Spectrum Agile Radios Project Report

Ji Woong Lee
Pedram Keshavarz
Miklos Christine
Sherman Ng
Sofie Pollin
Libin Jiang
Jean Walrand
Hidekazu Miyoshi

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-142
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-142.html

October 31, 2008

Spectrum Agile Radios Project Report

Jiwoong Lee, Pedram Keshavarz, Miklos Christine, Sherman Ng

Sofie Pollin, Libin Jiang, Jean Walrand, and †Hidekazu Miyoshi

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720

†Innovation Core SEI Inc. Santa Clara, California 94720

{porce,pedram,mchristine,sng629}@berkeley.edu,
{pollins,ljiang,wlr}@eecs.berkeley.edu, and †miyoshi-hidekazu@sei.co.jp

Abstract

Spectrum Agile Radios project studies dynamic channel selection problem upon random encountering of uncontrolled interference in a general wireless network environment. This research targets problem formulation, development of channel selection algorithm, selection of platform and prototyping the proof-of-concept in lab level. Among several candidates, MadWiFi/Ath5K is selected for platform and is studied in-depth to learn its capabilities and limitations. Realtime statistics collection with respect to wireless channel and device buffer is an ongoing work. Algorithms will be designed on top of the statistics collected and economic behavior of users. Automatic Rate Fallback issue of WiFi is discussed to improve the rate adaptation on top of best channel selection. Finally an optimal AP selection algorithm is proposed for corporate type wireless networks.

I. Project Overview

A. *Motivation and Scope*

The University of California at Berkeley (PI: Prof. Jean Walrand) and Innovation Core SEI have been working on a joint research, entitled Spectrum Agile Radio since Oct 1, 2007. The project focused on understanding the nature of wireless channel interference and congestion, to design an algorithm to optimally and dynamically search and change the current channel based on realtime statistics of wireless medium activities and economic user incentive, and prototyping a platform to show the proof-of-concept.

For the prototyping, it is important to first select the right platform in the first stage of the project since the platform will essentially determine the practical limitations that limit the scope of possible algorithm assumptions, the amount of resources required for prototyping efforts, and the range of experiment scenarios that can be performed. We first surveyed capabilities and limitations of a set of potential platforms. As our choice of platform, we selected the Linux based MadWiFi platform (and Ath5K as its friend). The platform selection issues are detailed in section II.

This report summarizes the study and progress results obtained for the project over the course of one year. We first focus on the results obtained so far for developing the prototyping framework. We briefly describe the driver structure first and some crucial capabilities including channel selection and medium statistics collection. Next we present a basic software structure for prototyping that will later embed the algorithms that we design. It is followed by a discussion how to track of client/AP link layer queue size, which is additional information required to optimally select jointly the channel and the AP. In section IV, we discuss an access point selection scheme for the optimal operation of network. For a chosen AP and channel combination, we are then interested in how we can further improve performance by considering link adaptation. Current issues in WiFi automatic rate fallback are discussed.

The progress of this research has been made possible through commendable team efforts: Pedram Keshavarz, Miklos Christine, and Sherman Ng as undergrad researcher as locomotives of the project, Jiwoong Lee and Michael Krishnan as PhD student mentors, Sofie Pollin as post-doctoral researcher providing insightful discussion, Prof. Jean

Walrand as Principal Investigator. Libin Jiang contributed to algorithm design and Hidekazu Miyoshi contributed to research motivation based on realistic market demand.

B. Platform selection

The development of a spectrum agile radio requires in a broad range of scenarios, we need a radio that can operate in a large set of operation modes to mimic clients, access points, interferer and monitors. We also need a highly configurable radio where the physical layer can be tuned, e.g., for the link adaptation, and the Medium Access Control (MAC) layer allows for a fast and precise timing controllability of its functionality. To collect statistics about the environment, we need open access to realtime medium statistics. Finally, these options should be available at a reasonable development cost. We give an overview of various boards that are available and were considered for our prototyping:

