FreeMAC: Framework for Multi-Channel MAC Development on 802.11 Hardware

Ashish Sharma, Elizabeth M. Belding
Department of Computer Science
University of California, Santa Barbara CA 93106
{asharma, ebelding}@cs.ucsb.edu

ABSTRACT

Exponential growth in the number of wireless devices that operate in the limited unlicensed frequency spectrum necessitates the next generation of radio devices to be reconfigurable and sensitive to changes in network conditions and spectrum availability. Most modern wireless devices offer increased software programmability and control over radio communication parameters. Since a large portion of the MAC protocol is implemented in software, with the firmware providing a set of functional primitives, it is possible to design and implement alternate MAC protocols in real testbeds equipped with commodity 802.11 devices. This paper describes FreeMAC, a reconfigurable MAC protocol development framework that enables the design and implementation of a general class of multi-channel MAC protocols on a typical Linux system. FreeMAC provides support for frequent channel switching and fine control over the timing of packet transmissions. We also propose a mechanism to reduce the latency in the scheduling of periodic operations of a software MAC protocol that have strict timing requirements. Results from our six node testbed indicate that using our approach, the scheduling latency of slot transitions in a TDMA-style MAC can be improved by up to an order of magnitude, with minimal overhead. FreeMAC also exports a number of radio configuration parameters as API functions to enable cross layer interactions among wireless networking protocols. As a proof of concept, we implement a simple multi-channel TDMA MAC on our testbed to demonstrate the utility of FreeMAC as a development framework.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]:
COMPUTER-COMMUNICATION NETWORKS Network Architecture and Design Wireless communication

General Terms

Design, Experimentation, Measurement

Keywords

MAC, TDMA, multi channel, wireless networks, medium access

PRESTO'08, August 22, 2008, Seattle, Washington, USA.

1. INTRODUCTION

The explosive growth in the popularity of 802.11 technology in recent years has given WiFi devices a ubiquitous presence around the world. The operation of 802.11 devices in the unlicensed frequency spectrum has been a major factor contributing to the large scale adoption of WiFi technology. Further, the economies of scale have dramatically reduced the cost of hardware and resulted in easy availability of 802.11 devices even in rural areas of the world.

Most 802.11 hardware manufacturers offer increased software programmability and control over several radio communication parameters. The increasing number of wireless devices in the unlicensed spectrum necessitates the next generation of radio devices to be reconfigurable and sensitive to changing network conditions and spectrum availability. In the future, the trend of increasing the programmability of the hardware is expected to grow due to the emergence of cognitive and software defined radio networking.

Several recent efforts have focused on the customization of the default 802.11 MAC to achieve significant performance improvement in specific network scenarios. 2P [11] and WildNet [10] are examples of a custom TDMA-style MAC on 802.11 hardware for long distance WiFi-based rural mesh networks. There are several radio communication parameters, such as transmit power, frequency channel, transmission rate, number of retries, slot duration, backoff intervals, per-packet acknowledgements and reception of erroneous frames, that can be used to improve the performance of different types of wireless networks. Since a large portion of the 802.11 MAC is implemented in software, with the firmware providing a set of functional primitives, it is even possible to design and implement new MAC protocols on commodity 802.11 devices [8, 14, 4].

Despite the promise of 802.11 devices as an affordable platform to build new MAC protocols, the real world implementation of new MAC protocols still faces many challenges. To date, most modifications to the 802.11 MAC focus on achieving specific functionality, making it difficult to adapt an existing solution to implement a different protocol. The shortage of generic wireless MAC protocol development frameworks and the high cost of software defined radios has forced most researchers to rely on simulations for evaluation of new MAC protocols. However, simulations not only fail to accurately capture the characteristics of a wireless medium, but also adopt an indifferent approach to the design and implementation constraints that arise when dealing with a real world system. Factors like the extent of support offered by the operating system, degree of programmability offered by a hardware manufacturer, cost and availability constraints of the suitable hardware often prohibit a MAC layer design from being deployed on a real testbed.

