MadMAC: Building a Reconfigurable Radio Testbed using Commodity 802.11 Hardware

Ashish Sharma, Mohit Tiwari, Haitao Zheng
Department of Computer Science
University of California, Santa Barbara, CA 93106 U.S.A
Email: {asharma, tiwari, htzheng}@cs.ucsb.edu

***Abstract*— Essential to adaptive devices is the ability to reconfigure Medium Access Control (MAC) protocols to environment conditions and application requirements. We propose MadMAC, a platform for building reconfigurable MAC protocols on commodity 802.11x hardware. Programming on top of MadWiFi, MadMAC transmits packets at configurable time and frame format. In this paper, we build a TDMA-based MAC protocol using MadMAC, and examine the impact of various design parameters. Experimental results show that MadMAC allows flexible control of protocol settings with small processing overhead. We also observe that the TDMA MAC protocol provides 20% throughput improvement over the CSMA protocol in a simple two-node network.**

I. Introduction

Proliferation of wireless networks under a limited spectrum requires adaptive radio devices that can adapt to various network settings and spectrum availabilities. Essential to adaptive devices is the ability to reconfigure Medium Access Control (MAC) protocols to efficiently access spectrum resources and avoid interference from concurrent transmissions.

To design a reconfigurable MAC layer, we can apply lessons from existing proposals that have been designed for different network configurations. Most of them can be categorized as either *reservation based channel access* or *random channel access*. Reservation based channel access protocols such as TDMA [5] and spatial TDMA [10] avoid contention by separating transmissions across orthogonal resource boundaries using *a priori* assignment. Transmission links are assigned with a set of time slots and channels to avoid conflict. These protocols are ideal for links with stable, predictable traffic loads. In contrast, random channel access protocols such as 802.11x CSMA take a distributed and on-demand approach, where nodes vie for transmission resources as necessary to forward their messages. This approach provides adaptivity to changing traffic requirements, avoids contentions without any central server or explicit user coordinations.

While both MAC protocols have distinct advantages, they work best under a particular network/application scenario. An adaptive MAC protocol should observe network environments and adjust its behavior according to application requirements. One simple solution is to switch between these protocols. While this concept is simple, the application to existing wireless devices is challenging. The majority of existing wireless devices are 802.11 devices, and hence confined to the built-in CSMA protocols. In addition, most experimental testbeds that were developed for networking research make use of 802.11 device due to their low-cost and wide-availability. As a result, many networking designs are limited to CSMA MAC protocols.

Our goal is to build a wireless testbed using commodity 802.11 hardware with a reconfigurable MAC layer. We propose MadMAC, a kernel-mode driver to build new MAC protocols, motivated by the work in [11]. This paper makes the following three contributions.

- By programming radio hardware, we design a MadMAC driver to transmit packets at a configurable time and frame format, without triggering CSMA contention and backoff. MadMAC is a kernel-mode driver built on top of MadWiFi, allowing tight control of radio hardware with minimal delay.
- Using MadMAC, we implement a TDMA based MAC protocol with tight timing control and reconfigurable slot structure. We address several design challenges to maintain persistent slot structure and continuous packet transmissions.
- We build a two-node testbed using two laptops with Linksys wireless 802.11a/b/g card. We measure MadMAC performance using both ICMP ping and bi-directional UDP flows. We verify that MadMAC provides tight control of transmission timing at a small processing overhead. We also measure the flow throughput and packet round-trip-time (RTT), and the impact of slot configurations. Finally, we compare the MadMAC TDMA protocol to CSMA and cheesyMAC protocols, and observe a 20% throughput improvement.

The rest of this paper is organized as follows. In Section II we review prior efforts on building reconfigurable protocols using 802.11x devices, and present the unique contributions of MadMAC. Section III provides an overview of MadMAC, and Section IV presents the detailed implementation of each component. We summarize experimental results in Section V. Finally, we conclude in Section VI and describe on-going directions.

II. Related Work

The low-cost of 802.11 device and its wide availability has made it the de-facto choice for developing and evaluating new wireless systems and applications. Many researchers have successfully built wireless 802.11x testbeds of different scales, from home [12] and local area mesh networks [8], [14], [9], to campus wide systems [1] and metropolitan networks [6].

