ON THE CHALLENGES OF BUILDING A MULTI-ANTENNA SOFTWARE DEFINED PACKET RADIO

Ketan Mandke, Robert C. Daniels, Robert W. Heath, Jr., and Scott M. Nettles
Wireless Networking & Communications Group (WNCG)
Dept. of Electrical & Computer Engineering, The University of Texas at Austin
1 University Station C0803, Austin, TX 78712-0240
Email: {mandke, rdaniels, rheath, nettles}@ece.utexas.edu

ABSTRACT

The development of software-defined radios presents a set of challenges that is uniquely different from those faced in the development of traditional hardware radios. This paper examines problems and design challenges encountered in the development of Hydra, a multi-antenna multihop wireless testbed. In particular, we address the impact of design choices in the software architecture of this prototype, software performance issues, and practical issues dealing with the radio frequency (RF) front-end.

1. INTRODUCTION

The design space for research and commercial wireless devices ranges from radios implemented entirely in hardware to completely software-defined radios (SDR). In the former, the physical (PHY), medium access control (MAC), and sometimes even Network layer are all implemented in hardware, namely in silicon or on field programmable gate-arrays (FPGAs). Hydra, an experimental wireless testbed that features a completely software-defined protocol architecture [1], is at the opposite end of this spectrum. The continuum of design choices in between presents various tradeoffs between performance, flexibility, reconfigurability, development time, and cost.

We present a set of challenges faced in designing and experimenting with Hydra. The goal of this prototyping effort is to develop a testbed that provides a flexible (i.e. easy to modify) platform for researchers to implement a wide variety of cross-layer wireless algorithms. In particular studying these algorithms in real propagation channels, with non-idealized radios, and realistic traffic will help to bridge the gap between theoretical results generated by researchers and practical systems designed by engineers. This, in part, motivated the pursuit of a software-defined design for Hydra. The goal of this paper is to provide a detailed discussion of the design choices made in the development of this prototype. In particular we examine tradeoffs and limitations in the system which may provide insight to researchers developing similar software-defined testbeds.

This material is based in part upon work supported by the National Science Foundation under grants EIA-0322957, CNS-0435307, and CNS-0626797, the Air Force Research Lab under grant numbers FA8750-06-1-0091, and FA8750-05-1-0246, the Office of Naval Research under grant number N00014-05-1-0169, and the DARPA IT-MANET program, Grant W911NF-07-1-0028.

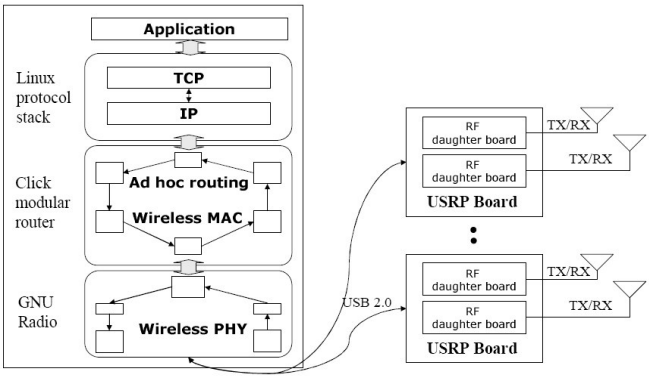

Fig. 1: Each Hydra node consists of a general purpose PC connected to the USRP over a USB 2.0 interface.

2. SYSTEM OVERVIEW

Hydra was designed to provide a flexible framework for rapidly prototyping cross layer protocols in multihop wireless networks. As shown in Figure 1, each Hydra node is composed of a general purpose PC and a multi-antenna RF front-end. The Network (routing), MAC, and PHY layers are implemented entirely in software running on the general purpose processor. This approach allows us to rapidly implement a wide range of wireless protocols in high level languages, which are suitable to most networking and communications researchers who may not have experience with hardware languages (e.g. Verilog, VHDL).

