks (Xmt)

Fig. 14. Implementation outline of ROPL using GNU Radio

The implementation sketch for ROPL using the GNU Radio is given in Figure 14. The solution consists of creating a new processing block (called *relay_processing*, and having the output of the last processing block in the receive chain be connected

is made on relaying. A proactive routing protocol injects next-hop forwarding table information into the physical layer. The ROPL cuts the switching time by a factor of more than 10x at expected data rates and packet size.

We also described S-PAC – a simple mechanism for path access control. S-PAC extends the hop-oriented RTS/CTS/DATA/ACK handshake to a path oriented regime. It works in units of *segments*, multiple segments constituting an end-to-end path. The S-PAC terminates and re-initates segments, allows packet bursts, assigns frequencies and creates *sentinel* nodes for protecting the transfer. Simple numerical analysis shows that the S-PAC reduces the end-to-end latency by more than a factor of 5, and may even increase the capacity upto a certain number of hops.



PhD Dissertation, MIT, 2008

The SoftPHY Abstraction: from Packets to Symbols in Wireless Network Design

by

Kyle Andrew Jamieson

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

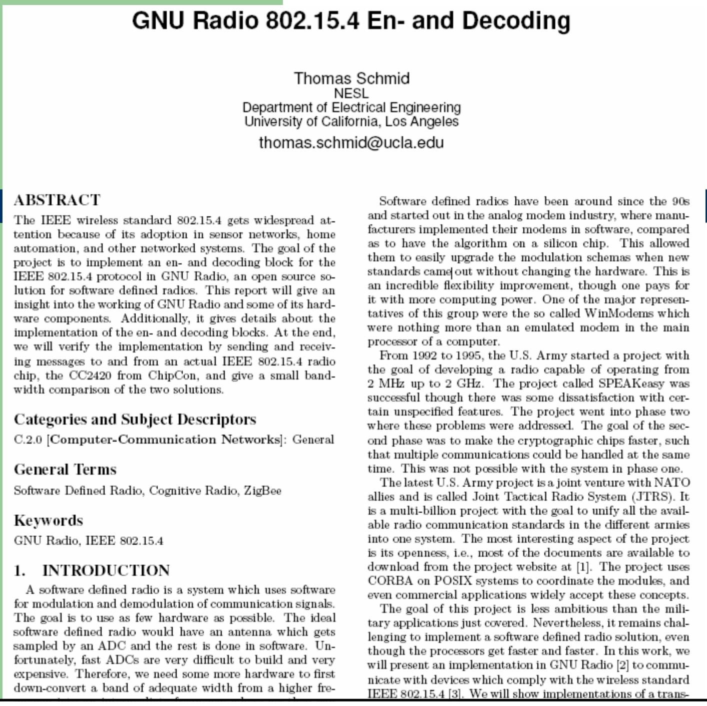
ABSTRACT

At ever-increasing rates, we are using wireless systems to communicate with others and retrieve content of interest to us. Current wireless technologies such as WiFi or Zigbee use forward error correction to drive bit error rates down when there are few interfering transmissions. However, as more of us use wireless networks to retrieve increasingly rich content, interference increases in unpredictable ways. This results in errored bits, degraded throughput, and eventually, an unusable network. We observe that this is the result of higher layers working at the packet granularity, whereas they would benefit from a shift in perspective from whole packets to individual symbols.

From real-world experiments on a 31-node testbed of Zigbee and software-defined radios, we find that often, not all of the bits in corrupted packets share fate. Thus, today's wireless protocols retransmit packets where number of the constituent bits in a packet are in error, wastin sources. In this dissertation, we will describe a physical layer tha mation about its confidence in each decoded symbol up to higher *SoftPHY* hints have many applications, one of which, more effic retransmissions, we will describe in detail. *PP-ARQ* is a link-lay transmission protocol that allows a receiver to compactly encod retransmission of only the bits in a packet that are likely in erro mental results show that PP-ARQ increases aggregate network th factor of approximately 2× under various conditions. Finally, we contributions in the context of related work and discuss other us throughout the wireless networking stack.

Thesis supervisor: Hari Balakrishnan

FIGURE 3-7—Experimental Zigbee testbed layout: there are 31 nodes in total, spread over 11 rooms in an indoor office environment. Each unnumbered dot indicates one of 25 Zigbee nodes. The six software defined radio nodes are shown dark, labeled with numbers.



GNU Radio 802.15.4 En- and Decoding

Thomas Schmid
NESL
Department of Electrical Engineering
University of California, Los Angeles
thomas.schmid@ucla.edu

Project Report, UCLA, 2006

ABSTRACT

The IEEE wireless standard 802.15.4 gets widespread attention because of its adoption in sensor networks, home automation, and other networked systems. The goal of the project is to implement an en- and decoding block for the IEEE 802.15.4 protocol in GNU Radio, an open source solution for software defined radios. This report will give an insight into the working of GNU Radio and some of its hardware components. Additionally, it gives details about the implementation of the en- and decoding blocks. At the end, we will verify the implementation by sending and receiving messages to and from an actual IEEE 802.15.4 radio chip, the CC2420 from ChipCon, and give a small bandwidth comparison of the two solutions.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General

General Terms

Software Defined Radio, Cognitive Radio, ZigBee

Keywords

GNU Radio, IEEE 802.15.4

1. INTRODUCTION

A software defined radio is a system which uses software for modulation and demodulation of communication signals. The goal is to use as few hardware as possible. The ideal software defined radio would have an antenna which gets sampled by an ADC and the rest is done in software. Unfortunately, fast ADCs are very difficult to build and very expensive. Therefore, we need some more hardware to first down-convert a band of adequate width from a higher fre-

Software defined radios have been around since the 90s and started out in the analog modem industry, where manufacturers implemented their modems in software, compared as to have the algorithm on a silicon chip. This allowed them to easily upgrade the modulation schemas when new standards came out without changing the hardware. This is an incredible flexibility improvement, though one pays for it with more computing power. One of the major representatives of this group were the so called WinModems which were nothing more than an emulated modem in the main processor of a computer.

From 1992 to 1995, the U.S. Army started a project with the goal of developing a radio capable of operating from 2 MHz up to 2 GHz. The project called SPEAKeasy was successful though there was some dissatisfaction with certain unspecified features. The project went into phase two where these problems were addressed. The goal of the second phase was to make the cryptographic chips faster, such that multiple communications could be handled at the same time. This was not possible with the system in phase one.

The latest U.S. Army project is a joint venture with NATO allies and is called Joint Tactical Radio System (JTRS). It is a multi-billion project with the goal to unify all the available radio communication standards in the different armies into one system. The most interesting aspect of the project is its openness, i.e., most of the documents are available to download from the project website at [1]. The project uses CORBA on POSIX systems to coordinate the modules, and even commercial applications widely accept these concepts.

The goal of this project is less ambitious than the military applications just covered. Nevertheless, it remains challenging to implement a software defined radio solution, even though the processors get faster and faster. In this work, we will present an implementation in GNU Radio [2] to communicate with devices which comply with the wireless standard IEEE 802.15.4 [3]. We will show implementations of a trans-

HOCHSCHULE LUZERN
Engineering & Architecture

Lucerne University of Applied Sciences and Arts

CEESAR - Center of Excellence for Embedded Systems Applied Research

KTI/CTI
HASLERSTIFTUNG

Implementation of an IEEE 802.15.4 Transceiver with a Software-defined Radio setup

Stefan Knauth
Lucerne University of Applied Sciences
CEESAR – Center of Excellence for Embedded Systems Applied Research
Technikumstr. 21
CH 6048 Horw / Switzerland
stefan.knauth@gmx.ch
www.ceesar.ch

Abs

An implementation of an IEEE802.15.4
technology on a standard PC has bee
USRP, a commercially available softw
GnuRadio software framework. The soft
is connected via USB2.0. The setup is
and for a deep understanding of the IEE
engineering students. The implemente
gain adjustment, a differential phase de
data frame interface for upper softwa
channels synchronously. The software i
the frame detection and synchronisatio
correlator. It operates with an acquisi
consisting of two IQ values.

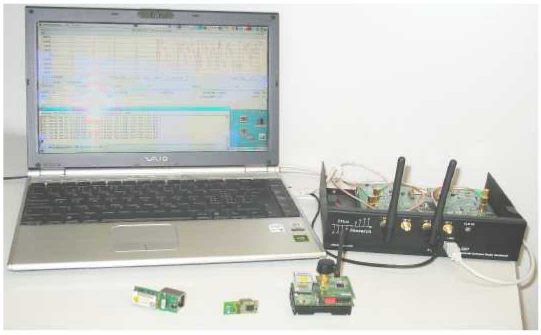
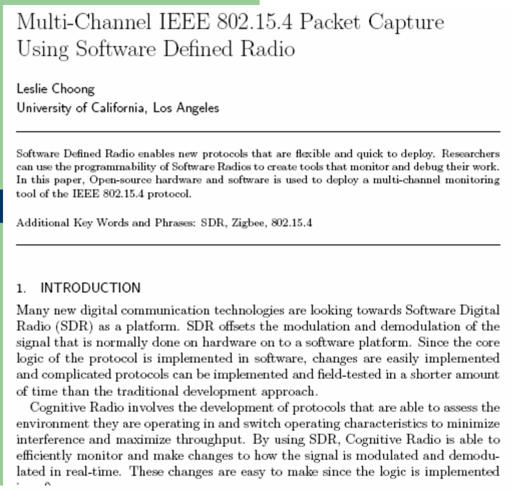


Fig. 1: Software defined radio setup: Notebook (left), SDR hardware (USRP [1], right) and some IEEE802.15.4 / ZigBee modules [2] (foreground)

Introduction

Wireless functionality is becoming more and n
embedded systems. As a designer, one typica
own application or one uses a highly integr



Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio

Leslie Choong
University of California, Los Angeles

Software Defined Radio enables new protocols that are flexible and quick to deploy. Researchers can use the programmability of Software Radios to create tools that monitor and debug their work. In this paper, Open-source hardware and software is used to deploy a multi-channel monitoring tool of the IEEE 802.15.4 protocol.