- **DINI board** series from the DINI group(http://www.dinigroup.com). The DINI series provide very powerful platforms for multi-radio multichannel PHY architecture with full open access to the firmware level. Firmware access enables accurate and precise timing control that a high speed spectrum agility algorithm requires. A critical disadvantage of this board is it provides a primitive dummy MAC only which is far from IEEE 802.11 compatibility. Also it is priced around USD 20,000 for one board set.
- **3DSP** (http://www.3dsp.com) provides a full software-based MAC that is compatible with IEEE 802.11. The cost of this platform is very reasonable. But it has issues in precise timing control and accessibility to the PHY layer, which is closed to developers.
- **MadWiFi**, which is an open driver for Atheros WiFi chipsets, provides a strong compatibility to IEEE 802.11 MAC and PHY. Its management plane is completely open but most of MAC and PHY code are implemented on PCMCIA type firmware, that are not open source. Also, the management plane works in the Linux kernel mode whose timing resolution is approximately 1msec, which is too large for spectrum agile operations.
- **WARP platform**, which is a product from Prof. Knightly's group in Rice University. This is a XiLinx FPGA based platform that enables open programming. At the time of survey, however, it supported ALOHA/CSMA MAC far from the compatibility to WiFi. It is priced USD a few thousands per board.
- **The GNU Radio** is a free software toolkit for learning about, building, and deploying software-defined radio systems. It is fully programmable, and gets more and more support in the research community. The platform used is the Universal Software Radio Peripheral (USRP), that only allows up to 16MHz of bandwidth. Moreover, because of software processing, the communication speed achieved is typically very low. As a result, this platform is not yet sufficient to achieve 802.11 performance.
- Platforms based on **802.15.4 chipsets** that can dynamically select up to 16 channels. We explored options that come with a full MAC implementation but suffer from very limited accessibility to and programmability of that MAC, such as the Chipcon CC2430 development kit or National Instrument's Bumblebee boards. Next, we tested UC Berkeley's CALinx expansion board that is used in the CS150 class. It comes with a Chipcon cc2420 chip and an FPGA. Although this platform is fully programmable, it comes with no protocol support at all.

While it is true that none of above satisfies stated requirements, we valued most the maturity of Medium Access Control layer, the broad WiFi compatibility and open architecture of management plane of MadWiFi.

It will be useful to describe some of capabilities and limitations of MadWiFi. MadWiFi has following interesting capabilities that might be useful for future algorithm design: noise floor sensing, per-packet RSSI sensing, Tx power control, rate control, power saving, modulation selection, CCA threshold control, beaconing, queueing, frame control, channel switching, multichannel management, long distance timing control, extended range support, antenna diversity and timer control. However, followings are known to be not possible: realtime CCA tracking, pausing CSMA timers/counter and direct firmware controls.

The communication between MadWiFi driver and HAL is done by IRQ and DMA. IRQ is used for software beacon alert, bumping TX trigger level, RX notice, TX notice and MIB handling. DMA is used by MadWiFi to notify HAL to copy the outbound packet.

II. MadWiFi and Ath5K Structure

MadWiFi driver was developed by a group of Atheros chipset employees. Atheros WiFi chipsets are known to support a wide range of frequency and has deep controllability over the radios. MadWiFi runs on top of HAL(Hardware Abstraction Layer) which is a wrapper around the hardware registers that has direct communication to the firmware of chipsets. HAL was devised to compromise between the open source community and the regulatory agencies.

MadWiFi itself is a huge package; its source consists of 205K lines plus binary modules(HAL). In the biggest picture, MadWiFi consists of 4 modules - `net80211`, `ath`, `ath_rate` and various utilities. `net80211` is responsible for 802.11 frame operation and all management operations including association, authentication and roaming. `ath` has Atheros specific callbacks for `net80211` and hardware access interface through HAL. `ath_rate` provides automatic rate fallback.

A. *MadWiFi Structure Overview*

MadWiFi is essentially an open source WLAN driver for Linux. This section discusses the basics of transmission and reception, by giving an overview of the structs used. The most common structs used are `net_device`, `ieee80211vap` (allows a physical access point to behave like a multiple access point), `ieee80211_node` (keeps node-specific information), and `sk_buff` (socket buffer). `if_ath.c` is responsible for starting transmission, by invoking functions in the `80211net` subdirectory (first function is `ath_hardstart`) `ath_hardstart` calls `ath_tx_start` and `ieee80211_encap` (to encapsulate the packet). Reception is handled by `ath_rx_tasklet` (`in_ath.c`), which calls other functions such as `ieee80211_decap` or `ieee80211_input`.