2.1. Open-source Software

Three open-source software packages are utilized to implement the MAC and PHY layers of Hydra, namely the

TABLE I: Hydra Implementation of the IEEE 802.11n PHY

System Bandwidth	$1\ MHz^*$
Center Frequency	$2.4 - 2.5\ GHz$
Maximum TX Power	$10\ mW$
Modulation	BPSK, QPSK, 16-QAM, 64-QAM
Coding	Bit-Interleaved Binary Convolution Coding
SISO Data Rates	0.65^*, 1.30^*, 1.95^*, 2.60^*, 3.90^*, 5.20^*, 5.85^*, 6.50^* $Mbps$
MIMO Data Rates	$2\times$, $3\times^+$, and $4\times^+$ SISO Data Rates
Diversity Schemes	Cyclic Delay Diversity, Space-time Coding, Spatial Mapping, Beamforming

* indicates non-standard values
$^+$ indicates capabilities in development

Click modular router [2], GNU Radio [3], and IT++ [4]. In Click and GNU Radio, packet and signal processing elements are built in C++ and then assembled using a glue language to compose a protocol. IT++, a C++ library of various digital communications and signal processing functions, provides many algorithms used in building the physical layer. The PHY and MAC of Hydra extend current wireless standards, namely: IEEE 802.11n [5] and the distributed coordination function (DCF) of IEEE 802.11. Table I provides an overview of IEEE 802.11n as implemented in Hydra.

2.2. RF Front-End

The physical layer of Hydra sends and receives baseband waveforms to and from an open-source RF front-end, the Universal Software Radio Peripheral (USRP version 1) [6]. The primary functions of this frequency-agile radio platform are filtering, digital-to-analog/analog-to-digital conversion, and upconversion/downconversion of baseband signals. A one million gate FPGA on the USRP implements most of this functionality. GNU Radio provides a convenient application programming interface (API) to control and communicate with the USRP, which connects to the host machine over a USB 2.0 connection.

Each USRP also features up to two RF daughterboards, which modulate signal from baseband to a carrier frequency and vice versa. Various daughterboards are available to allow operation in a variety of frequency bands, such as the Industrial, Scientific, and Medical (ISM) bands. The multi-antenna support of the USRP allows it to be used to implement multiple-input multiple-output (MIMO) systems.

2.3. Physical Layer Software-Architecture

Many of the design decisions discussed in this paper deal with the architecture of the Hydra physical layer. As mentioned, the physical layer is implemented using the GNU Radio framework. The software-architecture of this implementation consists of three main parts: the MAC/PHY interface, the PHY itself, and the radio interface. Each section is composed of an upstream and downstream part. As shown in Figure 2, there are multiple threads throughout

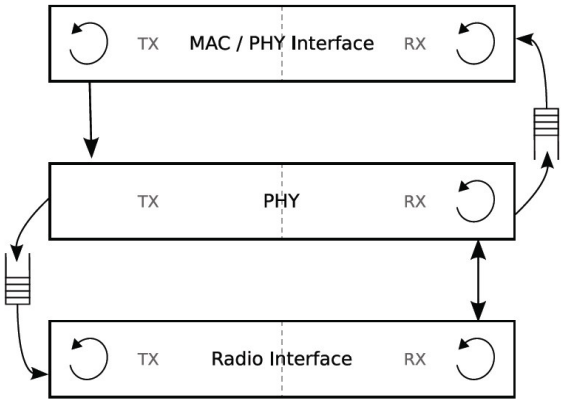

Fig. 2: Multi-threaded architecture of PHY layer in Hydra.

the architecture. These threads communicate with each other primarily through control or data queues.

Since the MAC layer is implemented in Click and runs as a separate process in its own address space, some type of interprocess communication (IPC) must be used to connect the MAC and PHY. Hydra utilizes a local socket connection to facilitate this interaction. The MAC/PHY interface in Figure 2 manages the PHY side of this interface and implements a flexible frame structure for tightly coupled cross-layer interaction between these layers.