Using both experimental simulations and testbed measurements, many studies have identified the limitations of the conventional CSMA MAC protocols used by 802.11x devices, and proposed modifications and alternatives. In addition to incorporating these new designs in future wireless devices, several efforts modify commodity 802.11x devices for immediate benefits. We can categorize these efforts into two types. The first approach builds an overlay on top of the CSMA MAC layer to implement alternative MAC protocols without modifying the built-in MAC operations [13], [15]. The overlay allows loose control of the packet transmission time by controlling the content and size of the network buffer. The second approach utilizes the reconfigurability of 802.11 devices with Atheros chipsets and MadWiFi driver to control transmission timing and format. The work by Neufeld *et al.* proposes SoftMAC [11] to support different frame formats and control over transmission timing. As an application of the SoftMAC architecture, Doerr *et al.* implemented a set of MAC protocols [2]. Unlike the overlay approach, this approach allows a direct control of MAC operations at much finer time granularity.

III. MadMAC Overview

MadMAC is a kernel-mode driver built on top of the MadWiFi driver. This configuration allows MadMAC to interact with the underlying radio device in real time, avoiding overheads due to context switching. Using the open-source programming capability provided by MadWiFi, MadMAC reconfigures 802.11 devices to transmit packets at controllable time and frame formats, without confining to the contention and backoff procedures of CSMA. Next, we describe the required device configurations, and present the general architecture of MadMAC.

A. System setup

We build MadMAC on laptops running Linux 2.6.15-26 kernel, compiled with gcc-4.0.3. We use Linksys WPC55AG wireless cards with Atheros chipsets, each equipped with an extended MadWiFi [7] driver and uses the Hardware Abstraction Layer (HAL) to take control over the radio hardware.

Before deploying MadMAC, we need to disable selected functions of CSMA protocols. Following the procedure proposed in [2], [11], we operate each device in the “monitor” mode to disable several default functions, including the default MPDU format, automatic acknowledgement (ACK) and retransmissions, RTS/CTS signalling and virtual carrier sensing. We control the duration of backoff by setting the value of (CW_{min} and CW_{max}), the maximum and minimum values of the contention window. We also set the *retry* flag to prevent outgoing data packets from being stamped with MAC sequence numbers.

B. MadMAC Architecture

The proposed MadMAC framework consists of the following three components, shown in Fig. 1.

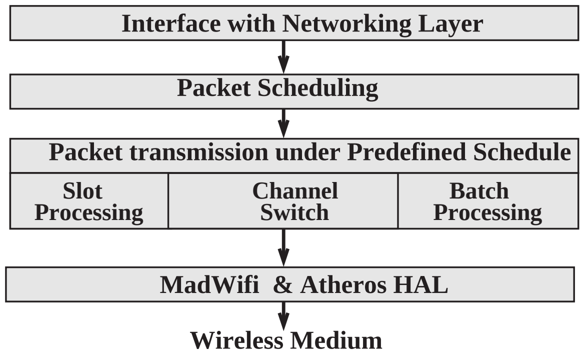

Fig. 1. MadMAC Architecture

- *Interface with the network layer*
  This module is responsible for transporting packets between the MAC and network layers. The interface converts out-going application packets (*i.e.* IP packets) into MAC frames in a format defined by the MAC protocol; and retrieves application packets from in-coming MAC frames.
- *Packet transmission and reception*
  This is the key component of MadMAC. It is responsible for transmitting or receiving packets according to a specific schedule, defined by the MAC protocol[1]. Using a well-defined schedule[2], packets do not experience contention or back-off. Transmission schedules, such as TDMA time slot duration, transmission mode per slot, can be configured through the `/proc`.
- *Peer synchronization*
  Since time-driven transmissions require tight clock synchronization, we implement a simple synchronization software. Using the given hardware device, it has a precision of 25μs.

IV. MadMAC Implementation

In this section, we describe each MadMAC component in detail.

A. Interface with Network Layer

This module is responsible for transporting packets between the MAC and network layers following their frame format. It accepts out-going packets from the network layer (IP packets), adds MAC specific frame headers and checksum, and enqueues them at the MAC buffer. Upon receiving MAC frames from the medium, it strips the MAC header and forwards the packet payload to the network layer.

We implement these functions using a *request routine*. Algorithm 1 summarizes the procedure for processing packets

[1]In a TDMA based system, each radio device transmits only during its scheduled time slots.

[2]In this paper, we focus on implementing packet transmissions under predefined schedules. The derivation of time schedules is beyond the scope of this paper. In our experiments, we assume a 2-node system where two nodes transmit at alternating time slots.

