A Case Study of A MIMO SDR Implementation

Xiaolong Li Weihong Hu Homayoun Yousefi'zadeh
Department of EECS
University of California, Irvine
[xiaolonl,weihongh,hyousefi]@uci.edu

Akber Qureshi
Network and Information Systems
The Boeing Company
akber.qureshi@boeing.com

***Abstract*— Funded by a research contract from the Boeing Company, we report on the implementation of a Multiple-Input Multiple-Output (MIMO) Software Defined Radio (SDR) platform forming an operational Mobile Ad-Hoc Networks (MANET) node. Our SDR platform utilizes Universal Software Radio Peripheral (USRP) including a motherboard for baseband processing, two daughter boards for RF frontend processing, and an embedded Intel Core General Purpose Processing (GPP) unit hosting Linux operating system. We use GNU Radio to program the PHY and DATA LINK layers of USRP. We also implement the TCP/IP stack on the SDR using the TAP/TUN driver of the Linux Kernel. We are able to run a variety of applications including file transfer, image delivery, stored audio delivery, and live speech. We report the results of our performance benchmarking related to these applications.**

***Index Terms*—Software Defined Radio, MIMO, USRP, GNU Radio, Click Router Module, PHY, MAC, FTP, HTTP, AUDIO.**

I. Introduction

Software defined radios are becoming more prevalent in the formation of wireless networks. Simply put, the use of an SDR can allow for moving the code as close as possible to a target hardware platform thereby treating various hardware problems as software problems. This characteristic of SDRs offers great flexibility for MANET research and development because a variety of newly-developed protocols can be implemented and verified in a testbed formed by real radio platforms. We are primarily interested in SDRs as a vehicle of evaluating the performance of a variety of cross layer MANET protocols spanning over different layers of protocol stack from the PHYSICAL layer to the APPLICATION layer. Specifically, in the context of a project funded by the Boeing Company, we are designing and implementing a pair of cross layer Medium Access Control (MAC) and routing protocols for MANETs that are capable of exchanging information in order to improve the routing performance. Among the necessary features of our cross layer design, we are adding anycasting features to the MAC layer such that link quality information collected by the MAC protocol at the data link layer can be exchanged with the routing protocol at the NETWORK layer.

This work was sponsored by a research contract from the Boeing Company.

This paper reports the results of our initial experiments with an SDR platform integrated in the context of our project and utilized to evaluate the performance of MANET protocols. Our SDR platform utilizes Universal Software Radio Platform (USRP) which contains the hardware implementation of the baseband processing unit as well as the RF frontend, GNU Radio an open-source software toolkit for programming software radios, and Click Router module an open-source software package for developing sophisticated MAC and routing protocols. The SDR interfaces with an embedded GPP unit through a Universal Serial Bus (USB) connection. The GPP runs Linux operating system and hosts GNU Radio as well as Click router modules in addition to hosting the TCP/IP stack.

In our experiments, we investigate and profile the characteristics of the above mentioned SDR platform including the achievable sustained throughput, transmission delay, and packet error rate in a number of operating scenarios. The applications of interest include data content delivery relying on File Transfer Protocol (FTP), web content delivery using Hyper Text Transfer Protocol (HTTP), and audio content delivery both stored and live. The rest of this paper is organized as follows. Section II introduces the USRP and GNU Radio framework. In Section III, we describe an overview of our SDR system. In Section IV, we report our experimental results and provide an analysis of our results. We present the related work in Section V. Finally, we conclude the paper in section VI.

II. The SDR Platform

In this section, we introduce USRP [1] the key hardware component for building our SDR platform. First, the USRP board is described. Then, GNU Radio [2] the software package necessary for programming USRP is discussed.

A. USRP

USRP is an open interface hardware platform for developing SDRs. Typically, a USRP system consists of a motherboard and up to four plug-on daughter boards that function as

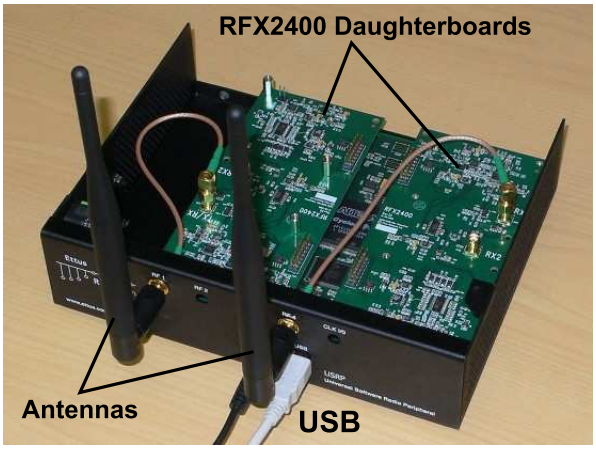

Fig. 1. The layout of the USRP-based SDR.

