7

Supporting Cognitive Radio Network Protocols on Software-Defined Radios

George Nychis, Srinivasan Seshan, and Peter Steenkiste

7.1 Introduction

Over the past few years, an increasingly diverse and ever-changing wireless spectrum has created a need for cognitive radio networks. Such networks leverage spectrum sensing and information from each layer in the protocol stack to overcome spectrum diversity by adapting all layers (e.g., the MAC and PHY) on the fly. By doing so, cognitive radios can achieve the greatest level of performance, given the current networking conditions. For example, in areas where access to the spectrum is highly contended, the radio can switch from using a carrier sense multiple access (CSMA) MAC protocol, to a time division multiple access protocol that reduces overhead in accessing the spectrum to increase capacity and reduce collisions. Despite the increased recent activity in cognitive radio networks, supporting the development of protocols at the MAC and PHY layers, as well as cross-layer optimizations for such networks, has been extremely challenging. Commodity wireless hardware does not facilitate such development, because the majority of MAC functionality is placed on the network interface (NIC) hardware, where programmability is limited and access to the software that runs on the NIC is often restricted.

The limited programmability of wireless NICs makes Software-Defined Radios (SDRs) an attractive alternative for building cognitive radio network protocols. SDRs implement the majority of functionality, including the physical and link layers, in software running on commodity hardware, making all layers of the protocol stack easy to modify. The SDR hardware simply translates the signals between the RF and digital domains, and the software does the majority of the processing. The processing of the digitized samples in SDR architectures[5,8,14,16,17] is commonly distributed across various processing units – including FPGAs and CPUs located on the SDR device and the host machine. Unfortunately, the high degree of flexibility offered by SDRs does not automatically

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

lead to flexibility in cognitive radio network protocol implementations. The heterogeneous processing units and interconnecting buses in such architectures often contribute large latencies and jitter in processing. These latencies can severely cripple the ability of the MAC layer, for example, to effectively respond to channel conditions, time transmissions, and communicate in a timely manner, which reduces the performance of the radio. These are important functions in a cognitive radio network, because being able to respond timely to spectrum conditions is the essence of a cognitive protocol. Placing the functionality on the SDR radio hardware to avoid these latencies would again make it difficult to develop, which is what made SDRs an attractive platform over NICs.

In this chapter, we explore an API enabled through the addition of a control channel and metadata that enables rich information and control between the radio hardware and the host, allowing adaptation at all layers on a per-packet basis. Additionally, we present a novel *split-functionality* approach to implementing core cognitive radio network functions that enables high-performance MAC (a common layer for adaptation) and cross-layer implementations. In this approach, a part of the cognitive radio function is placed on the SDR radio hardware for performance reasons, and a part on the host CPU to maintain easy customization of the protocol (at any layer) and radio. A set of novel techniques is presented to achieve the split of the functions, and how to properly distribute the functionality between the processing units on the hardware. Finally, we present a design and implementation of a cognitive switching layer that would allow control of the radio, such as the MAC protocol it is running, in the future Internet. Such a layer could be accessible by the future Internet through a global controller, or allow for local coordination within a single LAN.

This chapter makes the following contributions: Given that cognitive radio networks strongly leverage adaptation at the MAC layer, we place a major focus of our work at this layer. We identify a set of core protocol functions, from which many MAC and cognitive protocol layers are built, as well as cross-layer implementations that must be implemented close to the radio for performance and efficiency reasons. We define a *split-functionality* architecture that allows the functions to be implemented near the radio hardware while maintaining control on the host CPU through an API. We present an implementation of our architecture using the GNU Radio[5] and USRP[14] SDR platform. Using our implementation, we characterize the performance-flexibility tradeoffs for key protocol features. For example, our results show three orders of magnitude greater response time of the radio to spectrum conditions. Finally, we use our implementation for an end-to-end evaluation of the split-functionality architecture. We show how the system can support high-performance cognitive network protocols by first implementing 802.11-like and Bluetooth-like protocols for experimentation over the air, and then a cognitive protocol that can switch between the two MAC protocols based on current network conditions. The rest of the chapter is organized as

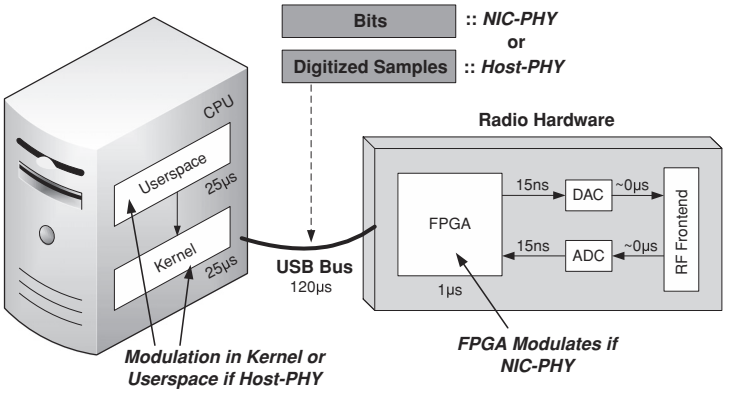

Figure 7.1. Generic SDR architecture.

follows. We discuss current radio architecture and its impact on MAC protocol development in Section 7.2. In Sections 7.3 and 7.4, we explore the core wireless MAC and cognitive protocol requirements and introduce our *split-functionality* architecture. Section 7.5 provides details for each component implementation with evaluation results. Finally, we present end-to-end evaluation results, related work, and a summary of our results in Sections 7.6 through 7.8.

7.2 Software-Defined Radio Architecture and Challenges

Software-defined radio architecture varies in the exact nature of the processing units and interconnecting buses; however, a common architecture which will be the focus of this work is shown in Figure 7.1. The frontend is responsible for converting the signal between the RF domain and an intermediate frequency, and the A/D and D/A components convert the signal between the analog and the digital domain. Physical and higher-layer processing of the digitized signal are executed on one or more processing units. Typically, there is at least an FPGA or DSP close to the frontend. The frontend, D/A, A/D, and FPGA are usually placed on a network card that is connected to the host CPU by a standard bus. In the next section, we quantify the delays between each of these components. In a *Host-PHY* architecture, the majority of the signal processing (e.g., modulation) would be done on the host-machine in userspace or in the kernel. On ther other hand, in a *NIC-PHY* architecture, this signal processing is done on the FPGA or a similar processing unit on the radio hardware.