B. *Ath5k Structure Overview*

Ath5k is a Linux wireless driver that is based on the MadWiFi wireless driver and OpenHAL. The main difference between MadWiFi and Ath5k is that Ath5k directly calls hardware functions and writes to the hardware registers of the Atheros wireless card. Ath5k consists of 10 files: `ath5k.h`, `base.c`, `base.h`, `debug.c`, `debug.h`, `hw.c`, `hw.h`, `initvals.c`, `phy.c`, and `reg.h`. `ath5k.h` defines the structure of the hardware abstraction layer and contains the settings of the driver, like transmission rate, reception status, an driver mode. The main files of the Ath5k driver are `base.c`, `base.h`, `hw.c`, `hw.h`, and `phy.c`. These files contain functions that are responsible for the transmission of packets, reception of packets, driver initialization, and other hardware functionalities. `initvals.c` fills in the registers in the wireless card with initial values. `reg.h` holds the values for the hardware registers of Atheros 5212, 5211, and 5210 cards. `reg.h` is directly derived from the OpenHAL reverse engineering efforts to produce an open source hardware abstraction layer and allow open source drivers like Ath5k to directly access hardware registers of the wireless card using hardware functions.

C. *Channel Selection*

The channel switching delay in MadWiFi is roughly 30-50ms. Our initial goal was to reduce this delay to 5-10ms by modifying the code. MadWiFi relies on `iwconfig` to change/set the channel. As a result, an analysis of the `iwconfig` package was a necessary step in understanding how MadWiFi changes the frequency of transmission. In this intermediate step, we looked into the `wireless_tools.29` package, and realized that the IO flag responsible for setting the frequency is `SIOCSIWFREQ` (which corresponds to `0xB04` in memory). Searching the MadWiFi source code for this flag pointed us to the function `ieee80211_ioctl_siwfreq`, which is located in `ieee80211_wireless.c`. This function takes a device name, a command tag, and a frequency as it's arguments, and sets the frequency of the device to the desired value.

D. *Statistics Collection*

Because the Ath5k driver has direct access to the hardware register values obtained from the OpenHAL, the statistics were obtained by directly reading the register values using the function `ath5k_hw_reg_read`. Our goal is to collect statistics on the fractions of time that the communication channel is busy and idle, and to do that, we read and printed the values of the registers `AR5K_PROFCNT_RXCLR` and `AR5K_PROFCNT_CYCLE`. The `AR5K_PROFCNT_RXCLR` register is supposed to contain the amount of time that the communication channel is busy and the `AR5K_PROFCNT_CYCLE` register is supposed to contain the total time that elapsed. The biggest problem with reading and analyzing these registers is that these registers reset after a certain period of time, which makes the printing output of these registers seem inconsistent. One property that always holds is that the value in the `AR5K_PROFCNT_RXCLR` register is always a fraction of the value in the `AR5K_PROFCNT_CYCLE` register, which is reasonable since the busy channel time should be a fraction of the total time elapsed. From experiments, the fraction of the value of the clear register divided by the cycle register is approximately .15 when the channel is clear and between .4 and .8 when the wireless card is sending packets.