Additional Key Words and Phrases: SDR, Zigbee, 802.15.4

1. INTRODUCTION

Many new digital communication technologies are looking towards Software Digital Radio (SDR) as a platform. SDR offsets the modulation and demodulation of the signal that is normally done on hardware on to a software platform. Since the core logic of the protocol is implemented in software, changes are easily implemented and complicated protocols can be implemented and field-tested in a shorter amount of time than the traditional development approach.

Cognitive Radio involves the development of protocols that are able to assess the environment they are operating in and switch operating characteristics to minimize interference and maximize throughput. By using SDR, Cognitive Radio is able to efficiently monitor and make changes to how the signal is modulated and demodulated in real-time. These changes are easy to make since the logic is implemented

UCLA Networked & Embedded Sensing Lab, 03 2009

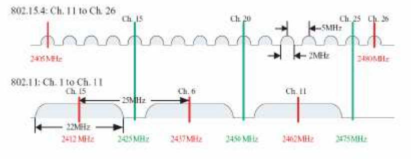


Fig. 2. Spectrum Overlap of WLAN and IEEE 802.15.4

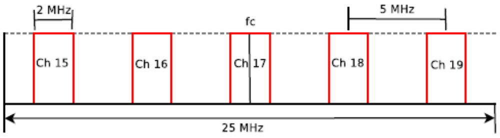


Fig. 7. Tuning the USRP2 to capture 5 channels. Center frequency denoted by fc

Radio Evolution

Conventional Radio

- Traditional RF Design
- Traditional Baseband Design

Software Radio

- Conventional Radio +
- Software Architecture
- Reconfigurability
- Provisions for easy upgrades

Cognitive Radio

- SDR +
- Intelligence
- Awareness
- Learning
- Observations

Software Defined Radio (SDR)

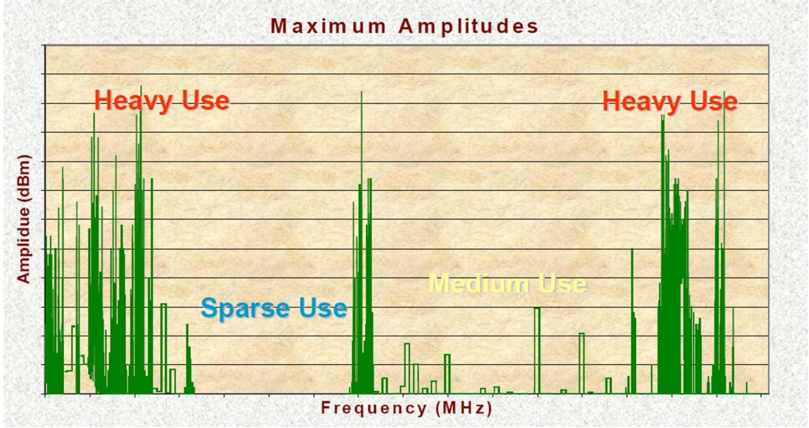
- Termed coined by Mitola in 1992
- Radio’s physical layer behavior is primarily defined in software
- Accepts fully programmable traffic & control information
- Supports broad range of frequencies, air interfaces, and application software
- Changes its initial configuration to satisfy user requirements

Cognitive Radio

- Term coined by Mitola in 1999
- Mitola’s definition:
  - Software radio that is aware of its environment and its capabilities
  - Alters its physical layer behavior
  - Capable of following complex adaptation strategies
- “A radio or system that senses, and is aware of, its operational environment and can dynamically and ***autonomously*** adjust its radio operating parameters accordingly”
- Learns from previous experiences
- Deals with situations not planned at the initial time of design

Cognitive radios are a powerful tool for solving two major problems:

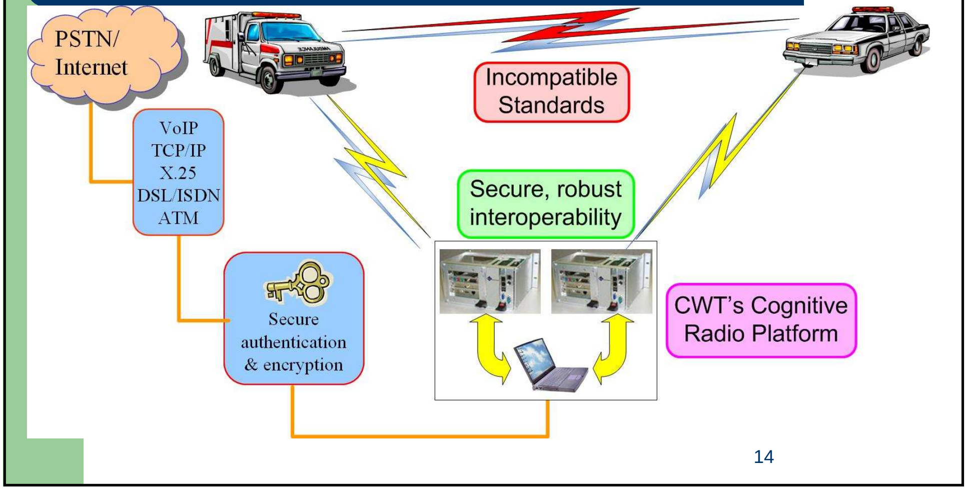
1) Access to spectrum (finding an open frequency and using it)



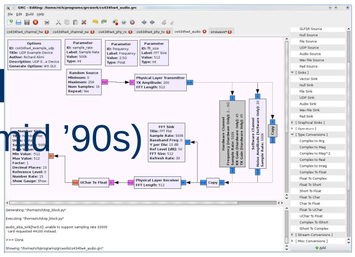
Source: FCC, Spectrum Policy Task Force, Technology Advisory Council (TAC) Briefing (December 2002).

Cognitive radios are a powerful tool for solving two major problems:

2) Interoperability (talking to legacy radios using a variety of incompatible waveforms)

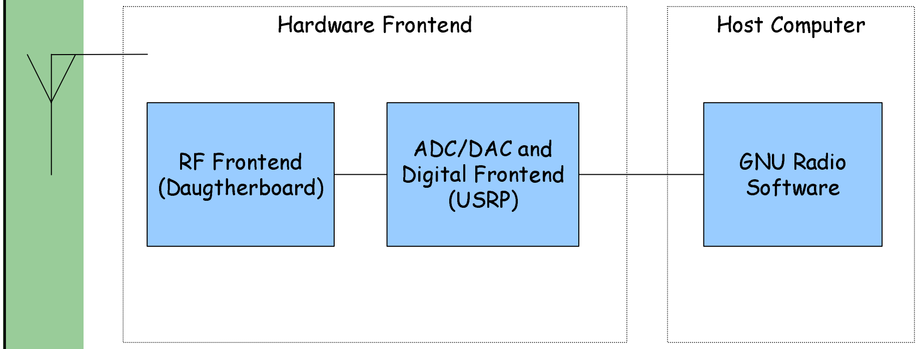


GNU Radio



- Origin: MIT’s SpectrumWare (mid ’90s)
- Software toolkit for signal processing
  - Software radio construction
  - Rapid development
  - Cognitive radio
- USRP (Universal Software Radio Peripheral)
  - Hardware frontend for sending and receiving waveforms

GNU Radio Components



Hardware Frontend

RF Frontend (Daugtherboard)

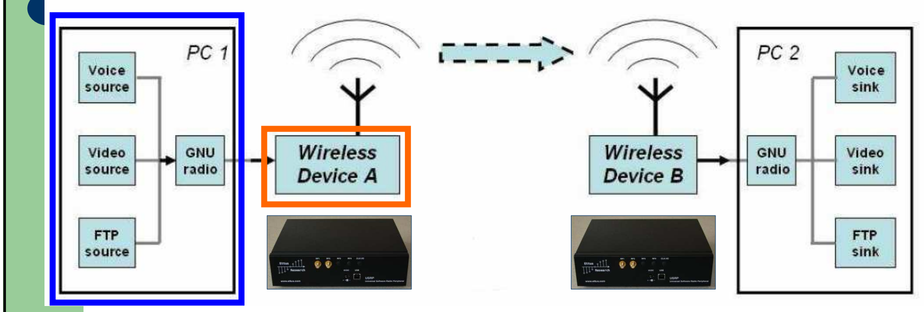
ADC/DAC and Digital Frontend (USRP)

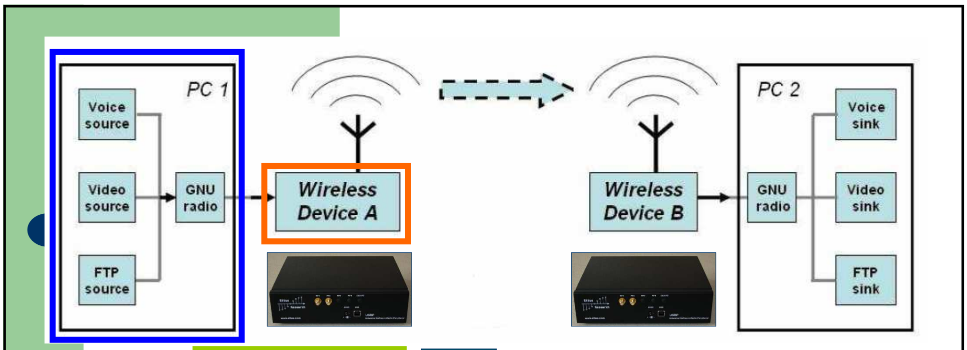
Host Computer

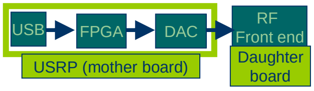
GNU Radio Software

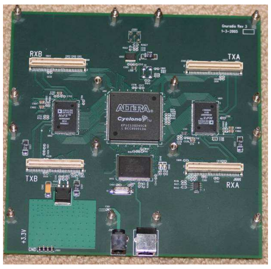
http://mobiledevices.kom.aau.dk/fileadmin/mobiledevices/teaching/software_testing/Gnu_radio_lecture.pdf