Unfortunately, SDRs have fallen short in providing rich physical layer information to the protocol stack and have failed to provide high-performance flexible protocol (e.g., MAC and cross-layer) implementations. Functionality implemented on the radio hardware (e.g., modulation) will have good performance but lack flexibility and will be hard to modify. We refer to this architecture as

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

NIC-PHY, exemplified by WARP.[17] The opposite is true of functionality in a *host-PHY* architecture where the functionality is placed on the host CPU (e.g., GNU Radio[5] and the USRP[14]). A *host-PHY* architecture has been considered incapable of supporting even core protocol techniques (e.g., carrier sense) due to the large processing delays inherent to the architecture.[12,15] However, the goal of our work is to support high-performance flexible protocol implementations in a *host-PHY* architecture to enable many MAC protocols, cognitive protocol techniques, and cross-layer optimizations such as those proposed between the MAC and PHY layers.[4,6,7] Given that cognitive radio network protocols often heavily adapt at the MAC layer, we focus the majority of our work on optimizing performance and development at this layer while increasing the reactiveness of the radio to the spectrum.

In the next section, we explore delay and jitter measurements in the *host-PHY* architecture, which are the major limiting factor on the performance of MAC implementations[12,15] and the responsiveness of the radio to cognitive protocol techniques. By understanding the sources of the delay and quantifying them, we can explore a *split-functionality* approach (Section 7.4) that places pieces of techniques (e.g., carrier sense) before specific sources of delay or jitter to achieve greater performance. Therefore, it is important to not only understand the total delay in the system, but the delay between each major component in Figure 7.1.

7.2.1 Delay in Software-Defined Radios

In this section, we explore sources of delay in a *host-PHY* architecture for the purpose of understanding why MAC and cognitive network protocol implementations have suffered in performance, such that we can overcome the delay. We use GNU Radio and the USRP for this work, which[12] present as a delay measurement for the platform; however, they focus on user-level measurements, largely ignoring precise measurement of delays between the kernel and userspace, and kernel and the radio hardware. Such measurements are important because they can provide insight into whether implementing protocol techniques in the kernel is sufficient to overcome the performance problems associated with user level implementations.

To obtain precise user and kernel-level measurements, we modified the Linux kernel to record nanosecond precision timestamps on the USB data between the host and radio hardware at various times in the transmission and receive process. All user-level timestamps are taken in user space right before data is submitted to the kernel, or when the data is first read in user space. To measure as close to the USB bus as possible, timestamps in the kernel are recorded at the last point in the kernel's USB driver before the DMA write request is generated, or after a DMA read request. To measure the roundtrip time between GNU Radio

Table 7.1. *SDR* host-PHY *Architectural Delay Measurements*

Units: μs	Avg	SDev	Min	Max
User→Kernel	24	10	22	213
Kernel→User	27	89	13	7000
4096 Kernel→FPGA	291	62	204	360
512 Kernel→FPGA	148	35	90	193
GNU Radio→FPGA	612	789	289	9000

(in user space) and the FPGA, we introduce a ping command on a control channel that we implement (Section 7.4.2). Using the measurements described earlier, we are also able to identify the sources of the delay by calculating the user-to-kernel space delay, kernel-to-user space delay, and roundtrip time between the kernel and FPGA based on ping. We ran the user process at the highest priority to minimize scheduling delay. We used the default 4,096 byte USB transfer block size for all experiments, and then perform an additional kernel to FPGA RTT experiment using a 512 byte transfer block size, the minimum possible, in an attempt to minimize queueing delay.

Averaged over 1,000 experiments, the delay results are presented in Table 7.1. The results show that the roundtrip time is dominated by the kernel-FPGA roundtrip time (291 out of 612 μs), whereas the user-kernel and kernel-user times are relatively modest (24 and 27 μs). The remaining time (270 μs) is spent in the GNU Radio chain. The high latency of the kernel-FPGA roundtrip time is somewhat surprising, given that the effective measured rate of the USB with the USRP is 32 MB/s. Focusing on the latencies between 4KB and 512B, the difference is only a factor of two, suggesting that the setup cost for transfers contributes significantly to the delay. The kernel-FPGA time also includes the time it takes for the data to pass through the USRP USB FX2 controller buffers, and to be copied into the FPGA for parsing. The time taken for the data to pass through the USRP USB FX2 controller buffers and copied into the FPGA for parsing also contributes to the kernel-FPGA RTT. The standard deviations and the min/max values show that the user-kernel and kernel-FPGA times are not highly variable, therefore contributing only a small amount of jitter. On the other hand, the kernel-user times are extremely variable, resulting in a high standard deviation for the GNU Radio ping delays. This is clearly the result of process scheduling.

7.2.2 Implications of SDR Latency on Cognitive Protocol Implementations

The delays shown in Section 7.2.1 have strong implications on cognitive protocol development, especially on protocols running at the host. Although the host CPU is easy to program, the significant delay and jitter shown between the

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

radio hardware and host CPU will impact a host-based cognitive radio network protocol's ability to react quickly to the spectrum, and for the MAC layer to precisely control packet timing, or implement small, precise interframe spacings. We conclude that *time-critical radio or MAC functions should not be placed on the host CPU*. On the other hand, processing close to the radio hardware on the FPGA has the opposite properties, making it attractive for implementing delay-sensitive functions and adapting quickly to the spectrum. Unfortunately, code running on the radio hardware is often closed-source and much harder to change because it is often hardware-specific and requires a more complex development environment. Therefore, we conclude that *in order to be widely applicable, the control of flexible MAC implementations and cognitive techniques should reside on the host*.

When distributing functionality between the host and radio hardware, three key properties of the SDR will be affected: network performance, flexibility in protocol implementations, and reprogrammability. Unfortunately, as discussed earlier, these properties are in conflict with each other and achieving the highest level for each is not possible. In this chapter, we present a split-functionality architecture that implements part of key protocol functions on the radio hardware for performance, and an additional part on the host that provides full control. As we will show, this allows us to simultaneously score very high on all four metrics, and it also allows developers and users to make tradeoffs across the metrics. Even though developers will always have to make tradeoffs, the negatives associated with specific design choices are significantly reduced in our design. Note that this does not imply that our design can support any arbitrary or even all existing MAC designs and cognitive network techniques. However, we believe that it is capable of supporting most of the critical features of modern designs that can be quickly adapted for cognitive techniques using a control channel we introduce. Therefore, we must first identify core functions that the design must support high performance and flexible implementations of, from which modern MAC designs and cognitive techniques can be built.