the RF frontends. In addition to four A/D and D/A converters, the motherboard contains a million gate Altera Cylone FPGA that performs the most intensive data processing such as data queuing and decimation/interpolation filtering. Moreover, as USRP is connected to a host PC using a USB2 interface, the FPGA is responsible for reducing the data rate to a suitable level for the USB connection. Fig. 1 shows the layout of the USRP-based SDR. Specifically, the data streamed over the USB2 interface is in the form of I/Q samples. Each sample consists of a 16-bit in-phase component and a 16-bit quadrature component. Given that the usual transfer rate of a USB2 interface is 32MBps[1] or 8 mega samples per second, a communication channel with a bandwidth of 4Mbps can be accommodated according to the Nyquist's Sampling Theorem [3]. In practice, the transfer speed of USB2 interface restricts the capacity of the radio channel. Fortunately, the next generation USRP [4] that will be available shortly can boost the speed to 1Gbps.

Notably, a Programmable Gain Amplifier (PGA) is employed on both receiving and transmitting path to amplify the input/output signal. The PGA can provide up to a 20dB gain and further allows for simulating various signal fading effects by regulating the gain of the radio transceiver. Furthermore, a USRP motherboard is equipped with four slots in which up to two receive (RX) daughter boards and two transmit (TX) daughter boards can fit. The fully synchronous transceiver design of USRP allows for multiple daughter board cards to be synchronized thereby creating a MIMO-capable system. In our experiments, we use two RFX2400 daughter board cards.

[1]The terms bps and Bps are used to indicate bits per second and Bytes per second, respectively.

B. GNU Radio

GNU Radio is an open-source software package that can run on various hardware platforms. Coupled with USRP, GNU Radio provides an ideal software platform for developing wireless protocols at the PHYSICAL and DATA LINK layers of the protocol stack. GNU Radio applications are primarily written using Python, while performance-critical signal processing blocks are implemented in C++.

GNU Radio is built on two important notions, "blocks" and "flowgraphs". Blocks are basic operation units that process continuous data streams. Each block has a number of input and output ports. GNU Radio has a number of signal processing blocks, i.e., modulation, demodulation, filtering, and communicating that can be hosted on USRP. A flowgraph is constructed by connecting blocks together. In a flowgraph, each vertex associates with a signal processing block and the edges represent the data stream between blocks. A radio can be built by creating a flowgraph. Furthermore, creating a new block is not difficult, which provides users with great flexibility. Generally, it is easy to build a customized radio platform by constructing a flowgraph. To execute, GNU Radio provides a single thread scheduler to sequentially traverse all of the blocks in a flowgraph. It is worth noting that the standard implementation of GNU Radio is stream oriented and does not support packet processing. An extension of GNU Radio called "m-blocks" [5] implemented a new primitive into GNU Radio by using message-blocks allowing for easy processing of packet-oriented data. However, due to the fact that there is no easy interconnection between m-blocks and flowgraph available [6], we choose not to use m-blocks in our experiments.

While the computational requirements depend on the type of running applications, most GPP units can fulfill the hardware requirements of running GNU Radio. Based on our experiments, a system with a 1 GHz processor and at least 256MB of memory satisfies the minimum requirement for running GNU radio. In our experimental studies, an end-to-end system is built to explore the potential of GNU Radio+USRP. For evaluating the system performance with various protocols and applications, we create several flowgraphs in Python using the existing signal processing blocks. We will discuss our system in more detail in the next section.