To enable research that has the potential for significant impact in the real world, the networking community needs platforms that

allow researchers to not only use the wide array of radio configuration parameters, but also develop new protocols than can be validated on real testbeds. Recent research has shown that by increasing cross layer interaction between MAC and upper layer networking protocols, significant improvement in the system performance can be achieved [1, 15]. Thus, exporting the functionality available at the MAC layer to the upper layers of the networking stack, by means of API functions, can pave the way for better protocol designs. Further, multi-channel MAC protocols can result in improved spatial reuse of the available channels in a network.

In this paper, we describe the design and implementation details of FreeMAC – a framework for the development of *multi-channel* MAC protocols, with strict timing requirements. We use the OpenHAL [9] port of MadWiFi [6] - an open source driver for Atheros chipset-based commodity 802.11 wireless devices. FreeMAC leverages the methodology described in the SoftMAC [8] development platform and our previous work on MadMAC [14]. However, unlike previous approaches that either focus on specific TDMA-style MAC development [11, 10] or provide MAC development frameworks for single-channel MAC protocols [8, 4, 14], we address a general class of MAC protocols that may require strict control over the timing of MAC protocol functions and/or the ability of the radio device to switch between different frequency channels. In summary, this paper makes the following key contributions:

1. We develop a framework for the implementation of *multi-channel* MAC protocols on commodity 802.11 hardware.
2. We propose a new implementation mechanism for software TDMA-style MAC protocols that improves the unpredictable latency in the scheduling of time-sensitive TDMA functions. Our approach reduces the system response time towards periodic time-critical functions from up to several hundred milliseconds to a sub-millisecond predictable delay, with minimal additional system overhead.
3. We provide a set of API functions that export a number of radio configuration parameters to the MAC protocol developer and other network applications.

The rest of the paper is organized as follows. Section 2 gives an overview of the FreeMAC framework and the supported feature set. In Section 2.1, we highlight the challenges in the implementation of any multi-channel MAC in software on a typical Linux system. Section 2.2 describes the implementation details of FreeMAC platform. In Section 3, we implement a simple multi-channel TDMA style MAC on a testbed of six nodes and present some experimental results. Section 4 presents the related work in the field. We conclude by summarizing our contributions and ideas for future work in Section 5.

2. FREEMAC DESIGN

FreeMAC is a framework for the design and implementation of wireless MAC protocols on a Linux system using commodity 802.11 devices. It allows tight control over several radio parameters, enabling the development of a general class of multi-channel MAC protocols that require time-critical scheduling of periodic MAC functions. The FreeMAC system design is based on three main principles. First, the system should focus on providing mechanisms to implement a wide variety of MAC protocols, rather than specific customizations. Second, all mechanisms proposed should be implementable on a typical Linux system with commodity WiFi cards, without the need for any additional hardware. Third, the system should refrain from relying on any proprietary modifications to the device firmware.

The FreeMAC development platform provides fine control over the timing of packet transmissions by exporting API functions to the MAC layer designer that manipulate the hardware queue size and control the contention backoff periods. It achieves precise control over the scheduling of time-critical MAC functions by programming service routines of periodic beacon interrupts instead of using software timers in the kernel. FreeMAC provides support for frequent channel switching by eliminating any scanning or neighbor discovery operations associated with the IEEE 802.11 protocol, as the ability to do so lies at the core of any multi-channel MAC protocol that requires a single node to communicate with several other nodes operating on different channels [16, 2].

The FreeMAC development framework provides fine control over a number of radio communication parameters, which can be used to improve cross layer interaction between networking protocols. FreeMAC supports the following feature set:

- Flexible frame formats
- Disabling of per-packet ACKs
- Disabling of virtual carrier-sense
- Support for rapid channel switching
- Disabling/control of random backoff intervals
- Predictable scheduling of time-critical functions
- Fine control over the time of packet transmission
- Device driver and hardware queue size manipulation
- Enable/disable per-packet successful transmit interrupt
- Per-packet control of the number of retransmissions, transmit power and rate

As a proof of concept, we use the FreeMAC framework to implement a simple prototype multi-channel TDMA based MAC on a testbed of six nodes. Multi-channel MAC protocols aim to improve the capacity of a wireless network by time-multiplexing the operation of nodes on orthogonal channels. While a full fledged multi-channel TDMA MAC protocol has to deal with a whole array of issues, such as distributed versus centralized scheduling, time synchronization, clock drifts, dissemination of schedules, static versus dynamic scheduling and admission control, we focus our attention on enabling the primitives for the deployment of such a MAC protocol on existing systems.