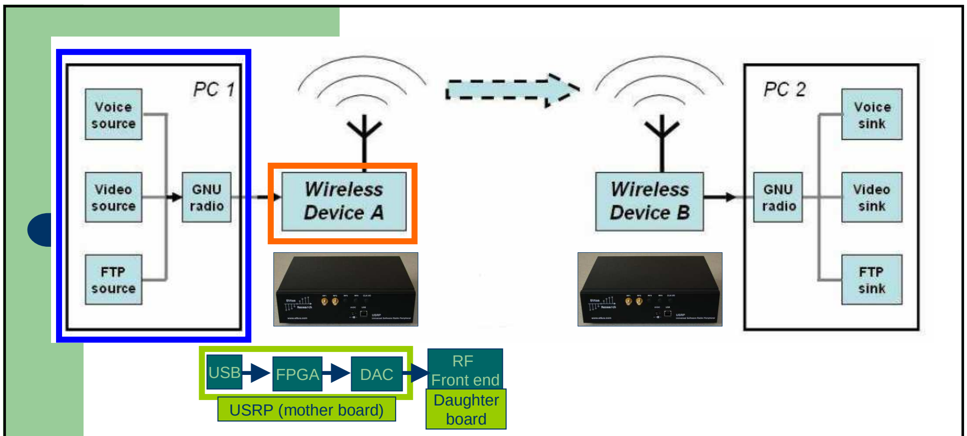
Architecture – overall





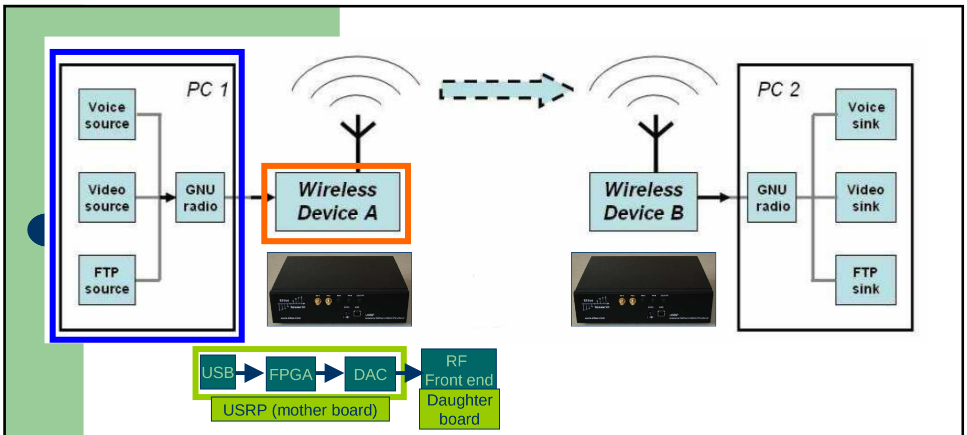






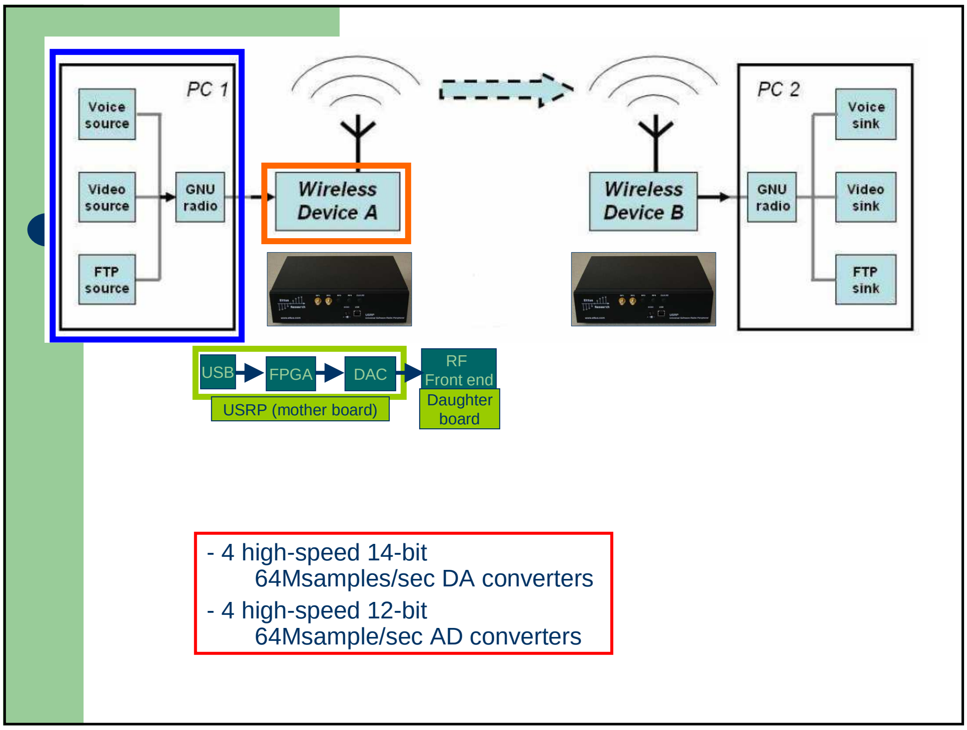
Support USB 2.0 at this stage, USB 1.x is not supported at all

1. Support 60MB/sec across the USB.
2. All samples sent over the USB interface are in 16-bit signed integers in IQ format, 16-bit I and 16-bit Q data (complex), resulting in 15 Msamples/sec across the USB.



Includes four digital down converters (DDC) and four digital up counters (DUC)

- Shift frequency from the baseband to the required frequency
- DDCs on the receiver side
- DUCs on the transmit side - actually contained in the AD9862 CODEC chips
- The only transmit signal processing blocks in the FPGA are the interpolators



PC 1

Voice source

Video source

FTP source

GNU radio

Wireless Device A

Wireless Device B

GNU radio

PC 2

Voice sink

Video sink

FTP sink

USB

FPGA

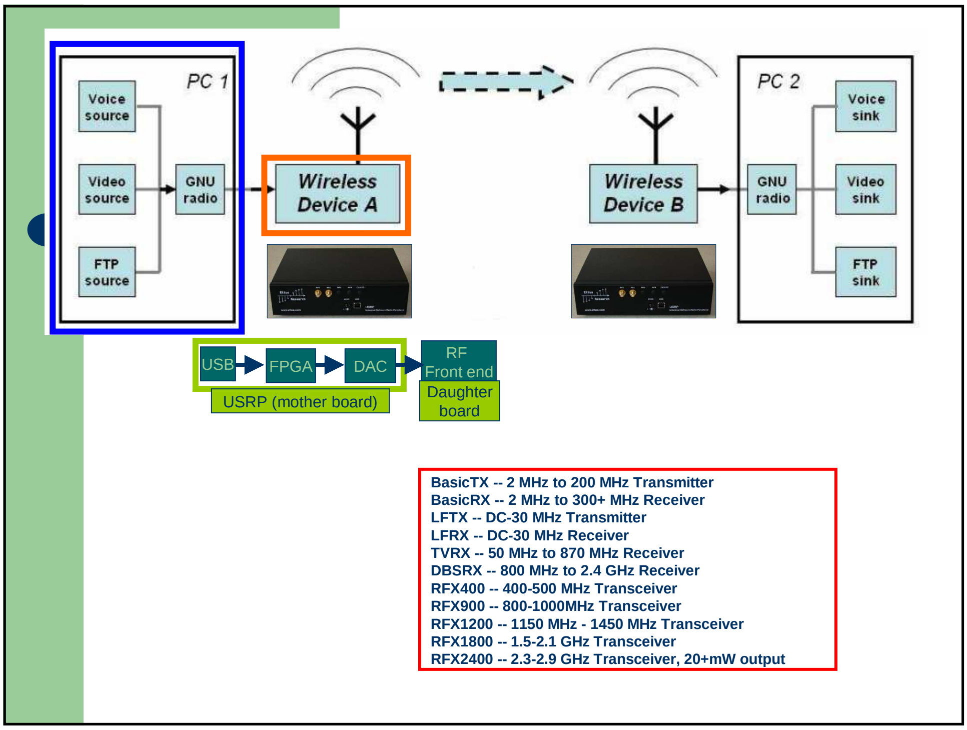
DAC

RF Front end

USRP (mother board)

Daughter board

- 4 high-speed 14-bit 64Msamples/sec DA converters
- 4 high-speed 12-bit 64Msample/sec AD converters