7.3 Core Cognitive Radio and MAC Functions

An ideal platform for cognitive radio network development, with a focus on a highly adaptable MAC layer, should support well-known MAC protocols, novel designs, and various cognitive techniques. A study of current wireless protocols including WiFi (both Distributed and Point Coordination Function), Zigbee, Bluetooth, and various research protocols shows that they are based on a common, core set of techniques such as contention-based access (CSMA), TDMA, CDMA, and polling. In this section, we identify the subset of functions that a platform must implement efficiently in order to support a wide range of MAC protocols and cognitive radio techniques. In further sections, we focus on

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

splitting these core functions in the architecture in such a way that we achieve high performance of each, while maintaining flexibility for development and fine-grained control over the functions to adapt the radio to the spectrum for cognitive techniques.

- **Precise Scheduling in Time**: TDMA-based protocols require precise scheduling to ensure that transmissions occur during time slots. Imprecise timing can be tolerated by using long guard periods; however, this degrades performance. Surprisingly, modern contention-based protocols also require precise scheduling to implement interframe spacing (i.e., DIFS, SIFS, PIFS), contention windows, back-off periods, and so on.
- **Spectrum Sensing/Carrier Sense**: Contention-based and cognitive radio network protocols often use spectrum sensing and carrier sense to detect other transmissions and available spectrum. Carrier sense may use simple power detection (e.g., using signal strength) or may use actual bit decoding. Network interfaces need to transmit shortly after the channel is detected to be idle. Additional delay increases both the frequency of collision and also the minimum packet size required by the network.
- **Backoff**: When a transmission fails in a contention-based protocol, a backoff mechanism is used to reschedule the transmission under the assumption that the loss was caused by a collision. Backoff is related to precise scheduling, but focuses more closely on fast-rescheduling of a transmission without the full packet transmission process (e.g., modulation).
- **Fast Packet Recognition**: Many MAC performance optimizations could use the ability to quickly detect an incoming packet and identify that it is relevant to the local node in a timely and accurate manner. For example, detecting and identifying an incoming packet before the demodulation procedure can reduce resource use on the processing units and on the bus.
- **Dependent Packets**: Dependent packets are explicit responses to received packets. A typical example is control packets that are associated with data packets, for example, for error control (e.g., ACKs) or for improved channel access (e.g., RTS/CTS). Network interfaces need to generate these packets quickly and transmit them with precise time scheduling relative to the previous packet.
- **Fine-grained Radio Control**: Cognitive radio networks need to adapt to the spectrum quickly, therefore the radio should also be able to switch and adapt all layers on the fly in a timely manner. Frequency-hopping spread spectrum protocols such as Bluetooth and the recently proposed MAXchop algorithm[9] require fine-grained radio control to rapidly change channels according to a pseudorandom sequence. Recent designs[1] for minimizing interference require the ability to control transmission power on a per-packet basis.

- **Access to physical layer information**: Many MAC protocol optimizations and cognitive network techniques could benefit from access to radio-level packet information. Examples include using a received signal strength indicator (RSSI) to improve access point handoff decisions or to locate unused spectrum, and using information on the confidence of each decoded bit to implement partial packet recovery.[6]

It is difficult to argue that this (or any) list of core functions is the correct one and is complete, but we believe that it is sufficient to implement a broad range of interesting MAC protocols and cognitive radio techniques. To provide some degree of confidence in this statement, we describe our implementation of an 802.11-like CSMA protocol and a Bluetooth-like TDMA protocol using our framework in Section 7.6, as well as a cognitive technique that switches between the two layers. As such, this is a reasonable first "toolbox" that protocol developers can extend over time.

7.4 Split Functionality Architecture

Having derived a set of core functions in Section 7.3, we can now determine the types of delay that can affect the performance of each function and discuss how to overcome them. For example, most cognitive radio network protocols need spectrum sensing and need to react quickly to the spectrum; however, the delays inherent in a host-based implementation in the given SDR architecture would make these functions inefficient or ineffective. We first introduce limitations that prevent high-performance implementations of the core functions, and then discuss how to overcome these limitations.

7.4.1 SDR Architectural Limitations

Enabling high-performance implementations of the core functions from Section 7.3 is prevented by three major factors in SDR architecture:

- **Bus delay:** A constant delay introduced by bus transmission is relatively easy to accommodate in supporting *precision scheduling*. However, large delays impact *spectrum sensing and carrier sense*, *dependent packets*, and *fast packet recognition*, as they require information, which is significantly delayed, to perform some task.
- **Queuing delay:** Although queueing delay can be smaller than the bus-transmission delay, it increases the amount of jitter in the system, which makes precision scheduling difficult, if not impossible, at the microsecond level (common in current protocols). In related work,[11] it is shown that this compression can be so significant in the given architecture that spacing

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

transmissions by less than 1 ms cannot be achieved reliably using host-CPU based scheduling.

- **Stream-based architecture of SDRs**: The frontend operates on streams of samples that can make *fine-grained radio control* and *access to physical layer information* from the host ineffective. The reason is that it adds complexity to the interaction between a MAC layer executing on a host CPU (or NIC CPU) and the radio frontend, because it is difficult to associate control information or radio information with particular groups of samples (e.g., those belonging to a packet). This problem consists of two components: (1) how to propagate information within the software environment that performs physical and MAC layer processing; and (2) how to propagate the information between the host and the frontend, across the bus and SDR hardware. This first issue is being addressed in the GNU Radio design with the introduction of m-blocks,[2] which is briefly discussed in Section 7.7, but we must address the second issue.

7.4.2 Overcoming the Limitations

We now present an architecture that overcomes the above limitations. The goal is to allow as much of the protocol to execute on the host as possible to achieve the flexibility and ease of development goals, both of which are important to a wireless platform for protocol development, as identified in Section 7.2. However, we must ensure that the high latency and jitter between the host and radio frontend does not result in poor performance and limited control, the other two criteria in Section 7.2. This is done by introducing two architectural features, *per-block meta-data* and a *control channel*, shown in Figure 7.2. The novelty is not in the two new architectural features, but in how we use them to implement the core MAC functions (Section 7.3) in such a way that we maintain flexibility while increasing performance (Section 7.5). We first discuss both features in more detail.

Per-block meta-data: Enabling the association of information with a packet is crucial to the support of nearly all of the core requirements in Section 7.3. Each packet is modulated into blocks of samples, for which we introduce per-block meta-data. The meta-data stored in the header includes a timestamp (inbound and outbound), a channel flag (data/control), a payload length, and single-bit flags to mark events such as overrun, underrun, or to request specific functions that we implement on the radio hardware. We limit the scope of the meta-data to the minimum needed to support the core requirements, thus minimizing the overhead on the bus.