2.1 Challenges

In the absence of inherent support from the device hardware, there are several challenges in implementing a TDMA-based MAC protocol in software. In this section we discuss the challenges in implementing a multi-channel TDMA MAC protocol on a Linux system, using commodity 802.11 hardware. Section 2.2 describes, in detail, our solutions to the challenges discussed below.

A. Insufficient support for time scheduling

Essential to the implementation of a TDMA style MAC protocol in software is the ability to precisely control the timing of state transitions at slot boundaries. In the case of multi-channel MAC protocols, where a node is required to switch to a different channel to communicate with a neighbor, the timing requirements become even more stringent. If a node fails to switch to the destined channel at the scheduled time, it will be unable to communicate with the neighboring nodes. This can lead to network partitions. The slot

duration of a TDMA style MAC implemented in software is typically of the order of tens of milliseconds (20-60 ms) [10, 14]. Patra *et al.* observe that in their WiLDNet [10] deployments, the UDP throughput levels off beyond a TDMA slot size of 20 ms. On the other hand, for a constant send window size, TCP throughput tends to reduce at higher slot sizes due to an increased bandwidth-delay product of the long distance link. Thus an inaccuracy of even a few milliseconds can have an observable impact on the performance of the TDMA MAC.

The Linux kernel, by default, does not allow fine control over the scheduling of time critical functions, required for TDMA based MAC schemes. Timers are handled in the kernel using *deferrable* functions that may be executed, depending on system load, a long time after the expiration of the timer. The kernel timers guarantee best effort scheduling of the timer tasks, ensuring only that the task will be scheduled for execution either at the moment of the timer expiration, or after a variable delay of up to tens or even hundreds of milliseconds [3]. For the above mentioned reasons conventional kernel software timers are unsuitable for applications that require the expiration time to be strictly enforced, such as a TDMA-style MAC.

B. Insufficient control over transmission time

There is a variable queueing delay, both in the device driver and hardware queues, between the time a packet is delivered by the software TDMA scheduler to the device driver and the time the packet is actually transmitted. The Clear Channel Assessment (CCA) and CSMA/CA backoff procedures, preceding a packet transmission in the 802.11 protocol, further add to an unpredictable delay before the packet is finally transmitted. In WiLDNet [10] and Overlay MAC [12] implementations, the TDMA scheduler is implemented as a *Click* [5] module between the network layer and the device driver. Such an approach may lead to an additional delay because of scheduling latency between the *Click* module and the device driver processes in the operating system. By providing TDMA scheduling support in the device driver and controlling the backoff thresholds and queue sizes, transmission timing of a packet can be controlled to a large extent.

C. State Transition Overhead

In a software TDMA style MAC, transitions between a transmit and a receive slot may incur a non-negligible overhead. State transitions at slot boundaries may involve several operations like changing the antenna configuration in multi-antenna systems, beacon transmission or channel switching. In the case of multi-channel MAC protocols, the channel switching delay may result in significant overhead, as discussed in detail in Section 2.2. The system must not only wait for any ongoing transmissions or receptions to finish, but also allow any DMA operations to complete ongoing transfers between the system memory and the wireless device. Depending on the system load, this delay may vary from time to time.

A common approach to minimize the effect of transition overhead and to overcome synchronization errors due to variable clock drift rates is to introduce a grace period (*guard band*) between consecutive time slots. Sharma *et al.* [14] study the effect of different guard time durations on throughput, in a single channel TDMA MAC implementation, and show that for slot durations between 20-60 ms, the observed UDP throughput decreases with increasing guard durations.