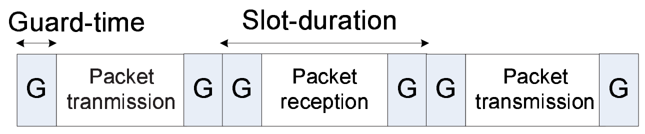

Fig. 2. TDMA Time Slot Structure.

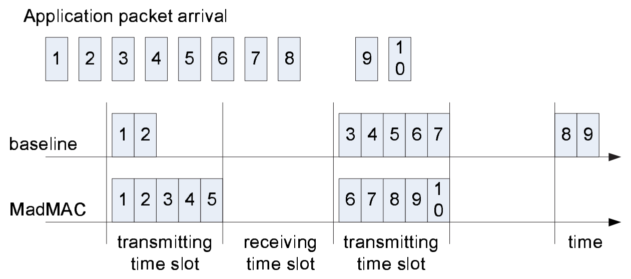

Fig. 3. MadMAC Packet Transmissions.

from the network layer to the MAC layer. We override the default 802.11 frame structure (MAC overhead) and add an ethernet interface, similar to that of [11]. Our implementation can also be extended to use other frame structures.

Algorithm 1 MadMAC Algorithm: Request Routine

```
1: Request_routine (packet) {
2: $receive\_packet\_from\_network\_layer(packet)$;
3: $mac\_frame = add\_mac\_headers(packet)$;
4: $enqueue\_at\_tail\_of\_buffer(mac\_frame)$;
   }
```

B. Scheduled Packet Transmission and Reception

We implement a *service routine* to transmit packets at pre-defined schedules. The routine acquires the transmit schedule and protocol parameters when MadMAC initializes. It wakes up periodically following the schedule and configures the radio hardware to transmit or receive packets. In the MadMAC enabled TDMA system, this module tracks the remaining time during a transmitting slot to maintain a persistent slot structure. At the end of a transmitting slot, this module re-enqueues the un-transmitted packets, configures a wake-up timer at the beginning of the next transmitting slot.

In this paper, we focus on implementing a TDMA protocol using MadMAC. The main design challenges are 1) how to maintain the slot structure, and 2) how to efficiently process multiple packet transmissions in each slot.

Issue 1) Maintaining the time-slot structure

Immediately switching operating modes (transmit, receive, sleep) at slot boundaries can incur unnecessary packet loss. In addition to clock drift between the transmit and receive nodes, delay fluctuations in radio configuration and packet transmission could result in mismatches of slot boundary. To compensate for these impairments, our TDMA system introduces a *guard-time* between consecutive time slots, shown in Figure 2. During guard-time, devices can configure their transmission mode and operating channels, and perform time synchronization. MadMAC provides the flexibility of reconfiguring these parameters according to the radio hardware's reconfiguration speed[3], the average packet transmission time, and the precision of clock synchronization.

[3]We are currently expanding MadMAC to enable channel switch across time slots, and hence the configuration of guard-time should consider the channel switching delay. We have performed some initial tests and measured the delay to be approximated 4-5ms.

The performance of the TDMA system depends heavily on the configuration of slot duration and guard-time. However, the choice is a tradeoff between protocol efficiency and application delay requirements. Since guard-time increases the protocol overhead, we should use large slot duration and small guard-time for better protocol efficiency. On the other hand, packet delay and jitter increase with slot duration - packets arriving after a transmit slot have to wait till the next transmit slot. The choice of guard-time depends on hardware configuration and synchronization precision. We will further evaluate their impact through experimental results in Section V.

Issue 2) Processing multiple packets

The slot-duration is in general much larger than the per-packet transmission time. Hence, one challenge facing MadMAC is how to process multiple packet transmission efficiently in each time slot. We need to validate time before each packet transmission to ensure a precise slot structure, incurring computation overhead. To control the overhead, we can pre-compute the maximum number of packets that can be transmitted in the current slot, assuming that packets are of the size of the Maximum Transmission Unit (MTU). When a slot starts, MadMAC sends a batch of packets together without performing per-packet time validation.

This approach is efficient when the buffer is backlogged. At the beginning of a slot, the transmitter can take a batch of packets from the buffer without recomputing the residue time or number of packets to send. However, most wireless devices have limited buffers, and application packets arrive sequentially.