Control channel: The *control channel* allows us to implement a rich API between the host and radio hardware and allows for less frequent information to

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

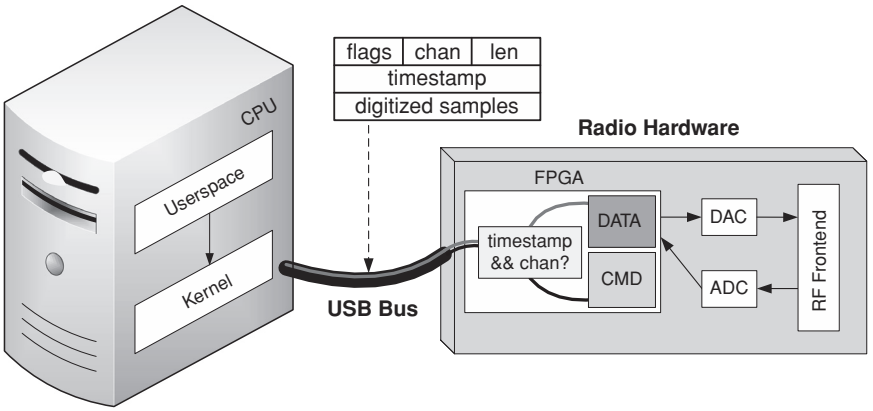

Figure 7.2. Split SDR architecture.

be passed. It consists of control blocks that are interleaved with the data blocks over the same bus. Control blocks carry the same meta-data header as data blocks but have the channel field in the header set to *CONTROL*. The control block payload contains one or more command subblocks. Each subblock specifies the command type, the length of the subblock, and information relevant to the specific command (e.g., a register number). Examples of commands include reading or writing configuration registers on the SDR device, changing the carrier frequency, and setting the signal sampling rate.

With these two features, we can effectively partition the core functions into a part that runs on the radio hardware close to the radio frontend, and a control part that runs on the host. Of course, meta-data and control channels are used in many contexts. The contribution lies in how we use them to partition the core functions, which is the focus of the next section.

7.5 Evaluating the Split-Functionality Approach for Cognitive Radio Networks

We now examine how the split-functionality approach can be used to implement the core functions described in Section 7.3, and just as importantly, how the split-functionality architecture can enable protocols that can react more quickly to the spectrum without sacrificing flexibility. We only present a subset of the functions that are crucial in supporting cognitive network protocols, which illustrate the split-functionality approach, and refer the reader for the details of the remaining functions in related work.[11] We focus our discussion on the GNU Radio and USRP platform.

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

7.5.1 Spectrum Sensing and Carrier Sense

The ability of a cognitive radio to react to the current state of the spectrum is extremely important to the performance of the radio and the effectiveness of the cognitive techniques. Given that cognitive techniques require information about the current state of the network to adapt, if this information is stale, the network will adapt inappropriately. The quicker the radio can adapt, the greater the performance will be. A perfect example of such a technique that requires physical layer information in a timely manner to react properly is carrier sense. The performance of carrier sense is crucial to CSMA protocols: The longer it takes to transmit a packet after the channel goes idle, the greater the chance of collision. Measuring the reactiveness of carrier sense is a raw measurement of the reactiveness of the radio. This timing, which we will refer to as *reactiveness*, is shown in Figure 7.3. Reactiveness is crucial to cognitive radio networks: It is the time it takes for the radio to adapt to a change in the spectrum. We therefore present the split-functionality design of carrier sense to demonstrate how we can achieve greater performance of the core function, and how the split-functionality design increases the reactiveness of the radio.

7.5.1.1 Carrier Sense Design and Evaluation

To significantly increase the reactiveness of the radio to the spectrum, and therefore the performance of carrier sense, we must avoid the associated delays by placing the decision at the radio hardware. However, the decision process should be controlled by software running on the host CPU to maintain flexibility. The first assumption we can make is that when a host wishes to perform carrier sense, it can modulate a packet and pass the computed samples to the radio hardware to wait for the carrier to be idle. The per-block meta-data for the transmission has a single bit flag set to indicate that the block should be held until there is no carrier using a locally computed RSSI value. The host can

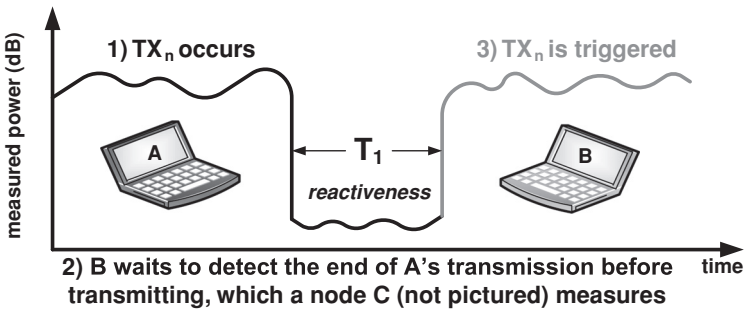

Figure 7.3. Measuring the reactiveness.

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

control the carrier sense threshold via the control channel. We use an RSSI value recorded in the radio hardware to implement a simple RSSI threshold carrier sense mechanism. Therefore, we split the carrier sense implementation in our split-functionality design by placing the carrier sense triggering mechanism on the radio hardware, and full control over the RSSI threshold and carrier sense algorithm on the host. As our evaluation will show, this allows us to achieve greater performance (a smaller reaction time), without sacrificing flexibility.

We now present an evaluation of this carrier sense design in comparison to performing carrier sense at the host. This compares the reactiveness of the radio if the core functions were implemented solely at the host, to the reactiveness of using a split-functionality approach. In the host implementation, the host estimates the received signal strength from the incoming sample stream and uses thresholds to control outgoing transmissions. We use the evaluation setup illustrated in Figure 7.3, where a USRP (node C) monitors two node's transmissions by measuring the magnitude of received complex samples. At 8 megasamples per second, the monitoring node (C) achieves a precision of 125 nanoseconds for measuring the reactiveness of the radios. The two contending nodes (A and B) exchange the channel using carrier sense 100 times, and we measure the spacing between each transmission as the reactiveness, as illustrated in Figure 7.3. The first contending node, A, finishes transmission TX_n, and B takes T_1 time to detect the channel as idle and begin transmission TX_{n+1}. T_1 represents the reactiveness.

As shown in Figure 7.4, taking the average gap observed across 100 exchanges, the results were 1.5 μs and 1.98 ms for the split-functionality and host implementations, respectively. The host-based latency could be reduced closer to 1 ms, or on the order of tens of microseconds, by moving the functionality to the USRP device driver, or the kernel, respectively. In our evaluation, the times were recorded at an application-level block in GNU Radio where a MAC protocol would reside. These measurements illustrate our design's ability

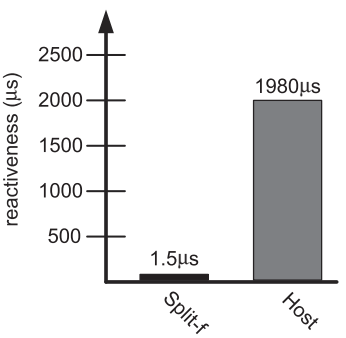

Figure 7.4. Comparing achieved reactiveness.