2.3.1. Standard Operation

In the typical transmission of a packet, the MAC layer will deliver a packet along with cross-layer side information to the MAC/PHY interface over a local socket connection. This cross-layer side information usually consists of parameters needed to configure the physical layer (e.g. rate, transmission power) or physical layer information needed in PHY processing (e.g. channel information, MIMO precoding matrices). After configuring the physical layer according to this side information, the interface sends the MAC packet downstream to the physical layer for transmit processing. The PHY layer then generates a baseband waveform and places it into a data queue for the radio interface. The waveform generated by the PHY is a sequence of floating point values. The transmit side of the radio interface must reformat this data into the fixed point format (i.e. 16-bit integers) required by the USRP. After reformatting, the waveform is sent on to the RF front-end.

Much of the processing in receiving a packet is the dual of the transmit operation. The reception of a packet is divided between a capture thread, residing in the radio interface, and a receiver thread, which resides in the PHY. The capture thread continually captures blocks of 2048 baseband samples, converts the fixed point values into floating point samples, places them into a capture buffer, and then notifies the receiver thread that new samples are available. The receiver thread listens to the baseband samples in this buffer until it detects a packet, i.e. performing a packet detection algorithm such as energy

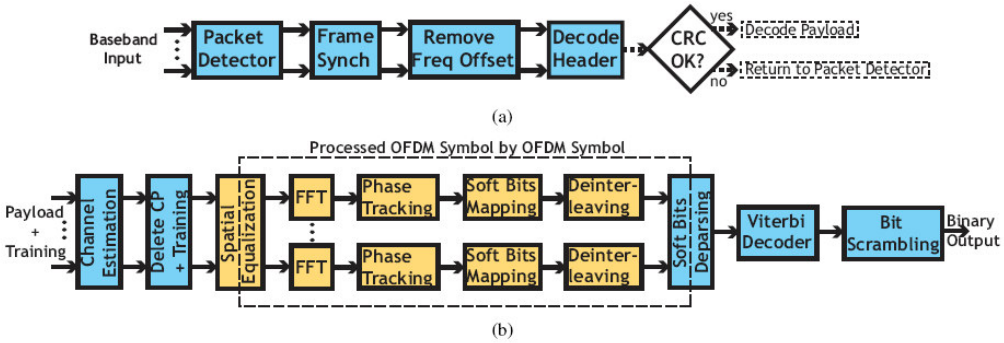

Fig. 3: (a) Packet detection and (b) decoding in MIMO-OFDM receiver of Hydra.

detection or the Schmidl & Cox method [7]. Once the entire detected waveform has been captured, the receiver thread attempts to decode the incoming packet. After successfully decoding a packet, the receiver thread sends it along with physical layer side information upstream to the MAC/PHY interface. The receiver thread also notifies the capture thread whenever it is finished processing a set of samples so that the capture thread can remove these samples from the buffer. Finally, the MAC packet and side information are delivered to the MAC layer.

Note, the capture and receiver threads in Hydra have a pseudo-producer-consumer relationship with regard to their shared buffer. That is, although the receiver thread reads samples out of the buffer, the capture thread manages all writing to and consuming from this buffer. This eliminates the need for any mechanisms to ensure mutually exclusive buffer access, as in a traditional producer-consumer. The concurrent operation of these threads requires a shared signaling semaphore (to indicate when new samples arrive) and an internal queue for control messages from the receiver thread to the capture thread (to indicate when samples are no longer needed and can be consumed). This internal semaphore and queue were omitted from the block diagram in Figure 2.

2.3.2. Multi-threaded Performance

In order to leverage a multi-threaded architecture such as this, it important to understand where there are opportunities to exploit concurrency or parallelism. This parallelism can be utilized by multi-core processors to significantly improve the performance of software-defined radios. For example, in the upstream or receive side of the physical layer, the capture operation can occur in parallel with physical layer detection and decoding. A general purpose processor with four cores could allow the MAC/routing layer, the receive/transmit processing, and the capture operation to be performed on separate cores while still having one core remaining for general usage.

Additional improvement in performance can be achieved by analyzing the processing burden of various parts of the physical layer. For example, the capture thread is memory intensive, but does not require much processing. The decoding algorithm however, is very processing intensive. To reduce the processing burden of a core running the physical layer receive processing, the detection algorithm could be offloaded to the core that is running the capture thread. Further extension of this idea could involve dissecting the physical layer decoding process shown in Figure 3 into pieces running on parallel cores (e.g. running channel estimation/equalization and Viterbi decoding on separate cores).