PC 1

Voice source

Video source

FTP source

GNU radio

Wireless Device A

Wireless Device B

GNU radio

PC 2

Voice sink

Video sink

FTP sink

USB

FPGA

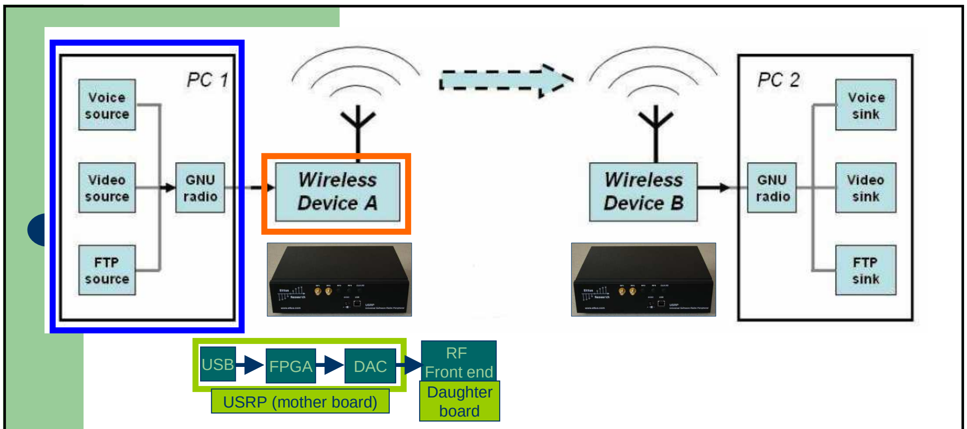
DAC

RF Front end

USRP (mother board)

Daughter board

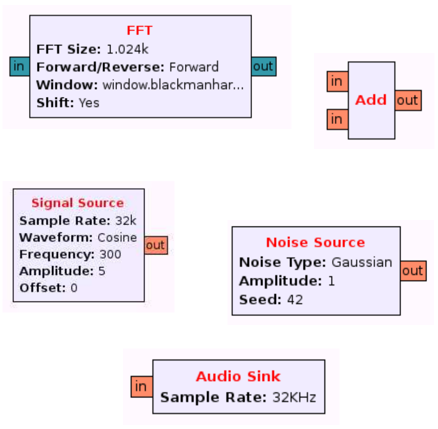
BasicTX -- 2 MHz to 200 MHz Transmitter
BasicRX -- 2 MHz to 300+ MHz Receiver
LFTX -- DC-30 MHz Transmitter
LFRX -- DC-30 MHz Receiver
TVRX -- 50 MHz to 870 MHz Receiver
DBSRX -- 800 MHz to 2.4 GHz Receiver
RFX400 -- 400-500 MHz Transceiver
RFX900 -- 800-1000MHz Transceiver
RFX1200 -- 1150 MHz - 1450 MHz Transceiver
RFX1800 -- 1.5-2.1 GHz Transceiver
RFX2400 -- 2.3-2.9 GHz Transceiver, 20+mW output



GNU Radio has provided some useful APIs such as modulation, demodulation, filtering, etc.

Basics: Blocks

- Signal Processing Block
  - Accepts 0 or more *input streams*
  - Produces 0 or more *output streams*
- Source: No input
  - `noise_source, signal_source, usrp_source`
- Sink: No outputs
  - `audio_alsa_sink, usrp_sink`



FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes

Add

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 300
Amplitude: 5
Offset: 0

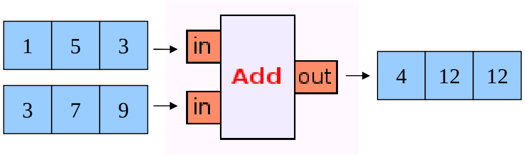
Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 42

Audio Sink
Sample Rate: 32KHz

http://zoo.cs.yale.edu/classes/cs434/

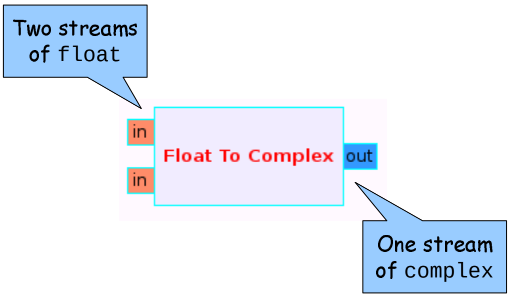
Basics: Data Streams

- Blocks operate on *streams* of data



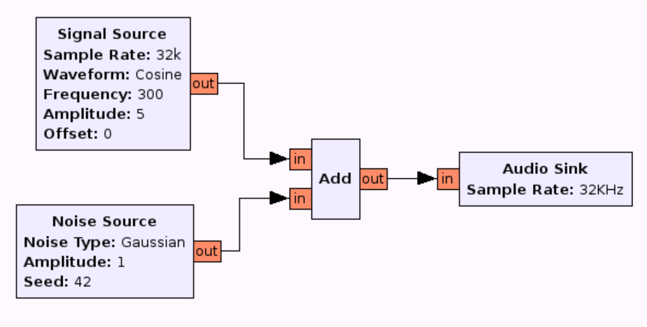
Basics: Data Types

- Blocks operate on certain data types
  - `char`, `short`, `int`, `float`, `complex`
  - Vectors
- *Input Signature*:
  - Data types for input streams
- *Output Signature*:
  - Data types for output streams



Basics: Flow Graph

- Blocks composed as a *flow graph*
  - Data stream flowing from *sources* to *sinks*



Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 300
Amplitude: 5
Offset: 0

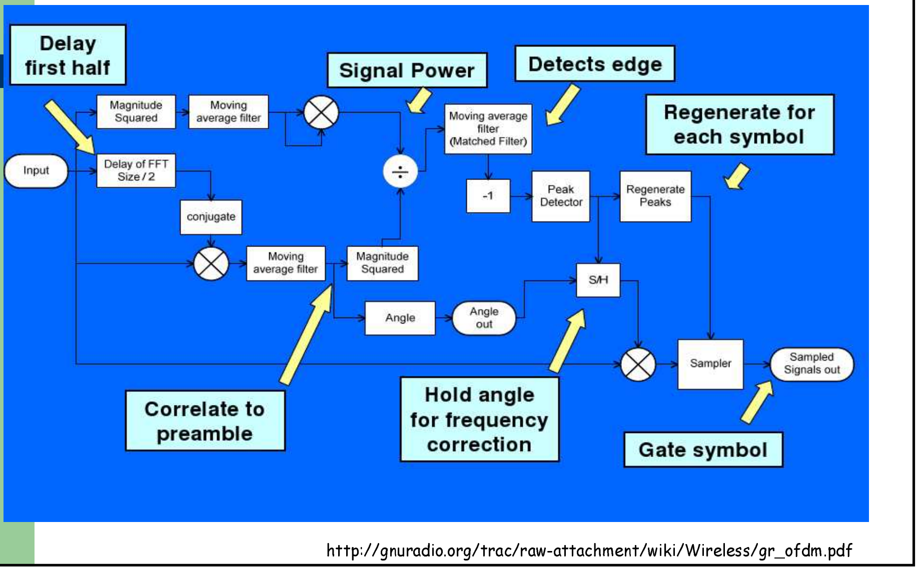
Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 42

Add

Audio Sink
Sample Rate: 32KHz

http://zoo.cs.yale.edu/classes/cs434/

Example: OFDM Synchronizer



Delay first half

Signal Power

Detects edge

Regenerate for each symbol

Correlate to preamble

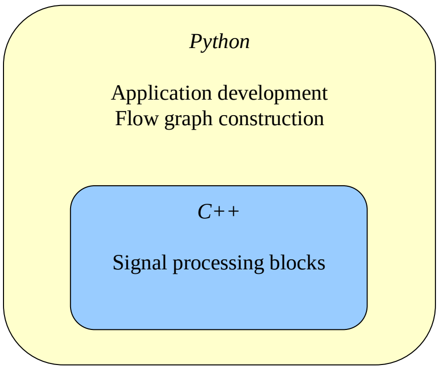
Hold angle for frequency correction

Gate symbol

http://gnuradio.org/trac/raw-attachment/wiki/Wireless/gr_ofdm.pdf

Development Architecture

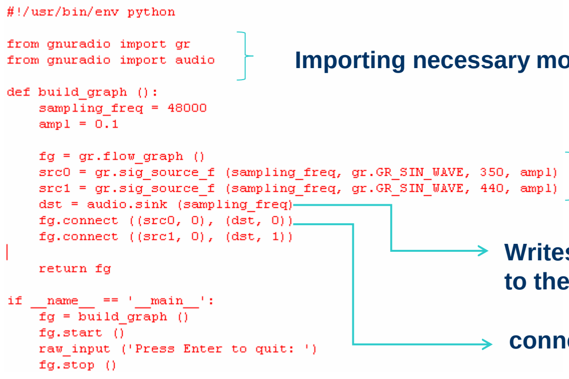
- Python
  - Application management (e.g., GUI)
  - Flow graph construction
  - Non-streaming code (e.g., MAC-layer)
- C++
  - Signal processing blocks
  - Certain routines also coded in assembly



http://mobiledevices.kom.aau.dk/fileadmin/mobiledevices/teaching/software_testing/Gnu_radio_lecture.pdf

Dial Tone Example

- **Generates two sine waves and outputs them to the sound card**



```
#!/usr/bin/env python

from gnuradio import gr
from gnuradio import audio

def build_graph ():
    sampling_freq = 48000
    ampl = 0.1

    fg = gr.flow_graph ()
    src0 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 350, ampl)
    src1 = gr.sig_source_f (sampling_freq, gr.GR_SIN_WAVE, 440, ampl)
    dst = audio.sink (sampling_freq)
    fg.connect ((src0, 0), (dst, 0))
    fg.connect ((src1, 0), (dst, 1))
|
    return fg

if __name__ == '__main__':
    fg = build_graph ()
    fg.start ()
    raw_input ('Press Enter to quit: ')
    fg.stop ()
```

