Masters Project Report: Second Stage
Design, Implementation And Evaluation of Multi-Hop Wireless TDMA System

Nirav Uchat

Guide. Prof. Kameswari Chebrolu and Prof. Bhaskaran Raman

Computer Science And Engineering Department
IIT Bombay, Mumbai

Abstract

Providing WiFi connectivity to remote isolated villages is a challenging task. It is well-known that 802.11 is not suitable for long distance links due to inefficient carrier sensing and link layer recovery. We therefore propose a TDMA-based MAC protocol which performs better in such situations due to its tight control over packet transmission timing. As a first step, we have developed a framework which will facilitate the implementation of TDMA-based MAC protocol. In this report, we describe the challenges faced in preparing this test-bed for TDMA implementation.

1 Introduction

Our objective is to give access to World Wide Web to the people living in remote villages using off the shelf cheap hardware while maintaining high data rate. This motivated us to use unlicensed frequency band and standard 802.11 protocol. However, 802.11 protocol was designed to work over a small distance (generally 50-100 meters) due to which there is a severe drop in the performance when it is used over longer ranges. In particular, it has been shown that carrier sensing in presence of external interference over longer ranges results in unpredictable protocol behavior [7]. In such situations, TDMA-based MAC proves to be a reliable alternative [1]. Existing TDMA implementations are limited to a single hop and in most cases between two devices. Our aim is to design multi-hop TDMA with precise time synchronization to cater to a large user base.

The remainder of the report is organized as follows. In section 2 we define the problem statement. In section 3 we give a brief overview of the work done during stage one of the project. In section 4 we explain the fundamental TDMA design. We also explain the basic modules which build up the proposed TDMA system. The proposed design may change as per the requirements during third stage of the project. In section 5 we will look at implementation done in the second stage of the project. Finally, the future work is listed in section 7 followed by conclusion.

2 Problem Statement

The aim of this project is to design, implement and evaluate a multi-hop wireless TDMA system in place of standard 802.11 CSMA/CA protocol. The proposed system should work for both long distance and local networks and it should support voice, video and www traffic. The intended end users are devices having ethernet port or wireless devices capable of running modified TDMA protocol.

3 Stage One Overview

3.1 Introduction

A large portion of the MAC protocol is implemented in software which makes it possible to design and develop alternate MAC protocols. In particular, it is possible to change the software to replace the CSMA-based MAC in 802.11 with TDMA-based MAC. In the next section, we will look at some such TDMA type extensions of 802.11 protocol. We also discuss an overlay implementation, wherein the MAC layer is left unchanged and an abstraction layer is built on top of it.

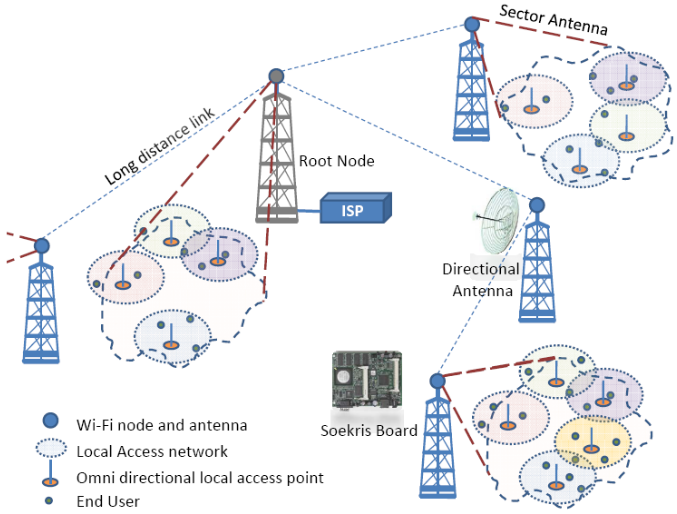


Figure 1: FRACTEL Architecture

3.2 Related Work

softMAC[2] proposed by Neufeld *et al.* gives us generic platform to experiment with MAC protocol. It disables CSMA/CA by switching device in monitor mode. Author lists six basic steps to disable CSMA/CA. As an example author has given sample TDMA system between two device. Though we are not going to use softMAC framework, we do require to disable CSMA/CA and softMAC gives steps to achieve it.

MadMAC[3] proposed by Sharma *et al.* extends softMAC and implements TDMA system between two node with tight time synchronization. It claims to give 20% throughput improvement then standard CSMA/CA system. It uses custom frame format, which includes guard time between transmission slot for synchronization. Our proposed TDMA system will require tight time synchronization and MadMAC provides some initial pointers for it.

On other hand, MultiMAC[4] , which also extends softMAC but handles multiple MAC protocol. A special marker in each packet identifies the protocol to use to decode the packet and passes to the network stack. These might incur packet processing delay. MultiMAC can be useful in situation where one wants to switch to CSMA for short distance and TDMA for long distance Link.