The operation of the downstream or transmit side of the physical layer is sequential and does not present much opportunity for parallelism. The reason for introducing threads in situations where there is no parallelism of which to take advantage is that it decouples the operation of logically separate parts of the architecture. This improves the overall modularity of the system at the cost of some additional context switching overhead.

3. CHALLENGES

The previous section described the capabilities and software-architecture of Hydra. In this section, we examine some of the engineering tradeoffs faced in the design of that system and some practical issues associated with the limitations of the RF front-end.

3.1. Capture Delay vs. Packet Detection Overhead

As discussed previously, in the capture operation the radio interface captures blocks of samples delivered by the USRP over the USB interface. The receiver then processes these incoming samples to detect if a packet has been received. Consider the capture of N samples, where B represents the number of samples in a block. The delay in capturing a single block of B samples can be characterized as

$$D_{capture} = \kappa(B) + B \cdot T_s + \theta_{USB} \quad , \qquad (1)$$

where T_s is the duration of a sample, θ_{USB} is a random delay associated with the USB latency, and $\kappa(B)$ is the overhead of memory access (i.e. buffer copy operations). Since we assume that $\kappa(B)$ is $O(B)$, the capture delay for a

block of B sample is also $O(B)$. The processing overhead incurred for packet detection can be characterized as

$$T_{pd} = N_b(\mu \cdot B + \theta_{cs}) = N\left(\mu + \frac{\theta_{cs}}{B}\right), \quad (2)$$

where $N_b = N/B$ is the number of blocks, μ is the processing overhead per sample for the packet detection algorithm, and θ_{cs} is a random delay due to context switching overhead. Equation 2 shows that T_{pd} is inversely proportional to block size, i.e. $O(B^{-1})$. This implies that as we reduce block size B to reduce capture delay, packet detection overhead will increase. If packet detection time B exceeds $N \cdot T_s$, the system will no longer be able to maintain real time operation. This imposes a bandwidth constraint on the system.

3.2. Block vs. Stream Processing

Figure 3 shows a block diagram of the MIMO-OFDM receiver in Hydra. As with most physical layer decoding, the operation of this receiver can be thought of as a processing pipeline of M stages acting on an endless stream of baseband samples. The processing requirements of these stages are not uniform. For example, a linear channel equalizer consumes fewer resources than a Viterbi decoder. We refer to this pipeline approach as stream processing. Hardware implementation (i.e. in silicon or FPGAs) of this processing model would naturally have a high degree of concurrency. At each time slice in the operation of such an implementation, M processing stages would be working on M different slices of the input stream. Similar parallelism could also be achieved in a software-defined radio, but would come at the cost of increased context switching overhead.

Another approach to this processing problem would be to serially perform each of the M stages of the pipeline on a large block of input slices (i.e. only one stage is active at any given time). We refer to this approach as block processing. A software implementation on a general purpose processor of the receiver structure of Figure 3 would naturally operate in such a fashion. Hydra uses the block (or sequential) processing model. To compare the performance of parallel and sequential processing architectures in a software-defined radio, we consider a PHY decoding algorithm with M processing stages. The time to process N input slices in a pipeline or stream processing SDR implementation can be approximated as

$$T_{parallel} \approx N\left(\tau_{\max} + \frac{M \cdot \bar{\theta}}{p}\right), \quad (3)$$

where we assume $N >> M$, $\tau_{\max}$ is the processing time for the slowest stage of the processing pipeline, $\bar{\theta}$ is the average cost of a context switch, and p is the number processor cores in the general purpose processor.

In a sequential or block processing model, where only one stage is active at any given time, the time to process N samples is approximately

$$T_{serial} \approx M \cdot (N\beta_p \tau_{avg} + \bar{\theta}), \quad (4)$$

where τ_{avg} is the average time of a processing stage and β_p is an efficiency factor that accounts for any speed up in processing that occurs when p processor cores work together (i.e. $p^{-1} < \beta_p \le 1$).