Importing necessary module

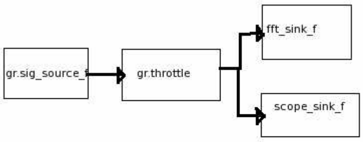
Generates two sine waves

Writes sampling _freq input to the sound card

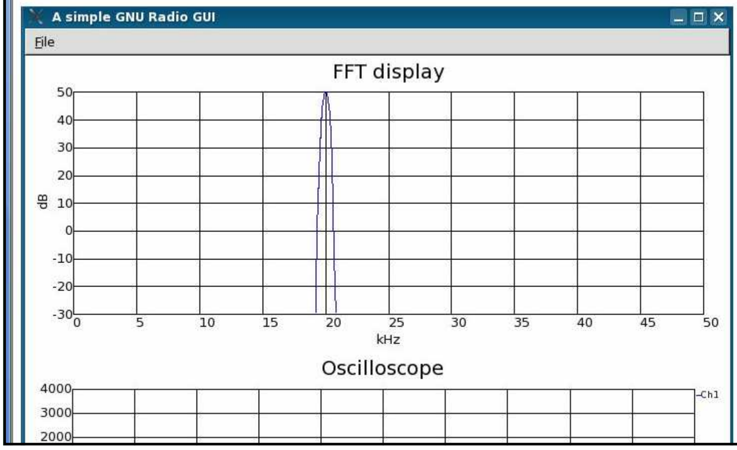
connect the blocks together

http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html

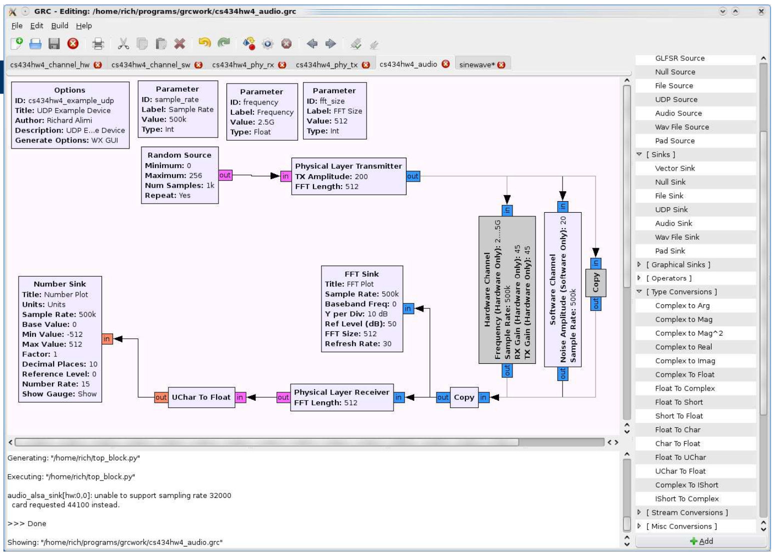
If you won't copy and paste you can download the full source code here: simplegui.py



Now, you should be able to run the program and get a display like this:



GRC (GNU Radio Companion)



Options
ID: cs434hw4_example_udp
Title: UDP Example Device
Author: Richard Alimi
Description: UDP E...e Device
Generate Options: WX GUI

Parameter
ID: sample_rate
Label: Sample Rate
Value: 500k
Type: Int

Parameter
ID: frequency
Label: Frequency
Value: 2.5G
Type: Float

Parameter
ID: fft_size
Label: FFT Size
Value: 512
Type: Int

Random Source
Minimum: 0
Maximum: 256
Num Samples: 1k
Repeat: Yes

Physical Layer Transmitter
TX Amplitude: 200
FFT Length: 512

Hardware Channel
Frequency (Hardware Only): 2....5G
Sample Rate: 500k
RX Gain (Hardware Only): 45
TX Gain (Hardware Only): 45

Software Channel
Noise Amplitude (Software Only): 20
Sample Rate: 500k

FFT Sink
Title: FFT Plot
Sample Rate: 500k
Baseband Freq: 0
Y per Div: 10 dB
Ref Level (dB): 50
FFT Size: 512
Refresh Rate: 30

Number Sink
Title: Number Plot
Units: Units
Sample Rate: 500k
Base Value: 0
Min Value: -512
Max Value: 512
Factor: 1
Decimal Places: 10
Reference Level: 0
Number Rate: 15
Show Gauge: Show

UChar To Float

Physical Layer Receiver
FFT Length: 512

```
Generating: "/home/rich/top_block.py"

Executing: "/home/rich/top_block.py"

audio_alsa_sink[hw:0,0]: unable to support sampling rate 32000
  card requested 44100 instead.

>>> Done

Showing: "/home/rich/programs/grcwork/cs434hw4_audio.grc"
```

Lab Report #3 (Due Sep. 18)

- Install GNU Radio (Gnuradio-3.2.2 stable version) and GRC (part of the standard package)
- Read on Python
  - http://www.python.org/doc/essays/blurb/
  - http://heather.cs.ucdavis.edu/~matloff/Python/PythonIntro.pdf
- Read on GNU Radio
  - http://spectrum.ieee.org/oct06/4654
  - http://dev.emcelettronica.com/gnu-radio-open-source-software-defined-radio
- Optional reading (main information source on GNU Radio and Python)
  - http://gnuradio.org/trac
  - https://radioware.nd.edu/documentation
  - https://www.cgran.org/
  - http://www.python.org/
  - http://www.python.or.kr/

- Report: Discuss what you learned and what you want to know more in less than 2 pages.

Lab Report #4 (Due Sep. 25)

- Try these and report your experience (screen shot if possible)
  - Make a simple GNU Radio GUI (http://www.funwithelectronics.com/?id=10)
  - Dial tone example (Example 1 of http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html)
  - Dial tone example with GRC
  - Any additional GRC example
  - Any additional GNU Radio example

- Discuss what you have learned
- Start early; you'll encounter a number of issues (hopefully not many) during the installation of GNU Radio.

Project

- Sep. 24 (R): Send an email about the project team and title
- Oct. 28 (W): Project presentation (~10 slides)
- Oct. 29 (R): Mid-report (3-5 pages with 11-point font)
- Dec. 14 (M) & 16 (W): Presentation (~15 slides)
- Dec. 17 (R): Final report (4-8 pages with 11-point font)
  - It is highly suggested to include details of the experiment as an Appendix at the end of your report.
- Please refer to the following webpage for project subjects(http://academic.csuohio.edu/yuc/mobile09/)

Project Suggestions

- Communication is now translated to computation in GNU radio as most of communication functions are implemented in software. We are interested in measuring the computational complexity of communication functions with the purpose of energy efficiency, reliability-performance tradeoff, and so on. (Modulation Scaling for Energy Aware Communications Systems, ISLPED'01)

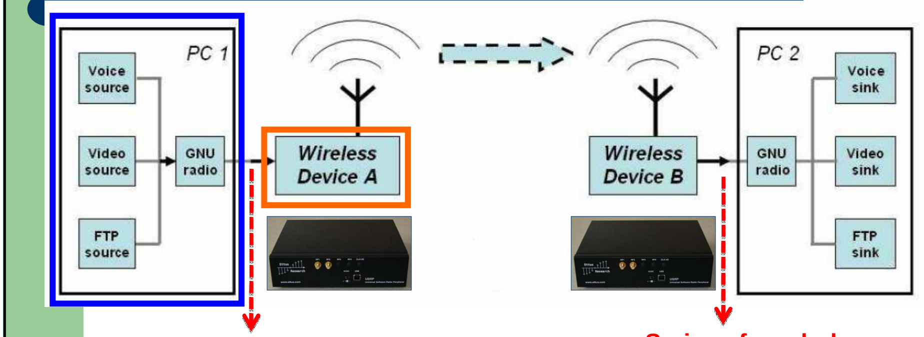
- GNU Radio and USRP can be used to implement and test cooperative communication technologies such as amplify-and-forward, worm-hole switching, virtual MIMO, etc. (Cooperative communication in wireless networks, IEEE Comm. Mag., 2004, Challenges: a radically new architecture for next generation mobile ad hoc networks, Mobicom 2005)

- Cognitive radio network where each radio monitors the channel utilization and intelligently switches to a underutilized channel. Synchronization with a receiver is a challenge. The design and implementation of a small testbed for a proof-of-concept is necessary.

Other Project Suggestion

- Ns-2 versus ns-3
  - http://www.nsnam.org/
  - http://www.nsnam.org/proposal.html
  - Become an early adopter.
  - Compare ns-2 and ns-3 simulation results.
- GNU Radio Experiment through Emulab
  - http://users.emulab.net/trac/emulab/wiki/Tutorial
  - http://www.cs.cmu.edu/~emulator/doc/softwareDoc/CMUlab.html
- USRP with SBC (Single Board Computer)
  - http://www.nd.edu/~jnl/pubs/commag2008.pdf
  - SBC LS-572 will be available

Architecture – Other Possibilities

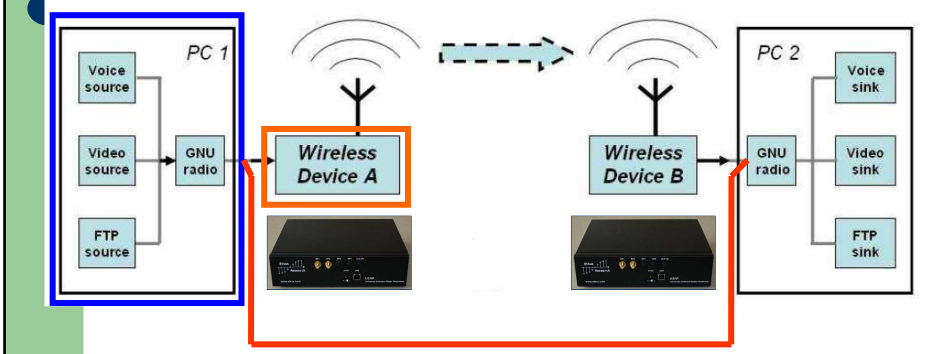


Series of symbols

Series of symbols

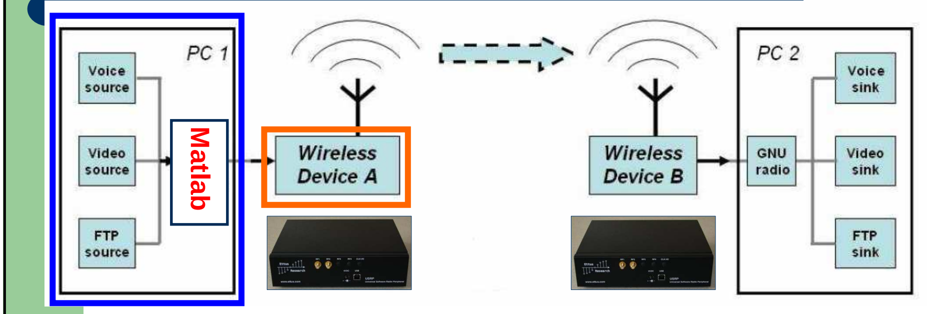
(Can we save and retrieve/reply them?)