Using the batch approach, the packets arriving after the start of a transmit slot have to wait till the next transmit slot. This not only increases packet delay and jitter, but also could lead to buffer overflow and dropping of packets (see Figure 3). To address these issues, the service routine periodically checks the MAC buffer during the transmit slot, and serves incoming packets continuously (see Figure 3).

C. Synchronization

We synchronize devices to a central node. We adopt the synchronization protocol proposed in [3] that was originally designed for motes with an accuracy of 20μs in a sensor network.

Algorithm 2 MadMAC TDMA Algorithm: Service Routine

```
1: Service_routine {
2: if $current\_slot \neq my\_transmit\_slot$ then
3:    $timer\_value = estimate\_time\_to\_next\_tx\_slot()$;
4:    $reset\_timer(timer\_value)$;
5:    $return$;
6: end if
7: $/* \ Check: \ Guard \ time \ at \ the \ start \ of \ slot */$
8: if $guard\_time < (current\_time - current\_slot\_start\_time)$ then
9:    $/* \ Check: \ Guard \ time \ at \ the \ end \ of \ slot */$
10:   while $(current\_slot\_end\_time - current\_time) > guard\_time$ do
11:      if $buffer\_is\_empty()$ then
12:         $break$;
13:      end if
14:      $dequeue\_from\_tail\_of\_buffer(packet)$;
15:      $tx\_duration =$
16:       $estimate\_tx\_duration(packet, \ bit\_rate)$;
17:      if $(current\_slot\_end\_time - current\_time - tx\_duration) \geq guard\_time$ then
18:         $transmit(packet)$;
19:      else
20:         $enqueue\_at\_head\_of\_buffer(packet)$;
21:         $break$;
22:      end if
23:   end while
24: end if
25: $timer\_value = current\_time + service\_period$;
26: $reset\_timer(timer\_value)$;
27: $return$; }
```

During neighbor discovery, a node claims the role of *master* while its neighbors behave as *slaves*. The master node broadcasts a packet to its peers with its system time. The broadcast packet received at each slave node experiences different propagation delay, resulting in time difference across slave nodes. Slave nodes respond to the master with its system time information, from which the master derives a correction factor and replies. Each slave node then adjusts its time according to the correction factor.

Our current implementation synchronizes nodes with a 25μs precision when MadMAC initializes. In our experiments, the clock drift between a pair of nodes is at a rate of 10μs per second. Hence, we need to synchronize nodes at least once every 100 second to maintain persistent slot structure. Our preliminary synchronization tool is appropriate single-hop networks. We are currently working on extensions to multi-hop systems.

V. Experimental Results and Discussion

We evaluate the performance of MadMAC using a simple 2-node network (see Figure 4). The MadMAC-TDMA protocol assumes a simple transmit schedule where nodes alternate across time slots. Both devices operate on a 802.11a channel. We synchronize nodes at the initialization.

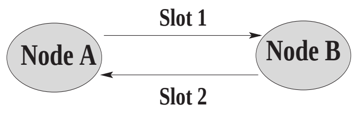

Fig. 4. Experimental Setup

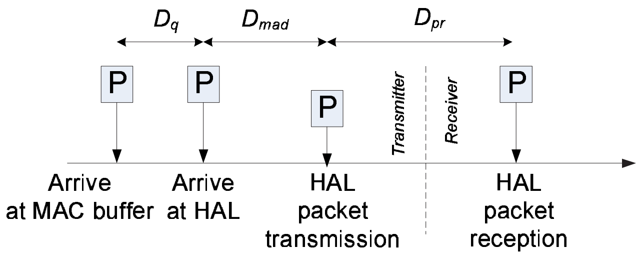

Fig. 5. Definitions of MadMAC processing overhead D_{mad} and propagation overhead D_{pr}.

We perform two set of experiments to measure packet delay and flow throughput. In the first set of experiments, we send a ping packet (64 bytes) every second, and measure the round-trip delay as well as the MadMAC processing overhead. In the second set of experiments, we inject bi-directional UDP traffic to the system and use Iperf [4] to measure flow throughput. We compare the performance of TDMA protocol to that of CSMA and CheesyMAC [11] protocols.