Assuming that $\tau_{\max} \ge \tau_{avg} \ge \bar{\theta}(p\beta_p)$, then block processing will be more efficient than stream processing when $T_{serial} < T_{parallel}$, or equivalently when

$$M < \frac{N\tau_{\max}}{N \cdot \left(\beta_p \tau_{avg} - \theta/p\right) + \bar{\theta}} \quad (5)$$

$$< \frac{\tau_{\max}}{\beta_p \tau_{avg} - \bar{\theta}/p}. \quad (6)$$

When N is sufficiently large, the upper bounds of Equations (5) and (6) are approximately equal. This implies that, as a general rule of thumb, whenever there exists a processing stage that is significantly slower than the average stage, i.e. the ratio $\tau_{\max}/\tau_{avg}$ is large, then block processing will be more efficient than stream processing.

Note, stream processing also benefits from the ability to potentially overlap capture and receive processing, although this was not considered in this analysis. It should also be noted that we are primarily concerned with the design of software-defined packet radios, which will only be decoding waveforms of finite sizes. This fact might further motivate the use of the simpler block processing model.

3.3. Practical Limitations of the Radio

In the development of Hydra many of the following limitations of the RF front-end impacted system performance. These features put practical constraints on the achievable performance of Hydra and how the testbed can be utilized in wireless experiments.

3.3.1. Digital Transmit Interpolation Filter

As mentioned previously, one of the tasks of the RF front-end is to interpolate (or upsample) the baseband waveform created by the physical layer. Part of this interpolation process is the application of an anti-imaging filter [8]. Digital interpolation of a signal produces frequency modulated copies or images of the original signal. An ideal digital interpolation filter would reject all of these images while allowing the original signal to pass without any distortion. In general, the design of an ideal filter requires a large amount of memory. Because of space limitations on its

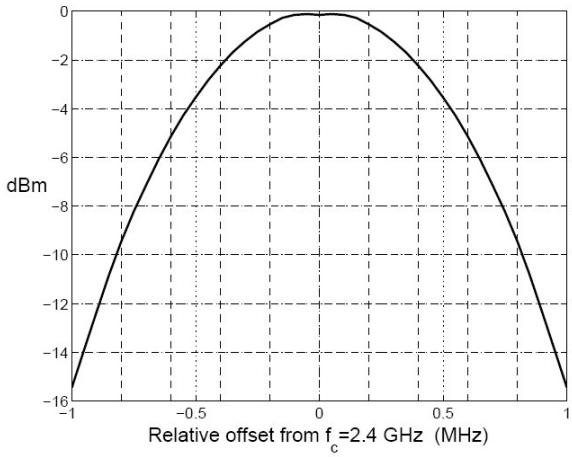

Fig. 4: Frequency response of transmit interpolation filter on USRP measured at 2.4 GHz. Interpolation factor is 64.

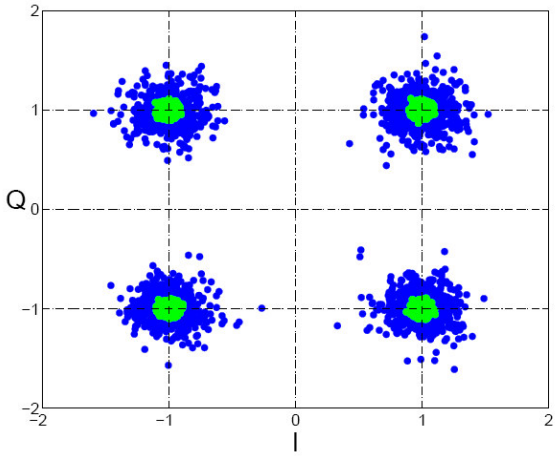

Fig. 5: Superimposed constellations of equalized signal in AWGN (SNR= 30 dB) with and without transmit filter.

FPGA, the USRP's digital transmit filter has a non-ideal frequency response. As shown in Figure 4, the frequency response of the transmitted signal after filtering drops as much as 16 dB. This significant loss in signal strength can adversely impact the performance of many communication systems.