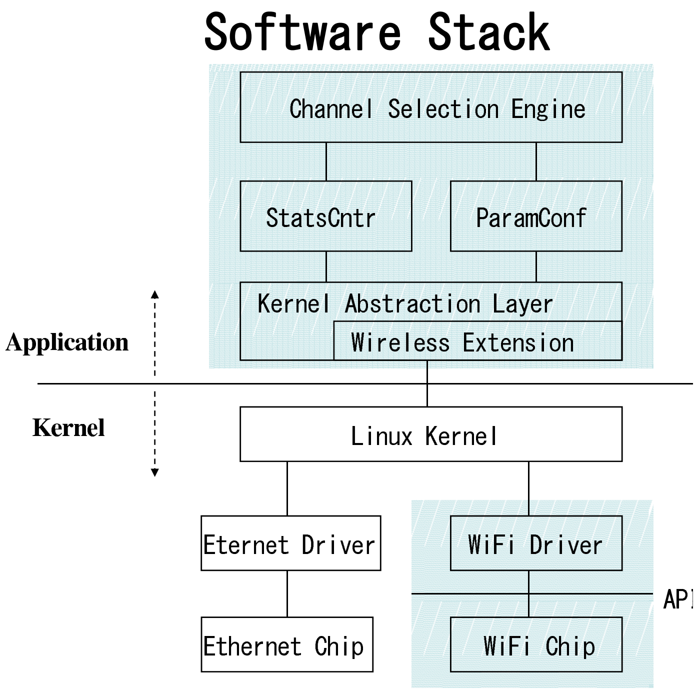

Fig. 1. Software Architecture

III. Prototyping

A. Software Architecture

1) Statistic Counter Collector Structure:

```
cb_get_basic_info()     // get basic configuration information
cb_get_stats()          // get statistic information

struct cb_ifstats {
    u_int32_t tx_packets;
    u_int32_t tx_mgmt;
    u_int32_t tx_xretries;
    u_int32_t tx_shortretry;
    u_int32_t tx_longretry;
    u_int32_t tx_fifoerr;
    u_int32_t rx_orn;
    u_int32_t rx_crcerr;
    u_int32_t rx_fifoerr;
    u_int32_t rx_badcrypt;
    u_int32_t rx_phyerr;
    u_int32_t rx_tooshort;
    u_int32_t rx_toobig;
    u_int32_t rx_packets;
    u_int32_t rx_mgt;
    u_int32_t tx_rssi;
    u_int32_t rx_rssi;
    u_int8_t link_qual;
    u_int8_t sig_level;
    u_int8_t noise_level;
};
```

2) *Parameter Configuration Interface:*

```
    cb_set_chan(channel)        // set channel
    cb_set_txpower(level)       // set txpower
    cb_set_fragoffset(length) // set max fragmentation
    cb_set_rate (rate)          // set PHY rate
    cb_set_config()             // set basic configuration
```

3) Wireless Extension: Wireless extension supports lots of useful interfaces to get statistics and basic configuration of chips, and also to set configuration parameters. For instance the `iwconfig` command is defined here. Wireless extension basically consists of two `C` source files, `iwConfig.c` and `iwLib.c`. `iwLib.c` interfaces with Linux kernel and eventually calls `ioctl()`.

4) Sample Code: This sample code changes channel to a new one at every 5 seconds ($1 \rightarrow 6 \rightarrow 11 \rightarrow 6 \rightarrow 11 \rightarrow \cdots$).

```
int cb_chan_select()
{
    static int cur_chan = 1;

    if(cur_chan == 11)  cur_chan = 1;
        else cur_chan += 5;

    return(cur_chan);
}

main(int argc, char *argv[])
{
    int s;
    int chan;

    s = socket(AF_INET, SOCK_DGRAM, 0);
    if (s < 0)  err(1, "socket");
loop:
    cb_get_stats(s, IFNAME_WIFI, &ifstats);   //IFNAME_WIFI:  wifi0
    chan = cb_chan_select();                  // pick up a channel
    cb_set_chan(s, IFNAME_ATH, chan);         // IFNAME_ATH:  ath0

    sleep(5);
    goto loop;
    return 0;
}
```

B. *Measuring Kernel Queue size*

We aim to measure the kernel queue size, and print out its value through time. At first, we tried to look for a `sysctl` variable that stores the queue length. To be certain, we printed all `sysctl` variables, and found no one relating to queue length. We then looked at the kernel code to get more insight. When a packet is ready for transmission, the function `dev_queue_xmit` is called. This function adds the packet to the device's queue (`dev->qdisk->enqueue`). The packet on top of the queue is then transmitted using `qdisk_restart(dev)`. After this, the queue is dequeued, to bring the next packet higher up in priority. The `qdisk` structure contains a stats field. The stats field has a variable, `qlen`, which collects the value of the queue length. By monitoring this value and printing it, we can determine the size of the network device queue at different times. This can be achieved by putting `printk` statements in the code to get the values for queue length.

IV. Algorithm Study

A. *AP Selection Algorithm*

In [1], we proposed the following simple AP selection algorithm, coupled with proper packet scheduling policies. (1) On the user (or "station") side, each user selfishly selects an AP that can gives him the highest throughput; (2) On the AP side, each AP implements a scheduling policy to allocates its bandwidth, such that the total utility of its intra-cell users is maximized.

It was shown in [1] that the algorithm optimizes the performance of the whole system under some reasonable assumptions. Also, since all decisions (AP selection by the users and intra-cell scheduling by the AP's) are made locally, the distributed algorithm is very easy to implement. Specifically, a possible implementation is as follows.

1) Before association, a user sends a REQUEST packet to all available AP's that he can choose from. The REQUEST packet also reports the user's application type, such as "data", "voice", or "video". Each application type is mapped to a certain utility function.
2) According to the utility function, each AP that has received the REQUEST packet computes its optimal bandwidth allocation assuming that the user would select the AP. (The allocation also depends on the current intra-cell users and their utility functions.) Then, the AP sends a REPLY packet to the user, reporting the "proposed" bandwidth to be allocated to the user.
3) After receiving the REPLY packets from all available APs, the user selects the AP with the highest proposed bandwidth.
4) Finally, the selected AP allocates its bandwidth as computed in step 2. The bandwidth allocations of other AP's keep unchanged.