A. MadMAC Packet Processing Overhead

We start by evaluating the efficiency of MadMAC by measuring processing overhead. The overhead D_{mad} is defined as the time lag between when a packet transmission request is sent to the HAL layer (during a transmitting slot) and the subsequent reception of a callback from the HAL layer indicating the completion of transmission. D_{mad} includes the time to transmit a packet at the physical layer and thus scales with the packet size. However, D_{mad} does not include the queuing delay when packets arrive outside of a transmit slot, referred to as D_q. We also measure *propagation delay* D_{pr}, the difference between the packet transmission time and the time the packet is received at the destination. Figure 5 illustrates the definition of D_{mad}, D_q and D_{pr}.

By time-stamping transmitted and received packets at both nodes, we measure the following factors,

- $Tx(m)$: local time at node m when MadMAC requests a packet transmission.
- $TxDn(m)$: local time at node m when MadMAC receives an indicator of successful packet transmission from the HAL layer.
- $Rx(m)$: local time at node m when its MadMAC layer receives a notification of receiving a packet from the HAL layer.

We estimate the MadMAC overhead at node A as

$$D_{mad} = TxDn(A) - Tx(A)$$

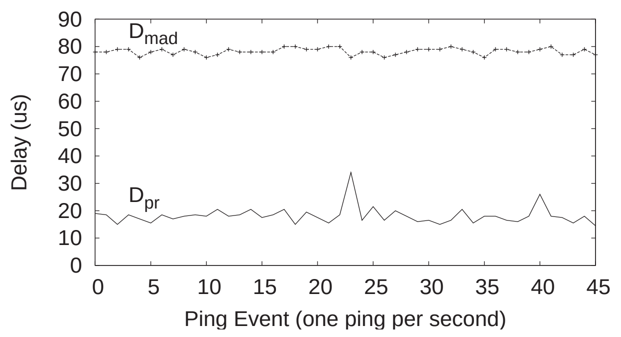

Fig. 6. MadMAC Processing overhead D_{mad} and Physical Processing/Propagation overhead D_{pr} measured from ping packets over 50s, one ping per second, 64 bytes per ping packet.

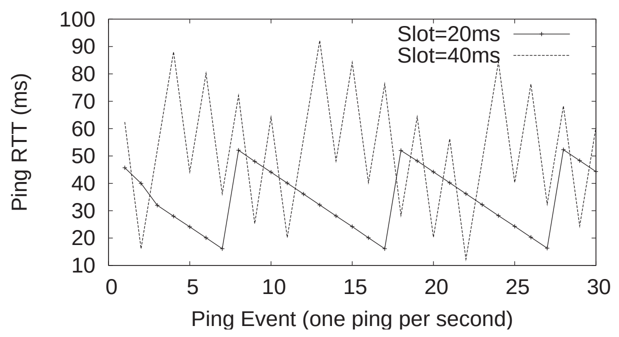

Fig. 7. Individual Ping Round Trip Time (ms) measured under different configurations of slot-duration, assuming 4ms guard-time.

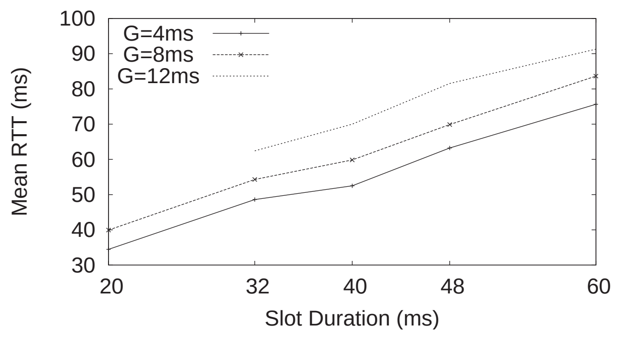

Fig. 8. The mean Round Trip Time (ms) measured under different configurations of slot-duration and guard-time.

and the propagation delay from node A to B as

$$D_{pr} = Rx(B) - TxDn(A).$$

To remove the impact of time drift between nodes, we compute the propagation delay by solving the following equations:

$$\begin{aligned} TxDn(A) + D_{pr} - CD &= Rx(B) \\ TxDn(B) + D_{pr} + CD &= Rx(A) \end{aligned} \quad (1)$$

where $CD = Clock(A) - Clock(B)$ represents the clock difference between node A and B. Note that this assumes the propagation delay between A and B is symmetric.