D. Time Synchronization

The greatest challenge in implementing a TDMA MAC protocol lies in synchronizing the clocks of nodes in the network. Different MAC protocols define different constraints on the precision levels expected from the clock synchronization technique used in the network. Raman *et al.* use special *marker* packets to achieve synchronous transmissions among nodes. Patra *et al.* synchronize the TDMA slots by adopting a notion of *virtual time* indicated in the first packet of each transmission slot. The Overlay MAC [12] and SSCH [2] protocols argue the case for employing advanced techniques from the sensor network research literature to achieve synchronization among nodes over multiple hops.

Broadly, time synchronization among nodes in a network may be achieved in two ways. In the first approach, an in-system communication protocol, as proposed extensively in the sensor network community, achieves time synchronization in a centralized or distributed fashion [13]. The other alternative approach is to employ an external channel to synchronize the nodes, such as through GPS based clock synchronization. Because our goal is to enable operating system support for a software implementation of a TDMA MAC, a detailed discussion of the various synchronization techniques is beyond the scope of this paper.

2.2 Implementation

In this section, we describe the internal implementation details of the FreeMAC programmable MAC development framework. The degree of programming flexibility of hardware varies from one manufacturer to the other. The FreeMAC platform uses the features of Atheros chipset-based 802.11 devices. In our system, we use the AR5212 chipset-based commodity 802.11 a/b/g hardware. The Atheros chipset provides a high degree of software control over several aspects of the radio device. To comply with wireless spectrum regulations, Atheros restricts all access to the hardware through a software Hardware Abstraction Layer (HAL), which is distributed in the form of a binary file. FreeMAC builds on top of the MadWiFi open source driver, which has the support of a large community of open-source developers. Recently, there have been attempts by the MadWiFi developer community to develop OpenHAL [9] - an open source project that aims to replace the proprietary Atheros HAL, mainly by reverse engineering, to directly access the hardware. FreeMAC is developed using only open source references and contains no proprietary code. We now describe, in detail, how FreeMAC achieves the aforementioned functionality and our approach to address the system challenges described in Section 2.1.

Precise event scheduling: To overcome the limitations of kernel timers in scheduling time-sensitive MAC functions, we devise a method to generate periodic hardware interrupts from the Atheros wireless device. Atheros hardware allows the creation of different types of hardware queues, one of which is a *beacon* queue. A *beacon* queue is initialized with a beacon interval (*bintval*) value that specifies the duration at which a periodic hardware interrupt (HAL_INT_SWBA) is generated. The default handler for this interrupt invokes a beacon generating function in the MadWiFi driver code. FreeMAC exploits this periodic hardware interrupt to execute the time-critical functions of the software TDMA MAC. This is done by substituting the default beacon interrupt handler with a custom handler in the interrupt service routine. Although the periodicity of beacon intervals can be programmed dynamically using FreeMAC API functions, such an approach of using beacon interrupts is most suited for operations that are periodic in nature such as TDMA slot transitions, instead of individually scheduled events.

The tradeoff in executing MAC functions in the *interrupt context* of kernel is that these operations take precedence over other system tasks. Thus, it is important to keep the number of operations in the interrupt handler to a minimum. Since interrupt handlers, by their nature, run concurrently with other system code, it is also essential to implement proper locking mechanisms in the MAC code. This

avoids concurrency related issues due to simultaneous contention for kernel data structures and device hardware.

Improved control over timing of packet transmissions: To provide fine control over the timing of packet transmissions, we reduce the random backoff by setting the CW_{min} and CW_{max} parameters to the minimum value (0 for beacons and 1 for normal traffic). We also disable the random *post-backoff* (backoff specified in the 802.11 protocol that requires a sender to backoff after a successful transmission), using the HAL_TXQ_BACKOFF_DISABLE flag in the transmit queue descriptors. We export API functions in the FreeMAC platform to specify the minimum and maximum CW thresholds dynamically. We reduce the queueing delay in the hardware by setting the queue size to one and buffer packets in the software instead. In Atheros based devices, a channel switch operation involves resetting of the hardware. Since all packets buffered in the hardware queue are flushed during a hardware reset, a queue size of one also prevents multiple packets from getting dropped during a channel switch. However, if a MAC does not require frequent channel switching then a small hardware queue size might incur a penalty in the performance of the software MAC.