III. System Overview

As indicated earlier, our design goal is to implement SDRs as functional prototypes of MANET nodes. Thus, we attempt at integrating the TCP/IP stack on top of a customized PHYSICAL (PHY) layer and a MAC protocol running at the Data Link (DL) layer. To enable experimental studies using common networking applications, we build a number of SDR

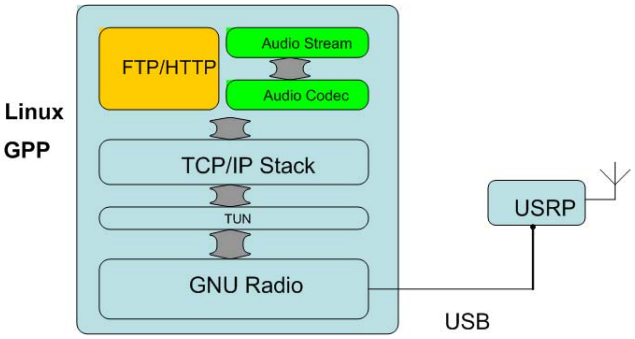

Fig. 2. The system architecture of USRP.

test nodes utilizing USRP and a GPP unit running Linux operating system. In this section, we describe the system architecture of such test nodes. Specifically, besides using USRP as the baseband processing unit and the MIMO RF frontend, the system is built on 3 components:

- **GNU Radio** provides PHY, MAC-DL layer functionality.
- **TUN** bridges TCP/IP and GNU Radio together.
- **Linux TCP/IP Stack** provides support of full NETWORK layer functionality.

In what follows, we discuss GNU Radio and TUN driver in more detail. The overall system architecture is shown in Fig. 2.

GNU Radio provides a relatively complete PHY support with various signal processing blocks. In this paper, we experiment with two modulation schemes: Gaussian Minimum Shift Keying (GMSK) and Differential Binary Phase Shift Keying (DBPSK) contained in Gauss and Fm blocks of GNU Radio.

As to the MAC layer function, we rely on an implementation of CSMA MAC protocol[2] provided by GNU Radio. Before sending data, the protocol checks the status of the channel. If the channel is busy, the protocol keeps executing *backoff* until the channel becomes idle. Once channel access is granted, the protocol start transmitting. In essence, the protocol only provides minimum channel access control to avoid collision. While the protocol's design satisfies performance profiling requirements of point-to-point transmission scenarios targeted in this paper, one has to rely on a more sophisticated protocol implemented by Click Router module [7] for multi-hop slot-based transmission scenarios. We note that the coverage of such scenario is outside the scope of this paper. That said, we analyze the effect of PHY parameters on the performance of the CSMA protocol utilized in this work.

[2]It is important to note that the current generation of GNU Radio is not appropriate for implementing MAC protocols such as TDMA requiring the use of timers for the purpose of information synchronization among communicating nodes.

While DL layer functions such as framing and CRC have been integrated into the latest GNU Radio release, FEC schemes are still not available. As such, our SDR test nodes utilize Forward Error Correction (FEC) codes at the DL layer in addition to basic framing and CRC function. Our FEC implementation takes advantage of Reed-Solomon (RS) codes available from RSCODE project [8]. However, we revise the implementation of RSCODE project to accommodate punctured RS codes. The use of punctured RS codes allows for transmitting incremental parity as oppose to retransmitting the full block for cases in which the number of accrued errors exceeds the error correction capability of the original block. Not only the use of punctured RS codes results in significant bandwidth savings, but it allows for reducing the delay which is specially important in the case of transmitting multimedia content such as audio.

As noted earlier, GNU Radio applications are primarily written using Python, and running in the user space of an operating system such as Linux. While there is no explicit interface available to connect GNU Radio with the protocol stack, the use of TUN driver allows for creating a virtual Ethernet device thereby providing an interface to bridge GNU Radio and Linux protocol stack together. Once a virtual device is created, GNU Radio is able to receive and send packets from/to Linux protocol stack via this device. Similarly, Linux protocol stack is able to communicate with GNU Radio via the same virtual device. Fig. 3 shows GNU Radio's flowgraphs of the outgoing and incoming paths of data.

IV. Experimental Studies

In this section, we present the results of experimental studies conducted in our testbed over a rich scattering link. The primary purpose of these studies is to evaluate the key performance metrics of the wireless channel including:

- **Channel Capacity:** The maximum number of bits that the channel can deliver in a second.
- **Round Trip Time:** The time elapsed for a round trip over the channel.
- **Packet Error Rate:** The ratio of the number of corrupted packets over the total number of transmitted packets.

The experimental studies fall into two categories. First, we tune up system configurations in order to discover the peak capacity of the channel as well as the Round Trip Time (RTT). Then, we measure and evaluate the performance of various common networking applications as well as audio applications utilizing configuration parameters that yield a reasonable channel quality. We also examine the effects of FEC. Specifically, a test node is comprised of a USRP set and a GPP unit. A USRP set consists of a motherboard (Rev.