Architecture – Other Possibilities



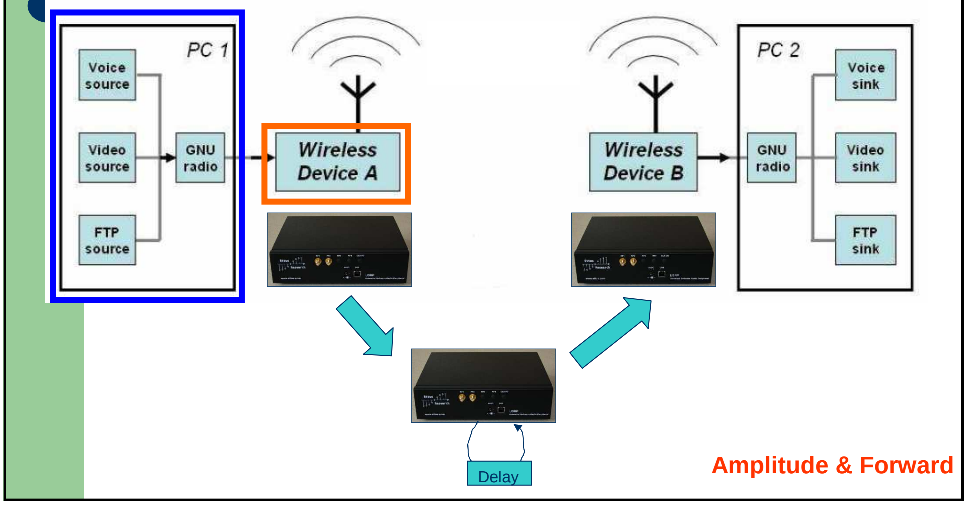
No need to use USRPs

Architecture – Other Possibilities



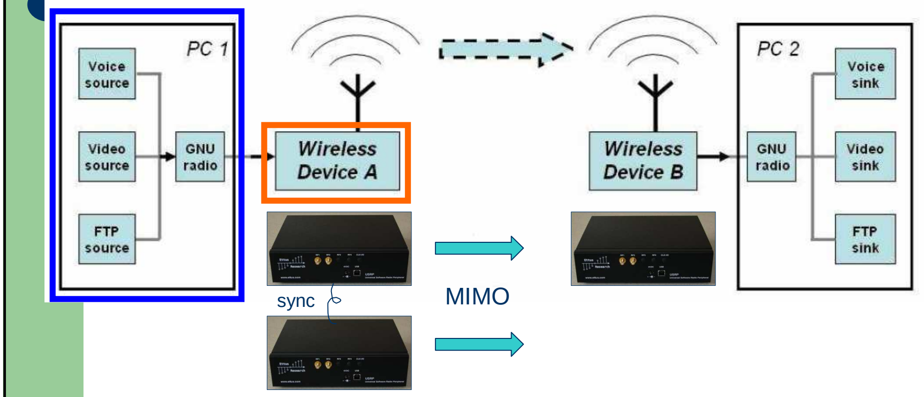
Doesn’t have to be GNU Radio

Architecture – Other Possibilities

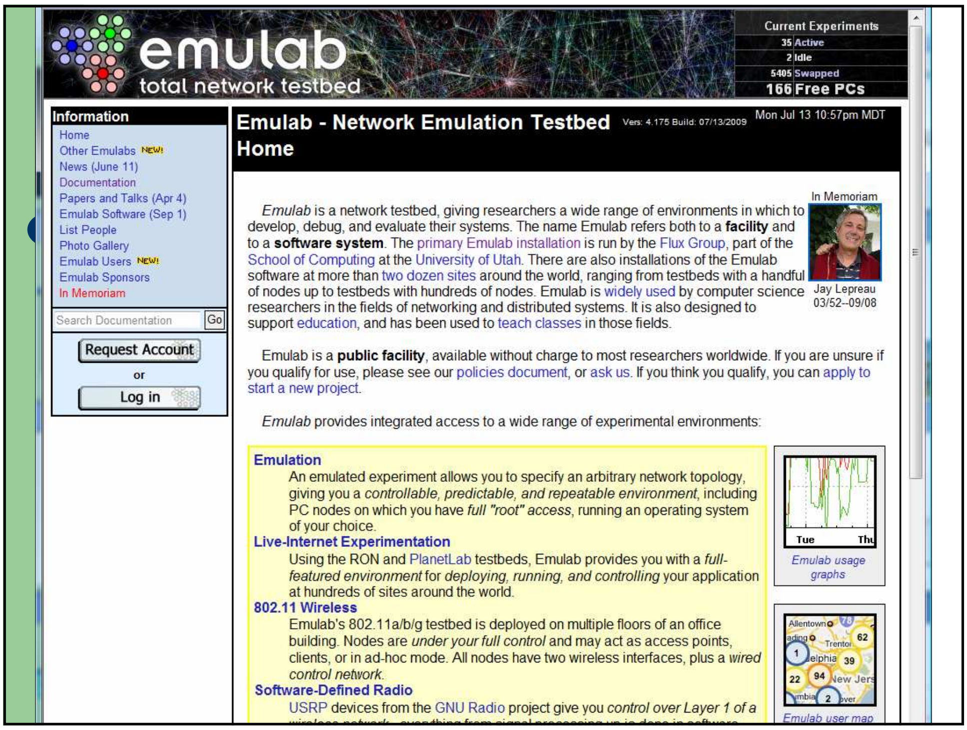


Amplitude & Forward

Architecture – Other Possibilities



(Virtual) MIMO



Current Experiments

35	Active
2	Idle
5405	Swapped
166	Free PCs

Emulab - Network Emulation Testbed

Vers: 4.175 Build: 07/13/2009 Mon Jul 13 10:57pm MDT

Home

In Memoriam

Jay Lepreau
03/52--09/08

Emulab is a network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. The name Emulab refers both to a **facility** and to a **software system**. The primary Emulab installation is run by the Flux Group, part of the School of Computing at the University of Utah. There are also installations of the Emulab software at more than two dozen sites around the world, ranging from testbeds with a handful of nodes up to testbeds with hundreds of nodes. Emulab is widely used by computer science researchers in the fields of networking and distributed systems. It is also designed to support education, and has been used to teach classes in those fields.

Emulab is a **public facility**, available without charge to most researchers worldwide. If you are unsure if you qualify for use, please see our policies document, or ask us. If you think you qualify, you can apply to start a new project.

Emulab provides integrated access to a wide range of experimental environments:

Emulation

An emulated experiment allows you to specify an arbitrary network topology, giving you a *controllable, predictable, and repeatable environment*, including PC nodes on which you have *full "root" access*, running an operating system of your choice.

Live-Internet Experimentation

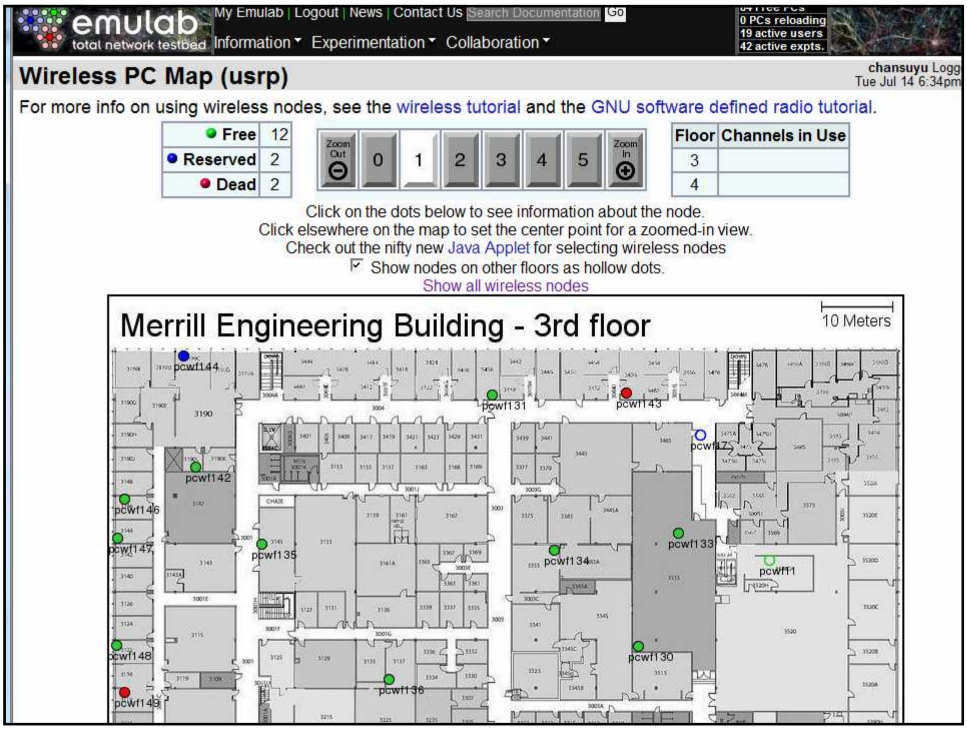
Using the RON and PlanetLab testbeds, Emulab provides you with a *full-featured environment* for *deploying, running, and controlling* your application at hundreds of sites around the world.