In Hydra, the physical layer uses orthogonal frequency division multiplexing (OFDM) modulation. Please refer to [9] for a detailed discussion of OFDM and the challenges in multicarrier systems. This modulation scheme divides the frequency spectrum into a set of parallel channels and then loads complex symbols onto each subchannel. In the context of OFDM, the digital transmit filter of the USRP causes the amplitude of the attenuated outer subchannels to be up to six times smaller than that of the inner subchannels. These lower fidelity subchannels bring down the average quality of the signal.

To illustrate this, consider an OFDM system in which the subchannels are loaded with quadrature amplitude modulated (QAM) signals. Figure 5 shows the superimposed QAM constellations of two signals that have passed through an additive white Gaussian noise (AWGN) channel with noise power $\sigma^2 = -30\,\text{dB}$. One of the constellations represents the equalized signal constellation of an OFDM waveform transmitted with the transmit interpolation filter in Figure 4, while the other represents one that was transmitted with an ideal filter. The inner shaded area at each constellation point depicts the normal region into which noise perturbed signals might fall, i.e. with an ideal filter. The outer area at each constellation point indicates the expanded area into which these points may now fall when this transmit filter is used. This example demonstrates how the average signal quality is degraded because of the frequency response of the digital transmit filter.

One method to overcome the frequency selectivity of this filter is to apply a digital filter to compensate for the shape of the output frequency response. This method is not advisable as it may lead to overflow in the representation of the outgoing signal, a phenomenon sometimes referred to as saturation or clipping. Another method, which is used in Hydra, is to interpolate the signal by a factor of two with a high quality filter prior to sending it to the RF front-end. This causes the signal to be isolated to only a fraction of the bandwidth of the transmit filter, which reduces the power loss of the outer subchannels to only $-4\,\text{dB}$.

3.3.2. Non-ideal Hardware

As discussed previously, the USRP provides the ability to modify the carrier frequency of the radio over a large frequency band (generally over a few hundred megahertz). This frequency agility does not, however, imply that all carrier frequencies behave the same. In fact, the frequency response of the analog RF front-end of the USRP is nonuniform, as shown in Figure 6. Similarly, antennas designed to operate in a particular frequency band may not provide uniform performance over all frequencies in the band. The S_{11} characteristics of an antenna provide a measure of the reflective properties of an antenna as a function of frequency. Consider the S_{11} measurements for an off the shelf antenna sold for operation in the 2.4 GHz ISM band, shown in Figure 7. This plot shows that a good operating range for this antenna is 2.41–2.43 GHz, i.e. where the power lost through the antenna is less than 1%. These nonuniformities cause certain frequencies to be better suited for operation. It is therefore important to profile the frequency response of both the RF front-end and antennas used in any wireless device. Based on the measurements in Figures 6 and 7, an ideal operating range for Hydra is between 2.41–2.43 GHz.

3.3.3. USB Interface

As previously mentioned, the USRP connects to the host machine over a USB 2.0 connection. This USB interface has a theoretical bandwidth of 480 Mbps. Since each complex sample is composed of 4 bytes, this means that the USB interface can support approximately 8 Msamples per second.

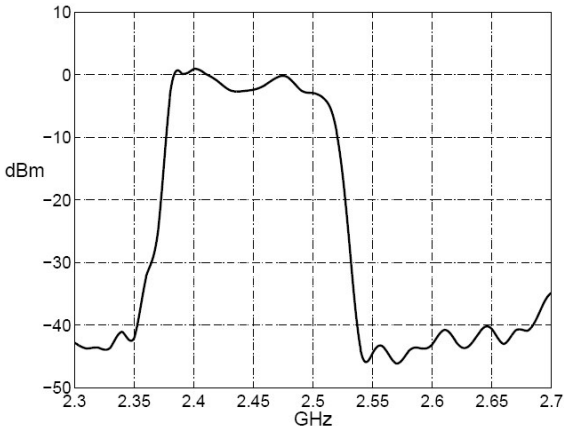

Fig. 6: Maximum transmit power of Flex 2400 daughterboard within linear range of USRP. Least attenuation occurs at 2.4 GHz.