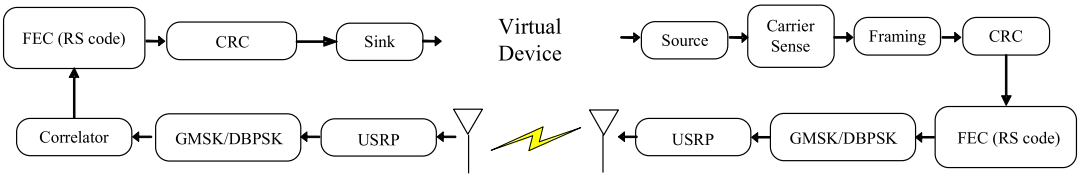

Fig. 3. The flowgraph of data on outgoing and incoming paths.

4.5) and two RFX2400 daughter boards each attached to an antenna. We have programmed the dual antenna radio to use Space-Time Block Codes (STBCs) of [9] when transmitting and Maximum Ratio Combining when receiving. The GPP unit carries a 1.8GHz Intel dual-core CPU, 3GB of memory, and runs Fedora Core 8 distribution of Linux operating operating system. We use release 3.1.2 of GNU Radio.

A. *Experiments on Channel Quality*

Channel capacity is affected by various issues. While the peak throughput is mainly limited by the speed of USB2 interface, PHY and MAC configurations have significant effects on the channel capacity. We measure channel capacity, Packet Error Rate (PER), and RTT with GMSK and DBPSK modulation schemes.

Fig. 4 shows that the choice of modulation scheme has a significant effect on channel capacity. In our experiments, GMSK achieves a much higher capacity than DBPSK. DBPSK reaches a peak throughput of 383.04Kbps when the raw bitrate is 400Kbps, while GMSK gets close to a peak throughput of 1368.8Kbps at a raw bitrate of 1400Kbps. It is intuitive that the capacity increases when the raw bitrate goes up. However, as the raw bitrate increases, the probability of experiencing collisions increases thereby limiting channel capacity. Hence, channel capacity drops for both modulation schemes as the raw bitate continues to increase beyond the point at which the peak capacity is achieved.

Fig. 5 and Fig. 6 illustrate average RTTs and PERs as a function of raw bitrate, respectively. As expected, RTT drops as the raw bitrate increases below a certain threshold. The raw bitrate threshold of RTT for DBPSK and GMSK is 400Kbps and 1000Kbps, respectively. For raw bitrates beyond the threshold, both GMSK and DBPSK show an increase of RTT as the result of experiencing a higher collision rate and a longer stay at backoff stages. Similarly, there is no significant PER for a raw bitrate below the associated threshold. Increasing the raw bitrate beyond the threshold point causes a rapid transition to a PER of 100%.

Furthermore, as the quality of wireless channel is affected by link fading and distance-related signal attenuation, we measure channel capacity and RTT in two scenarios of forming a line-of-sight and a rich scattering [3] link, respectively.

[3]The test nodes are about 10m away from one another and surrounded by major obstacles blocking the formation of a line-of-sight link.

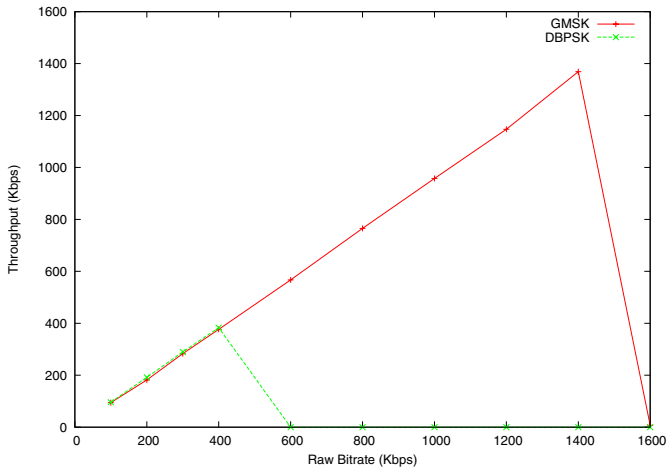

Fig. 4. The effects of modulation schemes on channel throughput.