Rapid channel switching: Channel switching in 802.11 networks is a rather infrequent operation and nodes typically have a *dwell period* of tens of milliseconds. During this time they perform scanning and neighbor discovery operations under the assumption that, upon a channel switch, a node breaks communication with the present wireless network. In multi-channel networks, the assumption that a node disassociates from the present network during a channel switch is invalid. FreeMAC enables the development of multi-channel MAC protocols that require frequent switching of channels by eliminating any scan or neighbor discovery operations. Although hardware specifications of several manufacturers indicate a channel switching delay in the order of 200 μs [7], we observed that on the AR5212 chipset, the channel switching operation requires a full hardware reset which incurs a delay of approximately 1.2 ms. An additional delay of about 3.2 ms is introduced in the channel switching operation due to other system operations, such as flushing any pending transmission buffers in the hardware queues and waiting for any pending transmit or receive DMA operations to finish. Table 1 shows the results from our testbed running a simple multi-channel TDMA MAC protocol.

FreeMAC provides the MAC designer with the choice to disable per-frame ACKs. Disabling per-packet acknowledgements requires two specific tasks. First, the receiver must be prevented from sending an ACK for a received frame. This can be achieved by marking the transmitted packet as a multicast packet, as described in [8, 4]. The second task is to prevent the sender from waiting to receive an ACK from the receiver. This can be achieved by setting the HAL_TXDESC_NOACK flag, which is set for broadcast and multicast packets, in addition to beacon frames. FreeMAC operates in the *monitor* mode of the MadWiFi driver, which allows several benefits such as transmission of raw packets (bypassing the default 802.11 frame format) and reception of control and data packets, whether or not they pass the checksum. Monitor mode allows easy manipulation of the frame format and disabling of virtual carrier sense (NAV), as described in [4]. We implement the multi-channel TDMA protocol using a custom frame format that extends Ethernet header with sequence numbers. However, FreeMAC can be easily programmed to support any other header format. In addition to the above mentioned features, FreeMAC also exports primitives to control MAC level parameters such as hardware queue size, number of retries, and transmission power and rate, by means of different API functions.

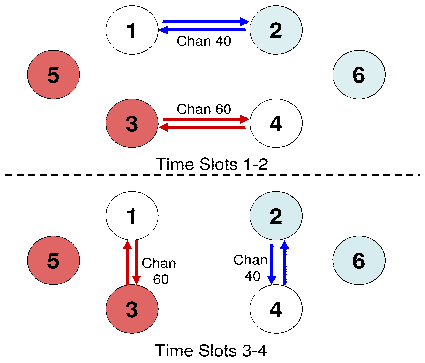

Figure 1: Experimental Setup.

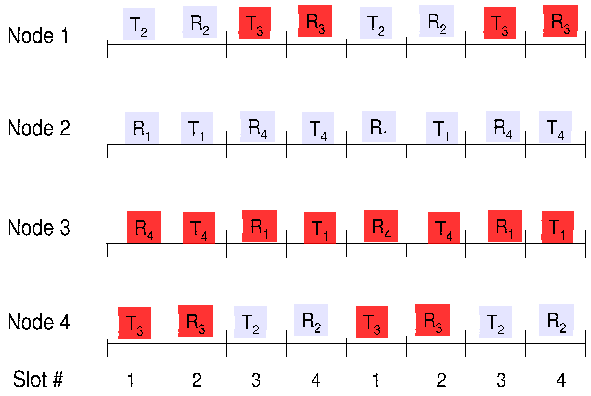

Figure 2: Multi-channel TDMA schedule.

3. MULTI-CHANNEL TDMA MAC EXPERIMENTS

In this section, we implement a simple multi-channel TDMA-style MAC protocol on a testbed of six IBM laptops to demonstrate the utility of FreeMAC as a generic MAC development platform. The goal of our experiments is to verify the correctness of the primitives exported by the FreeMAC platform. We characterize the improvement in scheduling latency of time-slot transitions due to the beacon interrupt scheme and measure the overhead of using such an approach. We also measure the channel switching delay of commodity 802.11 devices and the constraints it imposes on a multi-channel MAC protocol implementation. We now describe the system setup and the TDMA scheduling scheme followed by the results.