802.11 Wireless

Emulab's 802.11a/b/g testbed is deployed on multiple floors of an office building. Nodes are *under your full control* and may act as access points, clients, or in ad-hoc mode. All nodes have two wireless interfaces, plus a *wired control network*.

Software-Defined Radio

USRP devices from the GNU Radio project give you *control over Layer 1 of a*



Wireless PC Map (usrp)

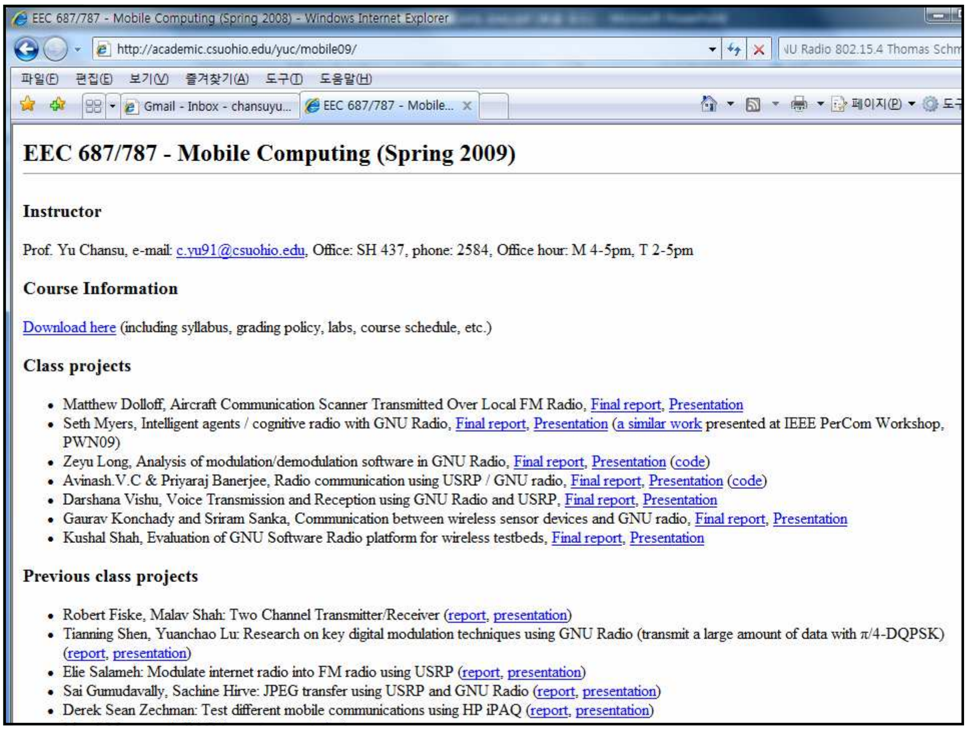
For more info on using wireless nodes, see the wireless tutorial and the GNU software defined radio tutorial.

● Free	12
● Reserved	2
● Dead	2

Floor	Channels in Use
3	
4	

Click on the dots below to see information about the node.
Click elsewhere on the map to set the center point for a zoomed-in view.
Check out the nifty new Java Applet for selecting wireless nodes
☑ Show nodes on other floors as hollow dots.
Show all wireless nodes

Merrill Engineering Building - 3rd floor


EEC 687/787 - Mobile Computing (Spring 2008) - Windows Internet Explorer
http://academic.csuohio.edu/yuc/mobile09/
NU Radio 802.15.4 Thomas Schm
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
Gmail - Inbox - chansuyu...
EEC 687/787 - Mobile...
페이지(P)
도구

EEC 687/787 - Mobile Computing (Spring 2009)

Instructor

Prof. Yu Chansu, e-mail: c.yu91@csuohio.edu, Office: SH 437, phone: 2584, Office hour: M 4-5pm, T 2-5pm

Course Information

Download here (including syllabus, grading policy, labs, course schedule, etc.)

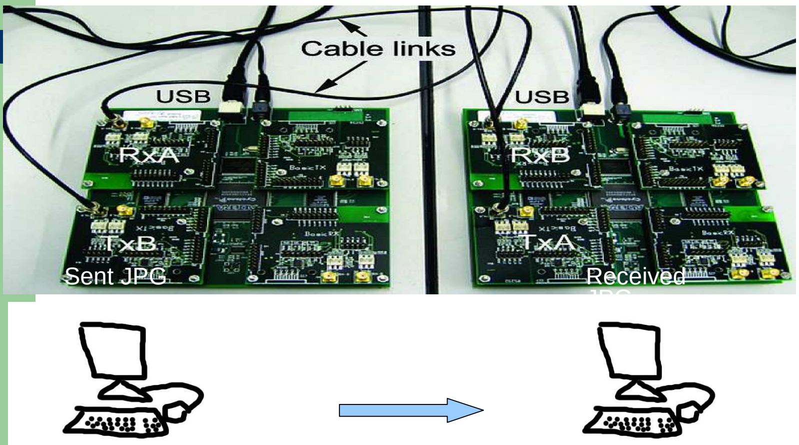
Class projects

- Matthew Dolloff, Aircraft Communication Scanner Transmitted Over Local FM Radio, Final report, Presentation
- Seth Myers, Intelligent agents / cognitive radio with GNU Radio, Final report, Presentation (a similar work presented at IEEE PerCom Workshop, PWN09)
- Zeyu Long, Analysis of modulation/demodulation software in GNU Radio, Final report, Presentation (code)
- Avinash.V.C & Priyaraj Banerjee, Radio communication using USRP / GNU radio, Final report, Presentation (code)
- Darshana Vishu, Voice Transmission and Reception using GNU Radio and USRP, Final report, Presentation
- Gaurav Konchady and Sriram Sanka, Communication between wireless sensor devices and GNU radio, Final report, Presentation
- Kushal Shah, Evaluation of GNU Software Radio platform for wireless testbeds, Final report, Presentation

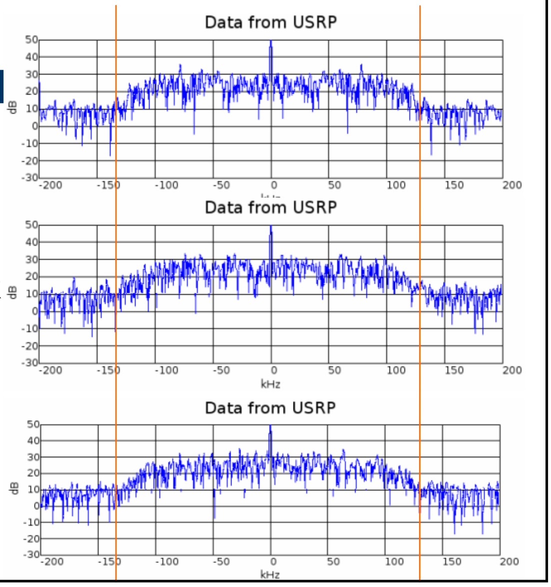
Previous class projects

- Robert Fiske, Malav Shah: Two Channel Transmitter/Receiver (report, presentation)
- Tianning Shen, Yuanchao Lu: Research on key digital modulation techniques using GNU Radio (transmit a large amount of data with π/4-DQPSK) (report, presentation)
- Elie Salameh: Modulate internet radio into FM radio using USRP (report, presentation)
- Sai Gumudavally, Sachine Hirve: JPEG transfer using USRP and GNU Radio (report, presentation)
- Derek Sean Zechman: Test different mobile communications using HP iPAQ (report, presentation)

File Transfer using Loop Back cable



Implement π/4-DQPSK



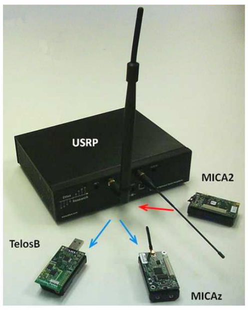
DBPSK

DQPSK

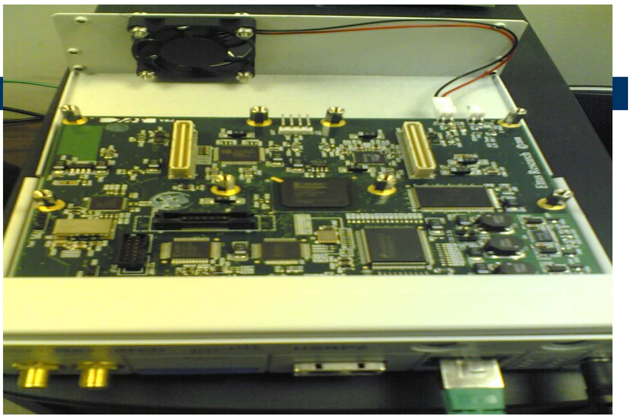
π /4
DBPSK

GNU Radio as Base-station

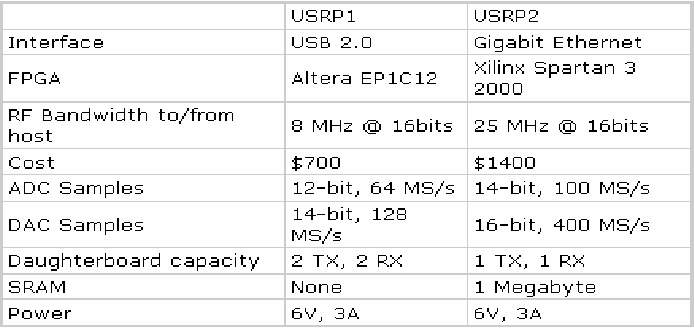
- Interoperability with heterogeneous motes
- Wideband nature of USRP enables multichannel communication
- Flexibility in selection of modulation scheme