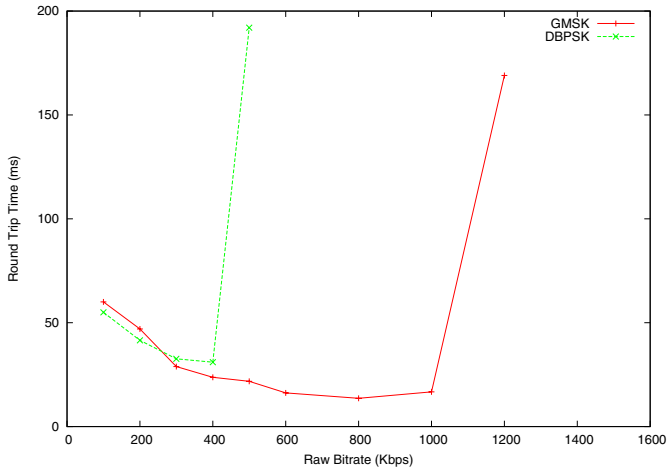

Fig. 5. The effect of the choice of modulation scheme on channel RTT.

While not shown here, we observe that the RTT is smaller in the case of experimenting with a line-of-sight link.

B. *The effects of Carrier Sense Threshold*

As we are using a CSMA MAC protocol, the value of Carrier Sense Threshold (CST) has a significant effect on the channel quality. We vary the value of CST to measure how the parameter settings of MAC affects the channel quality in terms of throughput, RTT, and PER.

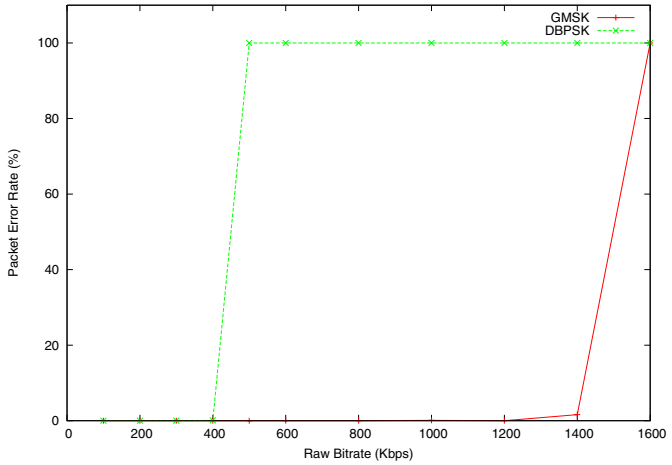

Fig. 6. The effect of the choice of modulation scheme on channel PER.

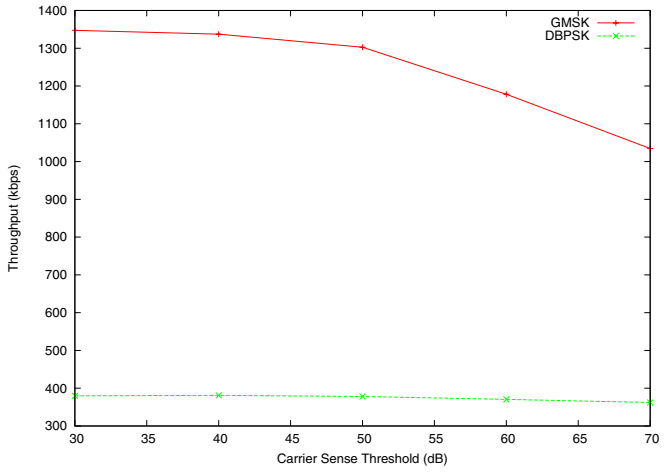

Fig. 7. An illustration of the achievable throughput as a function of the value of CST for GMSK and DBPSK modulation schemes.

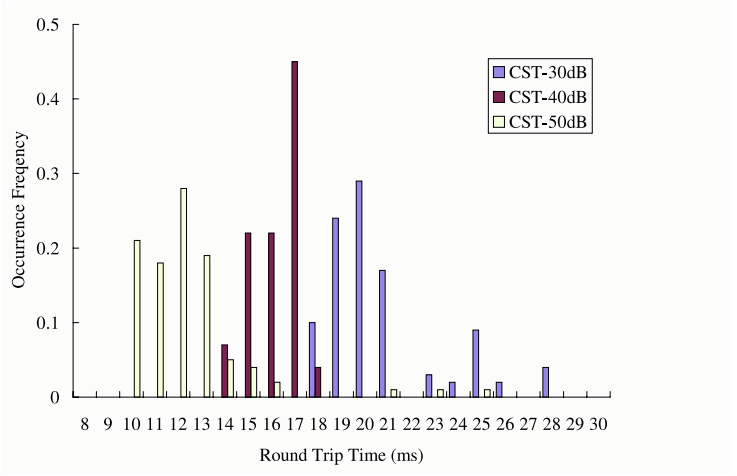

Fig. 8. The distribution of RTT for three different values of CST.