3.1 System Setup

FreeMAC is implemented on six IBM laptops, each running *preemptible* Linux kernel version 2.6.15.7 on Ubuntu distribution. Each laptop is equipped with an AR5212 chipset-based LinkSys 802.11 a/b/g PCMCIA card. In our testbed, we implement both single and multi-channel TDMA-style MAC protocols as described below. As shown in Figure 1, nodes 1 and 4 run a multi-channel TDMA protocol in which the nodes alternate between two different channels. Nodes 2 and 3 run a single channel TDMA protocol. Nodes 5 and 6 passively monitor the network activity on the two channels in use. In Figure 1, nodes 3 and 5 operate on channel 40 in the 802.11a 5GHz band, while nodes 2 and 6 operate on channel 60. All the nodes are synchronized using the Ethernet interface of each of these

	Average (ms)	Standard Deviation (μs)
Channel switch delay	4.378	12.061
Hardware reset delay (*ath_hal_reset*)	1.162	4.971
Interrupt Periodicity (*bintval = 25*)		
CPU usage = 1%	25.6	12.407
CPU usage = 99%	25.6	13.460

Table 1: Experimental results from the multi-channel TDMA implementation.

nodes, which provides an out-of-system synchronization channel. We use the synchronization protocol described in [13], which provides an accuracy of 25 μs on our system.

The above setup demonstrates the capability of the FreeMAC framework to implement both single-channel (for nodes 2 and 3) as well as multi-channel (for nodes 1 and 4) TDMA-style MAC protocols. There are four time-slots in our TDMA schedule and two orthogonal channels. Each node transmits in two of these slots and receives on the remaining two. In the case of nodes 1 and 4, which run a multi-channel TDMA protocol, the two transmit slots are assigned different channels such that only one node transmits on a particular channel at any time. The nodes switch channels at the beginning of each transmit slot. In the case of nodes 2 and 3, which run a single channel TDMA protocol, the transmit and receive slots alternate in each cycle. Nodes 5 and 6 are each assigned a separate channel to monitor and verify the correct operation of the proposed system. Figure 2 shows the state of the multi-channel TDMA system over time. T_x denotes a transmission slot where the receiver is node x. Similarly R_y denotes a receive slot, where the transmitter is node y.

We choose a time slot duration of 25 ms with a *guard band* interval of 5 ms preceding each slot (as described in Section 2.1). To demonstrate the ability to build a MAC protocol that uses a different frame format than that of the 802.11 protocol, we use a custom frame format. In the new frame format, we extend the Ethernet header to include a MAC sequence number field. IP packets from the network layer are encapsulated in the custom frame format using *Click* [5] and transmitted in *monitor* mode to allow raw frame transmissions. We disable per-packet ACKs and retries for the transmitted packets. A node buffers all incoming packets from the network layer and only transmits packets in its scheduled transmit slot. We disable random backoff and virtual carrier sensing. The following section describes the experimental results from our testbed.

3.2 Results

A. Timing Accuracy

Our experiments show that the hardware beacon interrupt can be used to schedule periodic time-critical events with a sub-millisecond accuracy. Beacon interrupts are generated by the Atheros hardware using an internal timer that has a frequency of 2^{10} ticks per second. The supplied *bintval* parameter b translates to a periodicity of a milliseconds according to Eqn 1 with a standard deviation of less than $15\mu s$.

$$a = 1.024 * b \quad (1)$$

Figure 3 shows the periodicity at which the custom beacon interrupt handler was called for slot transition functions in a standard Linux machine with a preemptible kernel. It is also worth noting that in our system, the periodicity of interrupt handler execution remained consistent, even during high system load conditions, as shown in Table 1.