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

to reduce the carrier sense blind spot by *three orders of magnitude* while maintaining host control on a per-packet basis. This can significantly increase the capacity in the channel by reducing the time it takes to detect it is idle. The host can even control the threshold on a per-packet basis by placing a control packet with a new threshold on the bus before the data packet.

7.5.2 Fast Packet Recognition

Cognitive radio network protocols not only need to react quickly to changes in the spectrum, but also to incoming packets. For example, cognitive radios may exchange packets to inform each other of protocol parameters, or even of complete switches of a layer (e.g., changing from TDMA to CSMA at the MAC layer). Therefore, the radio must be able to identify incoming packets to the node in a timely manner. Additionally, traditional software-defined radios in the receive state will stream captured samples at some decimated rate between the radio hardware and the host. For many MAC protocols, such as CSMA-style designs, the radio cannot determine when packets for the attached node will arrive. As a result, the radio must remain in the receiving state. The downside to this is that the demodulation process uses significant memory and processor resources despite the fact that incoming packets destined for the radio are infrequent. As such radios become more ubiquitous and common for implementation, resource usage will become increasingly important, especially for energy-constrained devices such as the battery-powered Kansas University Agile Radio.[8]

One simple solution would be to send samples when the RSSI is above some threshold. However, this does not filter out transmissions destined to other hosts and external signals. A better solution would be to have the radio hardware look for the packet preamble and the destination address, then transfer a maximum packet size worth of samples to the host after any match. As we also describe in related work,[11] the ability to identify packets and process them partially on the SDR hardware is also critical to supporting low-latency MAC interactions (e.g., packet/ACK exchanges or RTS/CTS) in a high-latency architecture.

7.5.2.1 Fast Packet Recognition Design

Our goal is to accurately detect packets at the radio hardware without demodulating the signal (to keep flexibility). To achieve this goal, we perform signal detection. The most relevant work in signal detection comes from the area of radar and sonar system design. From this area, we borrow a well-known technique, called a *matched filter*, to detect incoming packets at the radio hardware without performing demodulation.

Matched filter: A matched filter is the optimal linear filter that maximizes the output signal-to-noise ratio for use in correlating a known signal to the

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

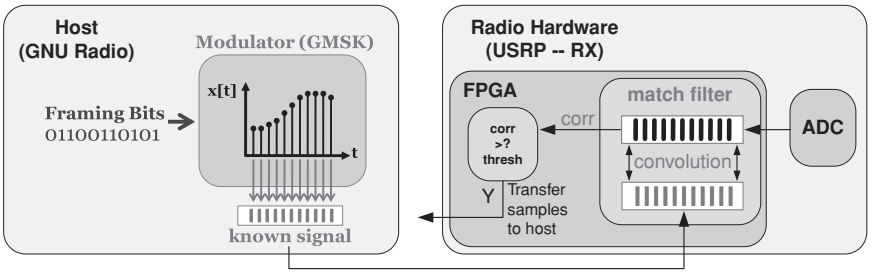

Figure 7.5. Matched filter and dependent packet design.

unknown received signal. For use in packet detection, the known signal would be the time-reversed complex conjugate of the modulated framing bits. These known signal's samples, which are referred to as coefficients, are stored in the matched filter's memory bank (Figure 7.5). The received sample stream is convolved with the coefficients. The result can be treated as a correlation score between the unknown and known signals. The correlation score is then compared with a threshold to trigger the transfer of samples to the host. The matched filter is flexible to different modulation schemes (e.g., GMSK, PSK, QAM) but requires a Fast Fourier transform for OFDM, given that the symbols are in the frequency domain. This would require an FFT implementation on the radio hardware.

To also detect that the frame is destined to the particular host, two different methods that have mathematically different properties can be used. *Single Stage*: Use a frame format where the destination address is the first field after the framing bits, and use this complete modulated sequence as the matched filter coefficients. *Dual Stages*: Detect the framing bits first, then change the coefficients to the modulated destination address. Our implementation uses the single-stage approach for simplification. However, a dual stage is more appropriate for monitoring multiple addresses such as a local address and a broadcast address.

7.5.2.2 Fast Packet Recognition Evaluation

We evaluate the effectiveness of the matched filter at detecting incoming sequences using simulations where we can control the noise level. Results are presented from over-the-air experiments with the presence of interference, multipath, and fading in related work.[11]

To evaluate the effectiveness of the matched filter with varying signal quality, we first run experiments with controlled signal-to-noise ratios (SNR) using the GNU Radio software. We introduce additive white Gaussian noise (AWGN) to control the SNR in terms of dB: $SNR(dB) = 10 * log_{10} * \frac{Power_{signal}}{Power_{noise}}$. To introduce the noise, we compute the signal power: $Power_{signal} = |Signal_{ampl}|^2$,

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

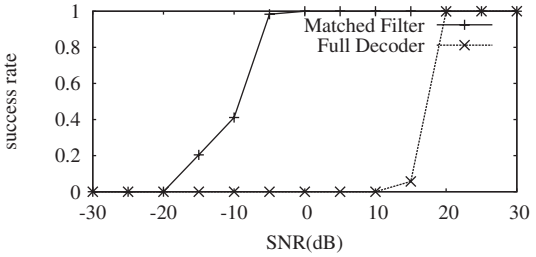

Figure 7.6. Success rate of the matched filter.

and then the noise power: $Power_{noise} = \frac{Power_{signal}}{SNR}$, based on the specified *snr*: $SNR = 10^{(snr/10)}$. For evaluation, 1,000 frames of 1,500 bytes are encoded using the Gaussian minimum-shift keying (GMSK) modulation scheme. These frames are used as the ground truth and mixed with the noise. We require that the matched filter detect the framing bits *and* that the transmission is destined for the attached host using the single-stage scheme (Section 7.5.2). The success rate is defined as the number of detected frames over the total number of frames in the dataset (1,000). For comparison, we also include the success rate of the full GMSK decoder. At a high noise level, even the full decoder will fail at detecting the frames. The success rate, as a function of the SNR, is shown in Figure 7.6. The results show that the matched filter can detect the frames at a much higher success rate than the decoder can, even at low SNR levels where the noise power is greater than the signal power.