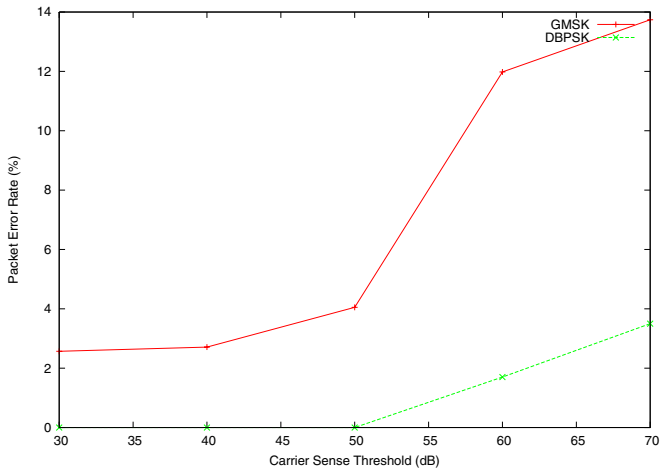

Fig. 9. An illustration of Packet Loss Rate as a function of the value of CST for GMSK and DBPSK modulation schemes.

Fig. 7 shows that channel capacity drops as the value of CST increases beyond 30dB. We justify the observation relying on the fact that an increase in the value of CST amplifies the probability of detecting false-collision thus decreasing the throughput. Furthermore, increasing the value of CST decreases the sensitivity of carrier sensing resulting in a lower backoff time. Consequently, the average waiting time before transmitting drops. Fig. 8 shows the distribution of the RTTs utilizing GMSK. It is worth noting that reaching a lower RTT comes at the cost of a higher PER. While not shown in the paper, the distribution of RTTs with DBPSK follows a similar pattern. Fig. 9 illustrates the variations of PER as a function of the value of CST for both GMSK and DBPSK modulation schemes. Both curves resemble hysteresis type behavior showing transitioning from low PERs to relatively high PERs as the value of CST increases.

Based on the results, the selection of raw bitrate and CST needs to address the tradeoff between a high throughput and a low RTT. In the following experiments, we use the parameters reported in Table I.

C. Experiments on Network Applications

Configuring our system according to the parameters of Table I, we examine the performance of a number of common network applications. In these experiments, the average completion time of FTP and the average retrieval time of HTTP access are measured as performance metrics. Each experiment is repeated 5 times to get the average results. The FTP server is $vsftpd$ built-in Linux Fedora Core 8 and $Apache2$ is used as the web server in our experiments. Our experiments show that the average duration of transferring a 5MB file and a 94KB page utilizing FTP and HTTP over our radio channel is 95.4sec and 1.23sec, respectively. The latter translates to an average sustained throughput of 53.67Kbps and 76.42Kbps for FTP and HTTP, respectively. The lower

TABLE I
SYSTEM SETTINGS

Application	FTP, HTTP, Audio
FEC	RS
MAC	CSMA
MAC Timing	backoff (1msec)
Rx-gain	45dB
Modulation	GMSK
Center Frequency	2.48GHz
Bit rate	1000kbps
Carrier Sense Threshold	30dB

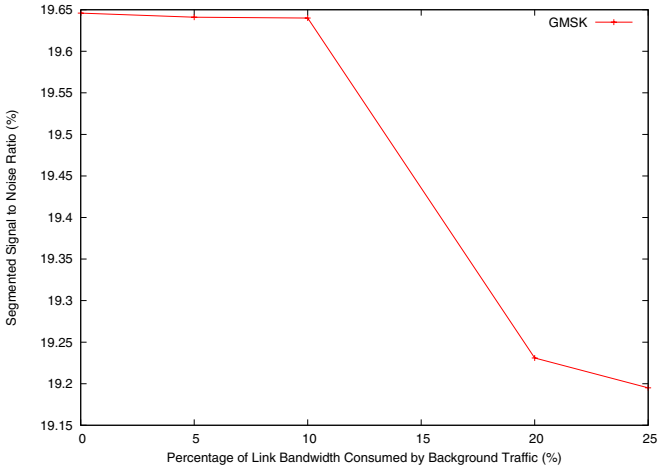

Fig. 10. An illustration of audio quality in the presence of background traffic.

throughput of FTP is related to a larger number of timeouts experienced by FTP as the result of facing channel variations over a longer period of time.