We measure D_{mad} and D_{pr} by sending ICMP ping packets (64 bytes) every second from one node to another. We timestamp *Tx, TxDn, and Rx* events at both nodes, and solve the above equations to derive D_{mad} and D_{pr}. Figure 6 shows the measurements for 45 ping requests. We see that MadMAC introduces an average delay of 78μs to process the ping packets. It is comparable to the propagation delay D_{pr} (15-20μs). Another interesting observation is that we observe a clock difference of $984 - 1464\mu$s over the course of the experiment, indicating a clock drift between the nodes at a rate of 10μs per second. As stated earlier, we can use this measurement to determine the synchronization period.

B. Ping Packets based TDMA Performance

We examine packet round-trip-time (RTT) by sending ping packets one per second. Fig. 7 shows the individual ping packet RTT under different slot-duration values. We see that RTT varies significantly over time since packets experience different levels of queuing delay due to TDMA schedule. The minimum and maximum RTT for a ping packet are $(2\cdot$ guard-time) and $(2\cdot$ slot duration + $2\cdot$ guard-time), respectively. We further examine the average RTT and its variance (jitter) in Fig. 8 and 9. Both scale linearly with the slot duration. In addition, guard-time has great impact on mean delay, but no impact on delay jitter.

C. TDMA Throughput Performance

We examine flow throughput by injecting bi-directional UDP traffic. Figure 10 shows a sample packet transmission trace in terms of the index of received packets over time. The results verify the time slot based transmission schedule. Each slot (20ms) transmits approximately 66 packets of 1470 bytes, resulting a flow throughput of $0.5(66\cdot1470\cdot8/20)$ = 19.9Mbps, and a total system throughput of 38.8Mbps.

In Figure 11, we examine the total system throughput measured by Iperf, under different configurations of slot duration and guard-time[4]. For a fixed slot duration, increasing guard-time reduces the effective transmission time in a slot and the system throughput. For a fixed guard-time, increasing slot-duration reduces the slot overhead and increases the system throughput. With small guard-time of 4ms, the impact of the slot duration on the system throughput is relatively small.

Next, we compare the throughput performance of the TDMA protocol to that of CSMA and cheesyMAC [11] protocols. We disable the RTS/CTS in CSMA for a fair comparison. Figure 12 illustrates the stacked throughput of both flows using different protocols. TDMA achieves the best throughput which is 20% better over CSMA, while cheesyMAC achieves the worst performance, 28% degradation over CSMA. In addition, we examine the fairness by examining individual flow throughput. We see that both TDMA and CSMA protocols provide balanced throughput to both flows, while cheesyMAC

[4]The system does not support 20ms slot duration with 12ms guard-time, since we require slot duration $> 2\times$ guard-time, so there is no result for this configuration.

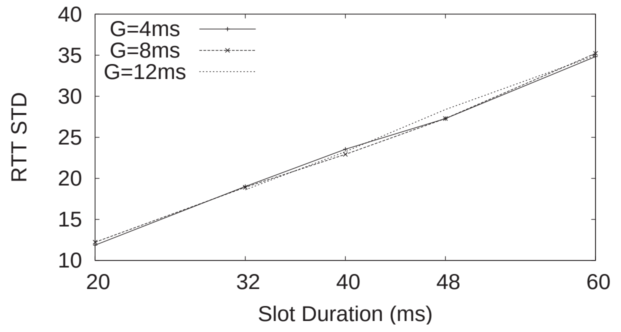

Fig. 9. The standard deviation of Round Trip Time (ms) measured under different configurations of slot-duration and guard-time.

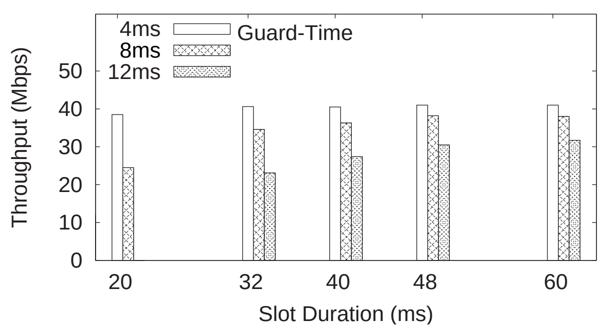

Fig. 11. System throughput under different slot configurations.