B. ARF Issue and Simulations

Basic ARF algorithms do not take into account different types of traffic loss. Modulating to a lower rate is not always the solution to compensate for loss. In the case of loss due to collisions, reducing the data rate will increase the packet length. This will increase the chances of collisions since the packet is longer and will be in the channel for a longer period of time. Our approach is to gather statistics to calculate the probability that each type of loss occurred.

The ARF algorithm is implemented within the MAC 802.11 code of NS-2. Each node has an instance of the MAC 802.11 layer and within each instance, the ARF algorithm is running. The issue with modifying the data rate is knowing whether or not changing the data rate changes any other parameter. There is little documentation on the updates from different versions of NS-2, which makes it difficult to understand the flow of the simulator.

A monitoring tool is implemented to keep a count of successful and unsuccessful packets, as well as collisions and error due to a poor channel. This is used to modify the simulation's parameters to meet the specified conditions. This tool helped to find the correct distances for the nodes in our simulation. It also helped to find the general throughput of each link.

The simulator mostly uses thresholds to determine whether or not each frame is correctly received by the receiver. This is another issue, since it uses a purely deterministic methodology when receiving packets, while real world wireless networks cannot be simplified in that manner. The simulator has no error models that take into account any fading or modulation errors. The simulator sets 2 main thresholds, the receiving and the carrier-sensing thresholds. If a packet falls outside the CS threshold, it is not noticed by the node. If it is within the CS threshold, but outside of the receiving threshold, it is counted as an error. Everything else within the receiving threshold is counted as successful. To simulate a more realistic model, the addition of an error model has been added to simulation. The error model takes into account the modulation bitrate that is used in the transmission and also calculates the probability of error given each rate. This allows each packet within the receiving threshold to still encounter an error. Taking into account these issues will produce a more realistic simulation to test our theory on improving the wireless network.

The application is a discrete event simulator targeted at networking research. NS-2 is a C++ object oriented simulator that incorporates an OTcl interpreter as an interface. The purpose of the simulation is to test the impact of an auto-rate fallback algorithm (ARF) given statistical information. Assuming that the nodes know if a loss is caused by a poor channel as opposed to a collision, we adapt certain parameters, such as the modulation rate, to increase the performance of the network.

The simulation scenario is a small adhoc network with 4 nodes in a line. 2 nodes will be sending packets while the other 2 will be receiving. The nodes are spaced appropriately such that the nodes are far enough to not have a lossless channel. The first receiving node is also susceptible to collisions since the sending nodes are on opposing sides of the receiver and cannot hear each other sending. The nodes are set with the appropriate routing tables, carrier sense threshold, receiving threshold, and other parameters to mimic the IEEE802.11b specifications. The auto-rate fallback algorithm states that after N successful packet transmissions increase the data rate if possible. For every M unsuccessful transmissions, decrease to a lower data rate.

There are 2 algorithms and one control tested with each given configuration. The control test uses no ARF algorithm and continues to send at a constant rate regardless of any losses. The second test uses the most common ARF algorithm that modifies the rates treating all losses the same. The final test modifies the modulation rate only if loss is caused by poor channel conditions, and does not acknowledge collisions as loss. Given that the types of losses are known, test if the ARF algorithm increases the performance of the network.

Future tests will include modifications to the packet length, contention window, or sensing thresholds.

V. Conclusion and Future Direction

Spectrum agility is expected to be one of the fundamental features of future wireless technologies. The right understanding of wireless channel interference and congestion problems and the right resolution of them will be driving force of future radio devices. For the proof-of-concept purpose, MadWiFi/Ath5K platform is selected and studied in-depth. This platform will host various kinds of candidate algorithms per scenario. As an example, we proposed an AP selection algorithm which optimizes the throughput performance of a network. Further exploitation of medium statistics are under study. In the second stage of the research we expect a working prototype will play a role of deeper research and initiative for future market driving.

References

[1] L. Jiang; S. Parekh and J. Walrand, ”Base Station Association Game in Multi-Cell Wireless Networks,” IEEE Wireless Communications and Networking Conference (WCNC) 2008, pp.1616-1621, March 31-April 3 2008