D. Experiments on Audio Application

Utilizing a LAME MP3 voice coder hosted in Linux user space, we evaluate the perceived quality of the received audio and voice sequences over the channel in the presence of background traffic. Background traffic is provided using a number of UDP session. We measure an RTT smaller than 150msec and a PER less than 10% in order to allow for reconstructing an audio sequence with an acceptable quality. As proposed by [10], we consider Segmented Signal to Noise Ratio (SSNR) as the metric of performance measuring the quality of audio. Fig. 10 illustrates the achievable SSNR as a function of background traffic percentage consuming the link bandwidth. As anticipated, the audio quality drops as background traffic increases. However, the drop in audio quality is not significant for background traffic consuming up to 25% of link bandwidth.

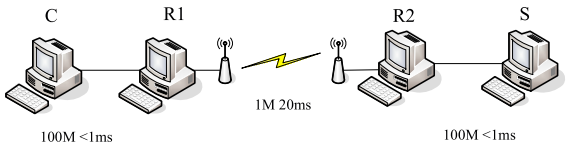

Fig. 11. The dumbbell topology used for FEC experiments.

E. Experiments with the effects of FEC

In order to evaluate the effects of FEC, we significantly increase the packet error rate by reducing the transmit antenna gain to 35dB. Then, we measure the effect of FEC on the performance of FTP in terms of completion time. We create a dumbbell topology with two SDR test nodes forming the bottleneck link of the topology.

Fig. 11 shows the topology of our experiment. The wireless link is formed between R1 and R2, operates in a full duplex mode, provides a nominal throughput of 1Mbps, and has an RTT of 20msec. The other links are wired with a nominal throughput of 100Mbps and an RTT of less than 1msec. An FTP server is set up at node S using the $vsftpd$ built in the Fedora Core 8 distribution of Linux. Five FTP requests start from node C to node S simultaneously each requesting a file of size 1.5MB. The FTP sessions utilize TCP BIC [11] available in the Linux Kernel for Fedora Core 8. A variety of TCP parameters are adjusted according to the TCP performance tuning guides of [12], [13] to optimize the TCP performance.

Fig. 12 shows the effects of enabling link layer FEC for the wireless channel. A FEC rate of 1 indicates no parity. For the experiment, the receive gain is set to 7dB causing a PER of 30% for a FEC rate of 1. Increasing the parity from a FEC rate of 1 to a FEC rate of 0.985, we observe significant improvements of performance of TCP measured in terms of FTP session completion times. However, increasing the strength of the FEC beyond 0.985 does not yield a proportional performance gain as the bandwidth overhead introduced by FEC results in reducing the link throughput available to data.

V. RELATED WORK

In this section, a number of projects working on the creation of SDR nodes or networks utilizing USRP are reviewed. Hydra [14] is a wireless network testbed developed by UT Austin. A Hydra node consists of an RF frontend USRP and a GPP machine running the frameworks of the Click Router module and GNU Radio. As a part of Hydra project, an experimental rate-adaptive MAC protocol has been implemented showing that the combination of USRP and GNU Radio may be used for cross layer design.

In [15], GNU Radio and Click Router module are used in conjunction with one another to support an integrated frame-

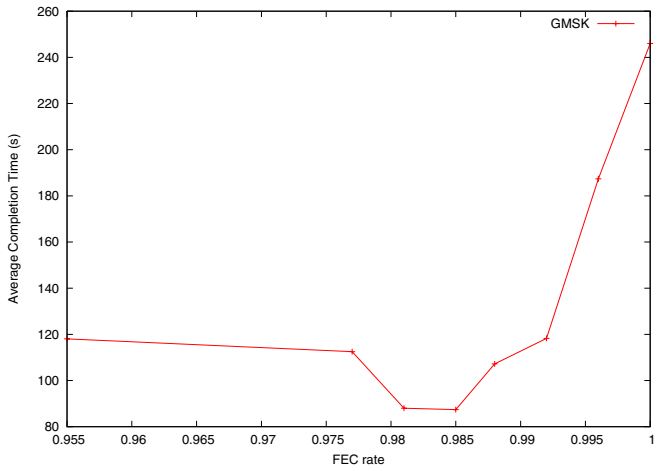

Fig. 12. The effects of FEC rate on the performance of TCP.

work of protocol development at the PHY and higher layers. The work demonstrates an implementation of a simple TDMA protocol which requires further work in terms of time slot synchronization and time slot size. An interesting aspect of the work of [15] is that it presents three different ways of combining GNU Radio with Click Router module and compares them with the respect to development cost.