Given these results, and further real-world results presented in related work,[11] we conclude that using the matched filter for detecting relevant packets is accurate enough that the host will never miss an actual frame due to the filter. In fact, the filter triggering samples to the host can be seen from a different perspective as providing further confidence to the host that there is actually a frame within the sample stream. The host could then perform additional processing in an attempt to decode the frame successfully.

7.5.3 Access to Physical Layer Information and Fine-Grained Radio Control

The underlying radio hardware in an SDR platform has many controls that are not configured by the transmitted sample stream (e.g., transmission frequency and power), and can make many observations that are not easily derived from the input sample stream (e.g., RSSI). We use our control channel between the SDR hardware and host to expose these controls and physical layer information to the MAC protocol implementation. Many existing network interfaces use similar

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

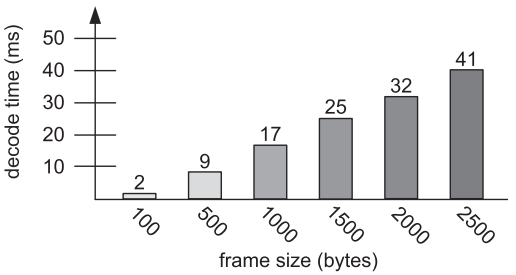

Figure 7.7. Decode times for various frame sizes.

designs for setting the transmission channel and obtaining RSSI measurements. One key difference is that our interface operates on blocks of samples instead of packets.

7.5.3.1 Physical Layer Information

Access to physical layer information at all other layers in the processing chain is important for supporting common cross-layer optimizations and extremely crucial to cognitive radio network protocols to adapt to the spectrum. This can be seen through recent work where per-bit confidence levels are used to perform partial packet recovery.[6] We enable this functionality in our architecture through the control channel and per-block meta-data. In our design, information from the SDR can be sent to the host using either the control channel or per-block meta-data. We use this mechanism to report RSSI to the host. Note that the host could calculate RSSI using the raw samples, but an RSSI value that takes into account the gain or attenuation in the RF stages is only available at the radio hardware. The control protocol is easily modified to support reporting additional properties; however, developers must reprogram the FPGA to report the desired values.

7.5.3.2 Radio Control

We also implement a set of radio hardware control messages on the control channel that can be synchronized with packet transmissions using the timestamp. For example, by placing a control block with a timestamp T before a data packet on the bus, which uses a *NOW* timestamp, the radio will be reconfigured at time T and the data packet will be transmitted immediately after the reconfiguration. This can be used to implement common techniques such as rapid frequency hopping or to reconfigure parts of the core functions that reside on the radio hardware. Unfortunately on the USRP, the daughterboards are tuned directly from the FX2 USB controller using the I^2C bus, which has no connection to the

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

FPGA. Therefore, we cannot issue daughterboard commands from the FPGA using the control channel and hardware clock to implement rapid frequency hopping. The USRP2 tunes the daughterboards directly from the FPGA. Therefore, if our design was implemented on the USRP2, unavailable at the time, rapid frequency hopping could be achieved.

7.6 MAC-Layer Evaluation

We now provide end-to-end results for a Bluetooth-like TDMA protocol and 802.11-like CSMA protocol. The protocols use the *split-functionality* design described in Section 7.5, and we compare their performance with that of full host-based implementations. This demonstrates the increased performance possible at the MAC-layer of the cognitive radio network. In the next section, we describe a design that enables a cognitive radio network protocol that switches between the MAC layers on the fly using information from the radio hardware (Section 7.6.3).

7.6.1 Bluetooth-Like TDMA Protocol

To illustrate the effectiveness of the overall system design, we implement a tightly timed Bluetooth-like TDMA protocol. Like Bluetooth, the network (piconet) consists of a master and a maximum of seven slaves. The slaves communicate with the master in a round-robin fashion within a slot time of 625 μs. Unlike Bluetooth, our protocol fixes its frequency instead of hopping (a limitation of the USRP discussed in Section 7.5.3), uses slightly simpler synchronization (bypasses *pairing*), and we also vary the slot guard time for evaluation.

Each slave in the network synchronizes with the start of a round by listening for the master's beacon, and calculates the start of transmission as the logical synchronization time T. The beacon frame also carries the total number of registered slaves (N) and the guard time (T_g). The slave can then compute the total round time, which must account for the master: $T_r = N + 1 * (T_s + T_g)$, where T_s is the slot time (625 μs). The start of round k is computed as: $T_k = T + T_r * k$. We remind the reader that this is a logical time kept at each node, taken from the beacon frame that is a global reference point. Finally, each slave's slot offset is computed from its node ID (n), $\delta_n = n * (T_s + T_g)$, which is then used to compute the local start time of slave n's slot in round k: $T_{n(k)} = R_k + \delta_n$.

7.6.1.1 TDMA Results

We use two metrics in our evaluation: ability to maintain tight synchronization and overall throughput. The synchronization error at the master is 15 ns, computed by measuring the actual spacing of 1,000 beacons using a monitoring node (discussed in Section 7.5.1). This illustrates the tight timing of the master's

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

beacon transmissions. To measure the synchronization error at the slaves, we record the calculated timestamps of 1,000 beacons at 4 slaves. Each timestamp should be exactly T_r apart from the next. The absolute error in spacing represents shifts in the slave's calculation of the start of the round. We find the maximum error of the 1,000 beacons at all 4 slaves to be 312 ns, with an average of 140 ns. This answers the question of our platform's ability to obtain tight synchronization at both transmitters (master) and receivers (slaves).

We compare a split-functionality implementation to a host implementation, which differ in their guard times. A guard time of 1 μs is used for the split-functionality implementation, which is nearly three times the maximum error. We use our roundtrip host and radio hardware delay measurements from Section 7.2.1, which accounts for both transmissions and reception timing variability, to estimate the host guard time needed. A guard time of 9 ms would be needed to account for the maximum error; however, this delay occurs rarely and we, therefore, present results using a guard time of 3 ms (approximately $3 * \text{sdev}$) and a more realistic guard time of 6 ms based on our recorded delay distribution.

We perform 100 KB file transfers, varying the number of registered slaves and presenting averaged results across 100 transfers in Figure 7.8. The *split-functionality* implementation is able to achieve an average of four times the throughput of the host-based implementation. While we had only been able to answer the question of obtaining synchronization, we find that throughout the full transfers, no slave drifts into another slot period using only the initial beacon for synchronization, illustrating the ability to *maintain* tight synchronization. These results are promising for the development of TDMA protocols on the platform.