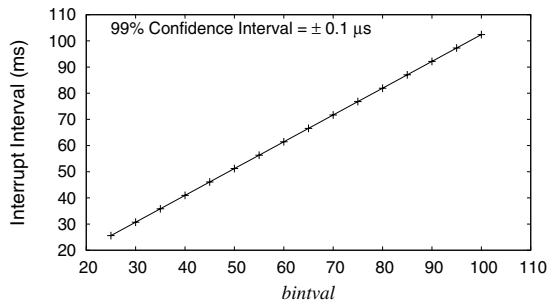

Figure 3: Interrupt periodicity as a function of *bintval*.

B. Channel Switching Delay

Most hardware manufacturers claim that the channel switching delay is of the order of 80-90 microseconds [7]. However, our experiments indicate that for AR5212 chipset based 802.11 a/b/g cards, using the MadWiFi driver, a channel switch results in an average end-to-end delay of 4-5 ms. In addition to the actual hardware *reset*, which takes about 1.2 ms on average, a delay of 3.2 ms occurs to account for any pending DMA operations between the system memory and the hardware to finish up.

Results from our testbed highlight an important limitation in the software implementation of multi-channel MAC protocols that require channel switching every few milliseconds. SSCH [2] suggests channel switching every 10 ms, while MMAC [16] suggests a channel switch every 100 ms. The current draft of the 802.11s standard for wireless mesh networks also mentions a multi-channel operation mode where nodes may rapidly switch channels for data transactions. Each of these protocols rely on the ability to switch channels within a period of 100-200 μs. While the millisecond delay in channel switching may be a limitation of the specific hardware used in our system, the waiting period for pending DMA operations may be a more generic issue. We note that while designing a multi-channel MAC protocol, either this limitation must be taken into account or specific system improvements may need to be designed. We plan to investigate this issue in our future work.

C. Interrupt Overhead

We use beacon interrupts to service time-sensitive MAC operations such as TDMA slot transitioning and initiate channel switching. Since we only modify the interrupt handler of beacon interrupts that occur in a conventional wireless access point, the only additional overhead this technique imposes on the system is due to the increase in the frequency at which such interrupts occur. The normal Linux kernel handles 1000 (*HZ=1000*) software timer interrupts per second in addition to hardware interrupts caused by other peripheral devices. In our system, we set the beacon interrupt interval equal to the time slot value of 25 ms. This results in the generation of at most 40 interrupts per second, as opposed to 20 received by a conventional wireless access point that sends out beacons every 50 ms, indicating that such an overhead is minimal.

4. RELATED WORK

Easy availability, low cost and open-source drivers have led researchers to use 802.11 hardware in ways that far surpass the intended use case scenarios of the 802.11 protocol. Several research efforts have been made to harness the programmability offered by 802.11 hardware to customize the MAC protocols for case-specific scenarios.

802.11 enhancements: In [11, 10], the authors implement a TDMA style MAC protocol on 802.11 hardware for long distance WiFi links. The 2P protocol [11] is a TDMA-style MAC protocol based on synchronous node transmissions, where special *marker* packets are used to indicate synchronous transmission slots. Patra *et al.* proposed WiLDNet [10], which extends 2P with bulk ACKs and FEC-based adaptive loss recovery mechanisms. In [12], an overlay MAC on top of 802.11 is proposed to improve fairness issues in 802.11. Both [10, 12] use *Click* [5] to loosely implement a notion of time slots by controlling the time at which a packet is handed to the device driver for transmission.

Reconfigurable MAC platforms: Neufeld *et al.* proposed SoftMAC [8] as a platform for building experimental MAC protocols. MultiMAC [4] builds on top of SoftMAC to implement multiple MAC protocols. In MadMAC [14], Sharma *et al.* developed a similar platform to implement a TDMA MAC protocol. Both [8, 14] allowed frame formats to differ from those of the 802.11 protocol and, unlike the approaches in [11, 10, 12], provided direct control of MAC operations and the timing of packet transmissions.