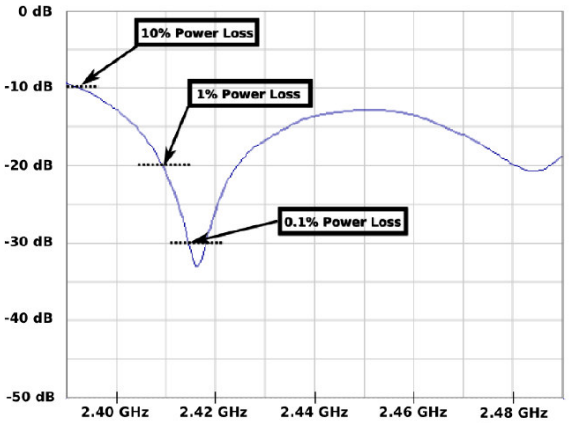

Fig. 7: S11 measurements for a 2.4 GHz antenna.

Since this is divided among two antennas, in theory the USRP could support a 2 antenna MIMO stream operating with up to 4 MHz of bandwidth. Although, in practice the USB operates at approximately half of its theoretical limit, thus the USRP can reliably provide MIMO samples at a rate of 2 Msamples/sec. Higher bandwidth connections are being explored for future versions of the hardware.

Because of the bandwidth limitations introduced by the USB interface and from packet processing overhead, the throughput of the MAC level in Hydra is at least an order of magnitude slower than that of normal IEEE 802.11. Although this software-defined approach in Hydra incurs the cost of reduced performance, it has the benefits of a greater degree of flexibility, reconfigurability, and monitoring/probing functionality which other higher performance platforms might lack. These are some of the major benefits to researchers employing software-defined radios for wireless experimentation.

4. CONCLUSION

This paper was motivated by the challenges and problems faced in the development of Hydra. It described the physical layer architecture of Hydra, which motivated a discussion of many general issues pertaining to the architecture and design of software-defined radios. Finally, there was a discussion of some of the limitations of the RF front-end and their impact on system performance. We conclude with a design feature of Hydra which motivates some concerns to be addressed in future work.

The capture and process approach that Hydra employs in processing received signals provides some architectural benefits, but it also introduces some unique timing issues in the MAC layer. One of these issues pertains to the definition of carrier sense. Because of the delay introduced in buffering and analyzing captured data, anytime a carrier sense decision is made it may quickly become out of date. For example, a typical transmitted packet in Hydra can be around 2500 samples. When using a block size of 2048 samples, this means that by the time the entire packet has been captured, any carrier sense decision made on the captured data may already be out of date. This is an example which illustrates some of the timing problems that are introduced in the MAC layer when using a completely software-defined protocol architecture. Such problems motivate the need to rethink definitions of carrier sense and other notions of channel state at the MAC level. We mention this as a final thought that might motivate future work on the architecture of softwaredefined radios.

REFERENCES

[1] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. C. Daniels, W. Kim, S. M. Nettles, and R. W. H. Jr., "Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed," in *Proceedings of the 65th IEEE Vehicular Technology Conference*, Apr. 2007, pp. 1896–1900.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[3] "GNU software radio." [Online]. Available: http://gnuradio.org/trac

[4] "IT++." [Online]. Available: http://itpp.sourceforge.net/

[5] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Draft 2.0: Enhancements for Higher Throughput*, Part 11 standard ed., IEEE 802.11n Working Group, February 2007.

[6] "GNU radio: universal software radio peripheral radio." [Online]. Available: http://gnuradio.org/trac/wiki/USRP

[7] F. Wu and M. A. Abu-Rgheff, "Time and frequency synchronization techniques for ofdm systems operating in gaussian and fading channels: A tutorial," in *The 8th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNET)*, June 2007.

[8] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing (2nd Edition)*. Prentice Hall, February 1999.

[9] A. Goldsmith, *Wireless Communications.* New York, NY, USA: Cambridge University Press, 2005.