USRP2



USRP2 Motherboard

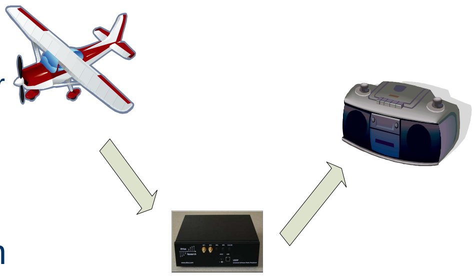


	USRP1	USRP2
Interface	USB 2.0	Gigabit Ethernet
FPGA	Altera EP1C12	Xilinx Spartan 3 2000
RF Bandwidth to/from host	8 MHz @ 16bits	25 MHz @ 16bits
Cost	$700	$1400
ADC Samples	12-bit, 64 MS/s	14-bit, 100 MS/s
DAC Samples	14-bit, 128 MS/s	16-bit, 400 MS/s
Daughterboard capacity	2 TX, 2 RX	1 TX, 1 RX
SRAM	None	1 Megabyte
Power	6V, 3A	6V, 3A

Table 1 – Comparison between USRP1 and USRP2 (www.gnuradio.org/trac/wiki/USRP2GenFAQ)

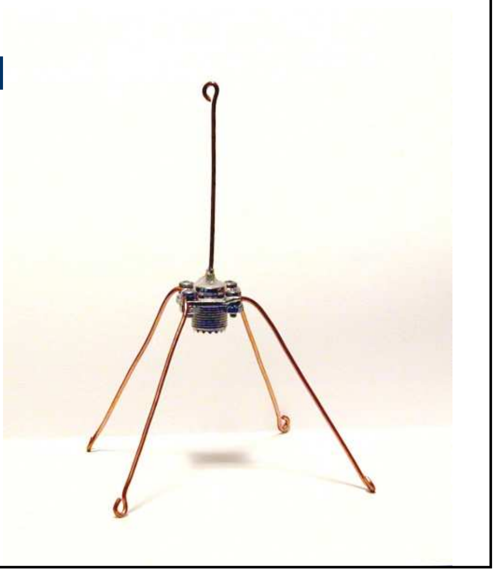
Air Band Scanner with Retransmission to Local FM Radio Using a Software Defined Radio

- Used for Aircraft Communications
  - Multiple frequencies for one airport (CLE uses at least 14)
- 108 MHz – 137 MHz
- Amplitude Modulation (AM) not Frequency Modulation (FM)



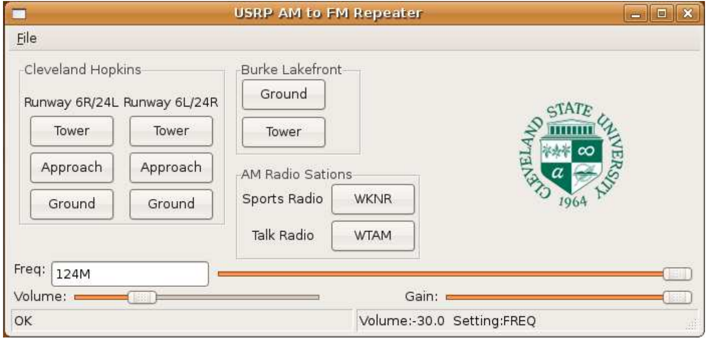
Quarter Wave Ground Plane Antenna

- Vertical – 23”, 10 AWG Copper House Wire
- Legs – 24”, 10 AWG Copper Wire.
  - Bent at 45º from the base
- Base – SO239 Bulkhead Connector
- Tuned to 120 MHz
- Built for portability



GUI Development

- wxPython Graphical User Interface (GUI)
- Developed within GNU Radio framework



Freq: 124M

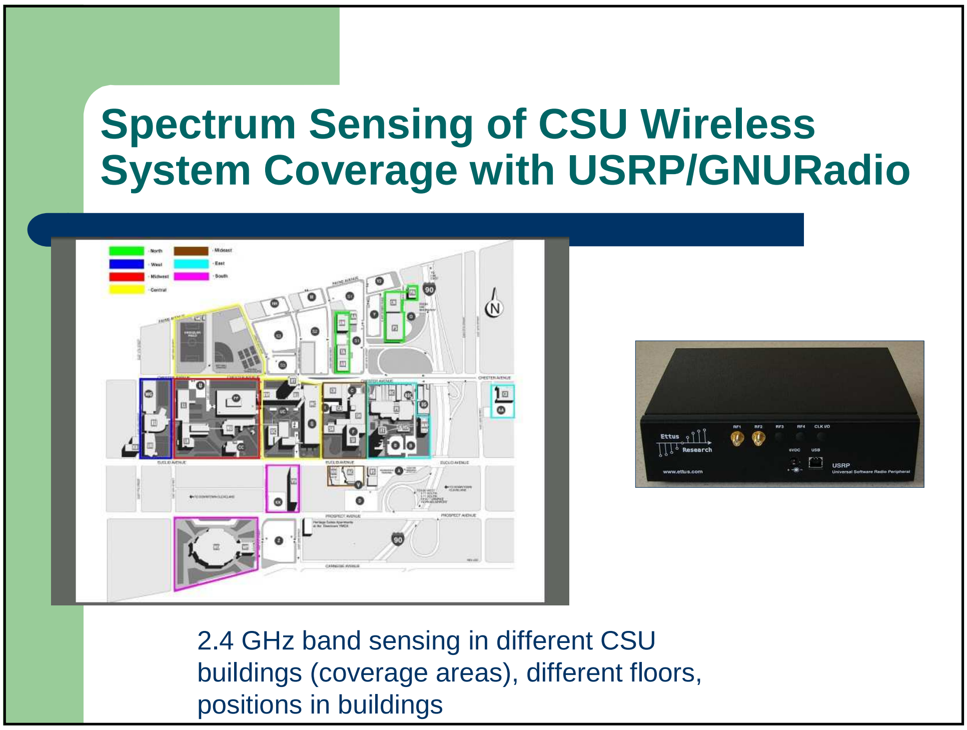
Volume:

OK

Volume:-30.0 Setting:FREQ

Software Tools Used for Development

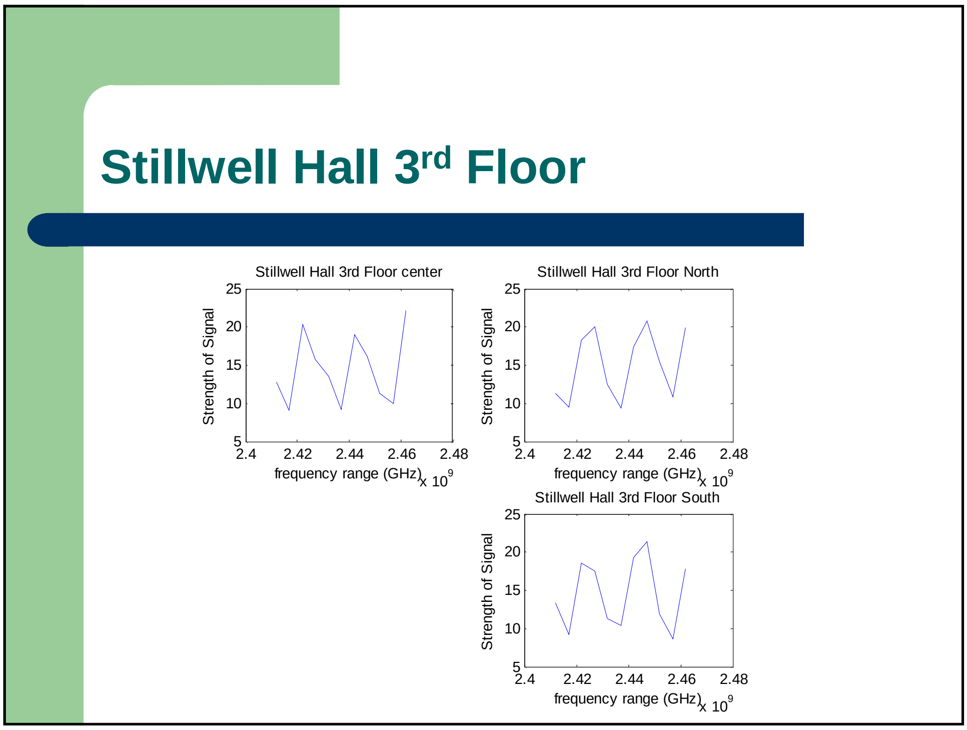
Software Package	Version	Use
Eclipse	3.4.2	Python code development
PyDev	1.4.5	Plug-in for Eclipse to enable Python developed in and IDE
Python	2.5.4	Scripting language used to access GNU Radio Libraries
GNU Radio	3.1.3	Libraries used to interface with the USRP and do signal processing
Sox	14.2.0	Used in getting MP3 to FM player working
Ubuntu Linux	8.04	Main OS used for development



Spectrum Sensing of CSU Wireless System Coverage with USRP/GNURadio

North
West
Midwest
Central
Mideast
East
South
N
Ettus Research
www.ettus.com
USRP
Universal Software Radio Peripheral

2.4 GHz band sensing in different CSU buildings (coverage areas), different floors, positions in buildings



Stillwell Hall 3rd Floor