Multi-channel MAC protocols: A number of multi-channel MAC protocols have been proposed that require a node with a single radio to switch between different channels on a per-packet or per-timeslot basis [2, 16]. MMAC [16] requires all the nodes in the network to periodically switch to a common control channel, negotiate their future channel selections, and then resume the channel contention mechanism of the IEEE 802.11 protocol after switching to the destination channel. The current draft of the upcoming IEEE 802.11s standard for wireless mesh networks also mentions a similar Common Channel Framework (CCF). In the proposed draft, nodes negotiate switching to a channel with little activity using RTX-CTX messages and return to the common channel periodically. SSCH [2] uses a distributed rendezvous protocol that relies on a pseudo-random generator to construct a channel hopping schedule.

5. CONCLUSION AND FUTURE WORK

In this paper, we have described FreeMAC, a framework for the design and implementation of a generic class of multi-channel wireless MAC protocols, on a typical Linux system using commodity 802.11 devices. FreeMAC provides fine control over several radio communication parameters using API functions. We also propose a novel implementation approach to support the periodic scheduling of time-sensitive MAC functions with high accuracy. We implement a simple multi-channel TDMA MAC protocol using the FreeMAC framework on a testbed of six nodes to demonstrate the utility of FreeMAC as a platform to deploy new MAC protocols on real testbeds. We plan to use the FreeMAC platform to design and validate dynamic TDMA-style multi-channel MAC protocols for deployment in rural mesh networks.

6. REFERENCES

[1] P. Acharya, A. Sharma, E. M. Belding, K. C. Almeroth, and K. Papagiannaki. Congestion-Aware Rate Adaptation in Wireless Networks: A Measurement-Driven Approach. In *SECON'08: Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, San Francisco, CA, USA, June 2008.

[2] P. Bahl, R. Chandra, and J. Dunagan. SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-hoc Wireless Networks. In *MobiCom '04: 10th Annual International Conference on Mobile Computing and Networking*, pages 216–230, Philadelphia, PA, USA, 2004.

[3] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel (3rd Edition)*. O'Reilly, 2006.

[4] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. C. Sicker, and D. Grunwald. MultiMAC - An Adaptive MAC Framework for Dynamic Radio Networking. In *DySPAN '05: First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Baltimore, Maryland, USA, November 2005.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. In *ACM Transactions on Computer Systems*, volume 18, pages 263–297, August 2000.

[6] MadWifi. http://www.madwifi.org.

[7] Maxim 2.4GHz 802.11b Zero-IF Transceivers. http://pdfserv.maxim-ic.com/en/ds/MAX2820-MAX2821.pdf.

[8] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald. SoftMAC - Flexible Wireless Research Platform. In *HotNets'05: Fourth Workshop on Hot Topics in Networks*, College Park, Maryland, USA, November 2005.

[9] OpenHAL. http://madwifi.org/wiki/About/OpenHAL.

[10] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer. WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks. In *NSDI '07: 4th USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, MA, USA, April 2007.

[11] B. Raman and K. Chebrolu. Design and Evaluation of a New MAC Protocol for Long-Distance 802.11 Mesh Networks. In *MobiCom '05: 11th Annual International Conference on Mobile Computing and Networking*, pages 156–169, Cologne, Germany, August 2005.

[12] A. Rao and I. Stoica. An Overlay MAC Layer for 802.11 Networks. In *MobiSys '05: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, Seattle, Washington, USA, June 2005.

[13] S. Ganeriwal and R. Kumar and M. B. Srivastava. Timing-sync Protocol for Sensor Networks. In *SenSys'03: ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, USA, November 2003.

[14] A. Sharma, M. Tiwari, and H. Zheng. MadMAC: Building a Reconfigurable Radio Testbed Using Commodity 802.11 Hardware. In *WSDR '06: First IEEE Workshop on Networking Technologies for Software Defined Radio Networks*, Reston, VA, USA, September 2006.

[15] I. Sheriff, P. Acharya, and E. M. Belding. Resource Estimation on Wireless Backhaul Networks. In *WICON'07: Third Annual International Wireless Internet Conference*, Austin, Texas, USA, October 2007.

[16] J. So and N. H. Vaidya. Multi-Channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals Using A Single Transceiver. In *MobiHoc '04: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 222–233, Tokyo, Japan, May 2004.