7.6.2 802.11-Like CSMA Protocol

We implemented two 802.11-like CSMA MAC protocols, one fully on the host CPU and one using our *split-functionality* optimizations including on-board carrier sense (Section 7.5.1), dependent packet ACK generation, and backoff

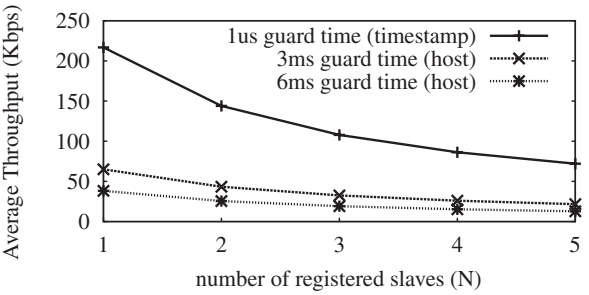

Figure 7.8. TDMA throughput comparison results.

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

(both found in related work[11]). The MAC implements 802.11's clear channel assessment (CCA), exponential backoff, and ACKing. Our protocol does not implement SIFS and DIFS periods; this work is in progress. For space reasons, we focus our description on how the 802.11-like protocol uses our architecture.

The host-based implementation places all functionality on the host CPU, including carrier sense, ACK generation, and the backoff. The optimized implementation uses the matched filter and SNR monitoring for ACK generation, and performs carrier sense and backoff on the radio hardware. We configure the USRPs for a target rate of 0.5 Mbps, and run 100 1 MB file transfers for each implementation using a center frequency of 2.485 GHz in an attempt to avoid 802.11 interference. This allows us to present results that highlight the differences in the implementation without the effect of uncontrolled interference. We also vary the number of nodes in the network, where each pair of nodes performs a transfer.

The results for the two implementations are shown in Figure 7.9. We see significant performance increases from the use of the split-functionality implementation. This nearly doubles the throughput on average, likely due to the time saved in decoding to generate the ACK, and the delays associated with carrier sense and backoff. We note that the matched filter detected every framing sequence, and the fast-ACK generation technique only failed two times over the total number of runs. To recover from these failures, we implemented a feedback mechanism on the host that checks the SNR monitoring technique's decision and retransmits. This is needed because we did not use a higher-layer recover mechanism like TCP.

7.6.3 Supporting Cognitive Switching of the MAC Layer

As discussed throughout this chapter, a cognitive radio network monitors current network conditions and adapts at all layers to achieve the greatest level

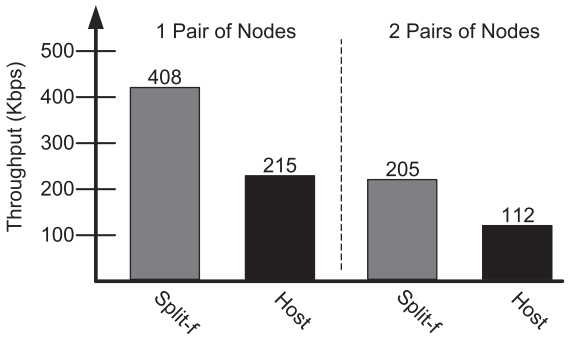

Figure 7.9. 802.11-like CSMA protocol results.

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

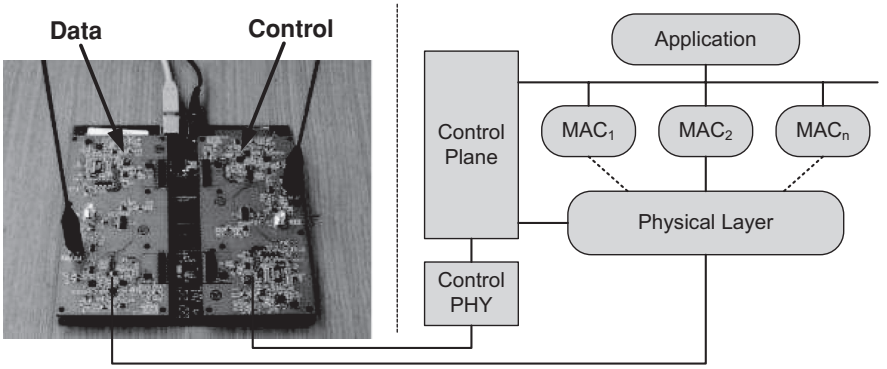

Figure 7.10. Design of a cognitive architecture for MAC layer switching.

of performance, given the current spectrum. For example, the radio can use spectrum-sensing information to find a less congested channel, or use noise/loss information from the PHY and MAC layers to change the protocols running to achieve a higher throughput. Under low loss rates and low congestion, the radio could use the CSMA protocol discussed in Section 7.6.2. Under high loss and congestion, the radio could use the TDMA protocol discussed in Section 7.6.1, which reduces the overhead involved with accessing the spectrum and reduces the chances of collision. While we have briefly presented a general design for a switching protocol, it is not the contribution of this section. The contribution is a general architecture and design of the components on a host-PHY radio, which allows for the switching of the layers, shown in Figure 7.10, as well as suggestions on how to better support layer switching in a host-PHY architecture. From this design, novel cognitive radio network protocols that govern the switching of the layers can be built. Additionally, it is a layer that can be accessible via a global controller in the future Internet to control the protocols (e.g., the MAC layer) in use by the cognitive radio. Through such a controller, new protocols could be designed and propagated to the radios for use.

As mentioned, the radio uses information from the protocol layers, such as the loss rate, to change the MAC that is operating. This logic cannot reside in the MAC layer because it needs to run independent of this layer. Additionally, we do not want to place it at the PHY layer because it violates the general hierarchy of networking protocols, and we would like the design to be general enough to allow switching of the physical layer. Therefore, we need to construct a new layer that monitors information from both the MAC and PHY layers, shown in Figure 7.10. We refer to this layer as a *control plane*, which has a connection to all possible MAC layers for control and status information. The control plane communicates to the MAC layers, activating the appropriate MAC protocol such that it communicates with the physical layer. There is also a connection from

https://doi.org/10.1017/CBO9780511921117.008 Published online by Cambridge University Press

the control plane to the physical layer for control and status information such that it can adapt based on physical layer information. As it is important that all radios in the network operate using the same MAC layer, we construct a control channel using a second frontend in a different frequency band on the USRP, which uses a separate physical layer at the host. The control channel is used to communicate a switch of the MAC layer, which all other radios will also switch to. All radios in the network use the control channel to agree on a MAC layer, sharing current channel information such as congestion and loss. The radios use a basic consensus mechanism to choose the layer, based on beacons of the current MAC layer sent by each radio on the control channel. Once consensus is reached to change the MAC layer, the control plane uses the control-and-status channel to the MAC layers to disable or enable the appropriate protocols.