In [16], the transmit and receive latency of the USRP are studied. Specifically, the work measures the time it takes between generating a sample by a GPP and transmitting that sample out through USRP. This latency is typically less than 1msec representing a major obstacle for the development of any MAC layer protocols requiring time synchronization. The development of the next generation of USRP, namely USRP2 to be released in near future, is anticipated to help address the issue through the use of a more powerful FPGA allowing for hosting time-critical components thereby satisfying the requirements of MAC protocol development.

VI. Conclusion

In this paper, we reported on the implementation of a MIMO SDR platform forming an operational MANET test node. Our SDR platform utilized USRP including a motherboard for baseband processing, two daughter boards for RF frontend processing, and an embedded Intel dual-core GPP unit hosting Linux operating system. We used GNU Radio to program the PHY and DATA LINK layers of USRP. We also implemented the TCP/IP stack on the SDR using the TAP/TUN driver of the Linux Kernel. We were able to run a number of applications including file transfer, web content, stored audio, and live speech delivery. We further reported the results of our performance benchmarking related to these applications.

We are currently working on the design and implementation of a hybrid anycast MAC protocol. The MAC protocol behaves similar to CSMA when the number of MANET nodes is smaller than a threshold or when there is no synchrony information available. As the number of nodes increases and when node synchronization information becomes available, the protocol switches to a TDMA-like mode of operation. Our anycast protocol allows for providing link quality information to the Hazy Sighted Link State (HSLS) routing protocol thereby improving its performance.

References

[1] -, "USRP Product Description," available at http://www.ettus.com/index.html.

[2] E. Blossom, "Exploring GNU Radio," Nov. 2004, available at http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html.

[3] T. S. Gray, *A First Course in Electronics, Electron Tubes, and Associated Circuits*, 1954.

[4] M. Ettus, "How to Get Ready for the USRP2," Jan. 2008, available at http://www.nabble.com/How-to-get-ready-for-the-USRP2-td14857846.html.

[5] -, "ADROIT: GNU Radio Architectural Changes," May 2007, available at http://acert.ir.bbn.com/downloads/adroit/gnuradio-architectural-enhancements-3.pdf.

[6] ——, "MBLOCK, Inband, USRP2 Future?" June 2008, available at http://www.nabble.com/mblock,inband,-usrp2-future—td18152536.html.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The Click Modular Router," *ACM Trans. on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[8] -, "RS Code Project," Jan. 2000, available at http://www.beartronics.com/rscode.sourceforge.net/.

[9] S. Alamouti, "A Simple Transmitter Diversity Scheme for Wireless Communications," *IEEE JSAC*, vol. 16, pp. 1451 – 1458, Oct. 1998.

[10] A. Khalifeh and H. Yousefi'zadeh, "An Optimal UEP Scheme of Audio Transmission over MIMO Wireless Links," in *Proc. IEEE WCNC*, Mar. 2008.

[11] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *Proc. IEEE INFOCOM*, 2004.

[12] J. Mahdavi, "Enabling High Performance Data Transfers on Hosts," *technical note, Pittsburgh Supercomputing Center*, Dec. 1997, available at http://www.psc.edu/networking/perf_tune.html.

[13] -, "TCP Tuning Guide," Nov. 2005, distributed Systems Department, Available at http://dsd.lbl.gov/TCP-tuning/TCP-tuning.html.

[14] K. Mandke, S. Choi, G. Kim, R. Grant, R. C. Daniels, W. Kim, R. W. H. Jr., and S. Nettles, "Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed," in *Proc. IEEE VTC*, Dublin, Ireland, Apr. 2007.

[15] R. Dhar, G. George, A. Malani, and P. Steenkiste, "Supporting Integrated MAC and PHY Software Development for the USRP SDR," in *Proc. IEEE Workshop on Networking Technologies for Software Defined Radio Networks*, Sept. 2006.

[16] T. Schmid, O. Sekkat, and M. B. Srivastava, "An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios," in *Proc. Intenrational Conference on Mobile Computing and Networking*, Sept. 2007.