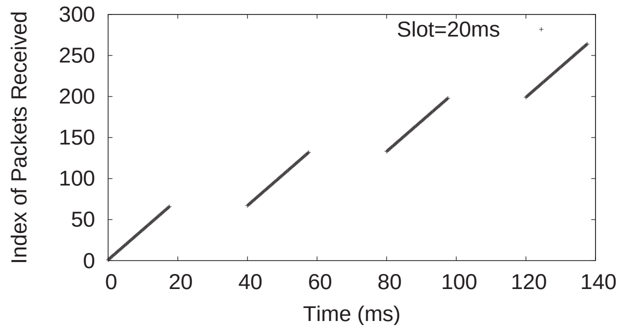

Fig. 10. The number of packets received at a MadMAC peer. Packets are of 1470 bytes. Slot duration=20ms, guard-time=4ms.

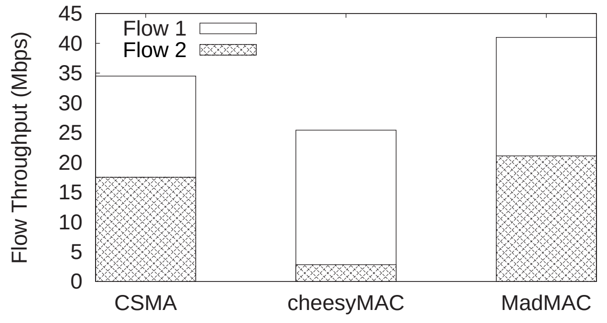

Fig. 12. Flow throughput (stacked) using different MAC protocols.

has poor fairness since one flow dominates the use of spectrum channels.

VI. Conclusion and Future Work

In this paper, we explore the feasibility of implementing reconfigurable MAC protocols on commodity 802.11x devices. We propose MadMAC, a kernel-mode driver built on top of MadWiFi, to provide packet transmission and reception with configurable time and frame formats. We implement a TDMA protocol using MadMAC, and perform experiments to examine the impact of various design parameters. Preliminary results in a 2-node network show that MadMAC provides flexibility to control packet transmission time and format with minimum processing overhead. We also show that the use of MadMAC TDMA protocol achieves 20% throughput improvement over the CSMA protocol. We are currently working on expanding MadMAC to perform real-time per slot channel switch, and conducting experiments on multi-hop scenarios.

References

[1] Dartmounth College. CRAWDAD: A community resource for archiving wireless data at dartmouth. http://crawdad.cs.dartmouth.edu/.

[2] Doerr, C., Neufeld, M., Fifield, J., Weingart, T., Sicker, D. C., and Grunwald, D. Multimac - an adaptive mac framework for dynamic radio networking. In *First IEEE Symposium on New Frontiers in Dynamic Spectrum Networks (DySPAN)* (November 2005).

[3] Ganeriwal, S., Kumar, R., and Srivastava, M. Timingsync protocol for sensor networks. In *Proc. of ACM SenSys* (2003).

[4] Iperf. `http://dast.nlanr.net/Projects/Iperf/`.

[5] Koutsakis, P., and Paterakis, M. On multiple traffic type integration over wireless TDMA channels with adjustable request bandwidth. *International Journal of Wireless Information Networks 7* (April 2000), 55–68.

[6] LaMarca, A., et al. Place lab: Device positioning using radio beacons in the wild. In *Proc. of Pervasive* (June 2005).

[7] MadWifi. `http://www.madwifi.org`.

[8] Microsoft. Microsoft Research Mesh Networking. http://research.microsoft.com/mesh/.

[9] MIT. MIT roofnet. http://pdos.csail.mit.edu/roofnet/doku.php.

[10] Nelson, R., and Kleinrock, L. Spatial TDMA: A collision-free multihop channel access protocol. *IEEE Transactions on Communications* (1985), 934–944.

[11] Neufeld, M., Fifield, J., Doerr, C., Sheth, A., and Grunwald, D. Softmac - flexible wireless research platform. In *Fourth Workshop on Hot Topics in Networks (HotNets)* (November 2005).

[12] Papagiannaki, K., Yarvis, M., and Conner, W. S. Experimental characterization of home wireless networks and design implications. In *Proc. of INFOCOM* (April 2006).

[13] Rao, A., and Stoica, I. An overlay MAC layer for 802.11 networks. In *Proc. of Mobisys* (April 2005).

[14] UCSB Moment Lab. UCSB wireless mesh network testbed. http://moment.cs.ucsb.edu/meshnet/.

[15] Wu, H., Wang, X., Liu, Y., Zhang, Q., and Zhang, Z.-L. Softmac: layer 2.5 mac for voip support in multi-hop wireless networks. In *Proc. of IEEE SECON* (September 2005).