Using this general architecture, we are able to change the active MAC layer based on the current state of the network. However, we are not able to change the connections between the MAC layers and the physical layer. To our knowledge, current host-PHY architectures do not directly support switching of the MAC layer; in fact, their general architecture inhibits it. In a host-PHY software-defined radio architecture, such as GNU Radio, it is common that multiple modular processing blocks are connected to create a layer such as the MAC or PHY layers. At runtime, instances of the blocks are created, and the application specifies how the blocks are connected. To our knowledge, there is no host-PHY SDR that allows for the connection of these blocks, and what blocks are instantiated, to be changed at runtime. This prevents a cognitive radio network protocol from making "extreme" changes to the processing, such as removing a series of blocks to completely replace the MAC layer. Therefore, as future work, we propose host-PHY architectures take into consideration a growing need to dynamically change the processing of the radio in an optimal manner, such that all blocks do not need to be instantiated at runtime, which requires additional memory and logic such that all blocks are connected.

7.7 Related Work

We review related work in the area of MAC development. Existing platforms mostly use the extremes of the design space where either the majority of functionality is fixed on the network card (*Traditional NICs*) or performs all processing at the host (*Software-Defined Radios*).

Traditional NICs: Several efforts[3,10,13] have built new MAC protocols on top of existing commercial NICs (e.g., 802.11 cards). Unfortunately, commercial 802.11 cards implement the bulk of the MAC functionality in proprietary microcode on the card, limiting what functions can be changed by researchers. As a result, this approach is not very satisfactory: The range of MAC protocols that can be implemented is limited and performance (e.g., throughput,

capacity) is often poor from the MAC needing to be implemented on the host. For example, past efforts have mostly implemented TDMA-based schemes.

Software-defined Radios: Software-Defined radios (SDRs) provide a compelling architecture for flexible wireless protocol development, considering that most aspects of both the MAC and physical layer are, by design, implemented in software and thus, in principle, easy to modify. However, so far, SDR efforts have focused on implementing the physical layer[16] whereas MAC and higher-layer protocol development has received little attention. Recent work by Schmid et al.[12] examines the impact of increased latency in SDRs using GNU Radio and the USRP. The authors address how the bus latency creates "blind spots" that increase collision rates when carrier sense is performed at the host, and how precomputation of packets is not possible without fully demodulating (at the host), resulting in larger interframe spacing. Our design provides solutions for both of these issues in Sections 7.5.1 and 7.5.2, respectively. Bus delay measurements were also taken by Valentin et al.[15]

A number of groups have developed software radios with architectures that differ from the current GNU Radio and USRP design by including a CPU on the radio hardware (NC-CPU), either as a separate component or as a core on the FPGA. Examples include the Rice University Wireless Open-Access Research Platform (WARP)[17] and USRP2. These designs are more expensive, but they offer additional flexibility for partitioning the MAC. However, there is still a nontrivial delay (compared with traditional radios) owing to physical layer processing and queueing. The NC-CPU is also likely to be slower than the host CPU, increasing the processing delay. Finally, in deployed products based on this architecture, the NC-CPU is likely to be off-limit to users, similar to the current situation with commercial wireless cards. As a result, we expect that our architecture will be useful for this type of platform as well.

7.8 Conclusions

In this chapter, we presented a set of techniques that support the implementation of diverse, high-performance cognitive radio network protocols on software radios. The work is motivated by an increasing diverse and ever-changing wireless spectrum, such that to achieve the greatest level of performance, the radio must adapt at all layers in the wireless networking stack. Software radios offer flexibility, but their architecture, specifically the delay between the host and the radio frontend, has traditionally been a problem for protocols. We introduce a split-functionally approach, which addresses this problem, and show that it enables the implementation of a set of core MAC functions and cognitive functions that can react to the spectrum more quickly for greater performance. An implementation for the USRP and GNU Radio, along with the implementation of an 802.11-like and Bluetooth-like protocol, shows the approach is effective.

Additionally, we presented a basic design to support cognitive switching of the MAC layer on the fly, which can be extended. To our best knowledge, these protocol implementations are the first high-speed, bidirectional MAC implementations for the GNU software radio platform. For future work, we plan to implement a more diverse set of protocols to further evaluate our design and implement the architecture on different SDR platforms to evaluate its generality.

References

[1] A. Akella, G. Judd, S. Seshan, and P. Steenkiste. Self-Management in Chaotic Wireless Deployments. *ACM MobiCom*, pages 185–199, 2005.
[2] BBN Technologies Corperation, GNU Radio Architectural Changes (m-block). http://acert.ir.bbn.com/downloads/adroit/gnuradio-architectural-enhancements-3.pdf
[3] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. C. Sicker, and D. Grunwald. MultiMAC – An Adaptive MAC Framework for Dynamic Radio Networking. *IEEE DySPAN*, 2005.
[4] S. Gollakota and D. Katabi. Zigzag Decoding: Combating Hidden Terminals in Wireless Networks. *ACM SIGCOMM*, 2008. ACM Press.
[5] Gnu radio. http://www.gnu.org/software/gnuradio/
[6] K. Jamieson and H. Balakrishnan. PPR: Partial Packet Recovery for Wireless Networks. *SIGCOMM Comput. Commun. Rev.*, 37(4): 409–420, 2007.
[7] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-Level Network Coding for Wireless Mesh Networks. *ACM SIGCOMM*, 2008. ACM Press.
[8] Kansas university agile radio. https://agileradio.ittc.ku.edu/
[9] A. Mishra, V. Shrivastava, D. Agrawal, S. Banerjee, and S. Ganguly. Distributed Channel Management in Uncoordinated Wireless Environments. *ACM MobiCom*, pages 170–181, 2006.
[10] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMAC – Flexible Wireless Research Platform. *Fourth Workshop on Hot Topics in Networks (HotNets)*, 2005.
[11] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste. Enabling MAC Protocol Implementations on Software-Defined Radios. *NSDI*, 2009.
[12] T. Schmid, O. Sekkat, and M. B. Srivastava. An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios. *WiNTECH'07*, 2007.
[13] A. Sharma, M. Tiwari, and H. Zheng. MadMAC: Building a Reconfigurable Radio Testbed Using Commodity 802.11 Hardware. *IEEE Workshop on Networking Technologies for Software Defined Radio Networks*, Reston, 2006.
[14] The Universal Software Radio Peripheral. http://www.ettus.com/
[15] S. Valentin, H. von Malm, and H. Karl. Evaluating the GNU Software Radio Platform for Wireless Testbeds. *Technical Report TR-RT-06-273*, 2006.
[16] Vanu Software Radio Systems. http://www.vanu.com
[17] Rice University Wireless Open-Access Research Platform (WARP). http://warp.rice.edu