FreeMAC[5] goes one step ahead and provide multichannel communication. It uses hardware beacon interrupt timer in place of software kernel timer for fine control over packet transmission. It also characterizes delays involved in channel switching. It uses monitor mode with custom frame format. In first stage we are not looking for multi-channel TDMA system but we do require more precise timer than software kernel timer. In our implementation, we might use something similar to beacon timer proposed by FreeMAC.

WiLDNet[7], implements TDMA system with bulk ACK and FEC type loss recovery mechanism. Unlike softMAC, the WiLDNet uses 802.11 frame format. It uses click[8] module on top of MAC layer to implement bulk ACK and FEC mechanism and does some MAC layer modification. Our proposed system is fundametally different from WiLDNet. One of the key contribution of WiLDNet is it gives insight in to understanding of 802.11 poor performance in long distance link.

Overlay Layer[6] proposed by Rao *at el.* builds on top of MAC layer to achieve fairness issue related to 802.11. It is some what different approach than changing MAC itself. It runs on top of MAC layer and uses functionality provided by MAC layer. In general it can control packet queue buffer but has no control over packet transmission timing. Our proposed system requires precise control over packet transmission and overlay layer is not capable of it.

Table 1 shows the comparison between different MAC protocol that we have discussed with our proposed system(FRACTEL).

Taking a birds eye view, the overlay layer sits above MAC layer and thus provide loose control over underlying hardware. While implementing at MAC layer gives direct access to hardware and provides tighter control over it. After considering both methods, we decided to implement TDMA at MAC layer for better control over time critical function in TDMA. In next section we will look at work done in first stage of the project.

3.3 Work Done

We are using Soekris board [10], which runs voyage linux, a strip down version of debian distribution. Soekris board has 266MHz CPU with 128 MB RAM and 256 MB hard drive. It also has provision of attaching minipci WiFi card. On software side we are using madwifi driver [9], an open source WiFi device driver for Atheros wireless chipset in Linux. Shown in Fig 1, is our envisioned architecture of proposed system.

Implementation of TDMA protocol at MAC layer requires disabling default CSMA protocol itself. Disabling CSMA in madwifi was not a trivial task. Before disabling CSMA, we had to first understand the complete transmit and receive path in madwifi driver and going through minute detail of interrupt handler, tasklets and different data structures. Our first task in stage one was to setup Soekris board with voyage linux and installing madwifi driver on it. Once done, we

	softMAC	MADMac	MultiMac	FreeMac	Overlay	FRACTEL
Works At	Mac	Mac	Mac	Mac	Above Mac	Mac
Timer Type	Hardware	Software	Software	Hardware	Software	Software
Multi Channel	No	No	No	Yes	No	Yes
Time Sync.	loose	loose	loose	Better	loose	Custom Protocol
Multi-Hop TDMA	No	No	No	No	No	Yes
TDMA Schedule	Static	Static	Static	Static	Static	Dynamic
Pkt. Gen. At MAC	No	No	No	No	No	Yes
Schedule Generation	No	No	No	No	No	Yes

Table 1: Protocol Comparison

looked at various commands such as wlanconfig, iwlist and iwconfig to create and configure wireless adapter. We then tried to disable CSMA as explained by [2] and [5]. Author lists the following six tasks in order to disable CSMA :

1. Disabling MAC level ACK
2. Disabling RTS/CTS exchange
3. Override 802.11 frame format with custom TDMA frame
4. Disable virtual carrier sensing
5. Disable Transmission backoff
6. Disable CCA (clear channel assessment)

The madwifi driver allows WiFi adapter to run in monitor mode, which when set allows us to sniff traffic. Apart from sniffing, it also achieves first three tasks required for disabling CSMA. However, there was one major drawback in using it. By default, it is not possible to do communication in monitor mode. (Section 5.1 explains it in more detail). We also disabled CCA [2] mechanism by setting noise floor to high value. Though we have followed required steps for disabling it, we have not yet verified it.

Though monitor mode was good fit for implementing TDMA, there were few task that we still needed to achieve for completely turning off CSMA mechanism. They were,

- Disabling virtual carrier sensing
- Disabling transmission backoff
- Enabling communication between two nodes in monitor mode (such as working of ping and ssh)

It was observed that when the adapter is set in monitor mode; the hardware was changing certain bits of the packets put in hardware queue for transmission. Due to which actual data was getting corrupted at the receiver end.

After spending good amount of time in figuring out changes required for monitor mode, we decided to implement initial prototype TDMA system in Ad-hoc mode. It was kind of TDMA on top of CSMA with MAC-ACK disabled. It had very loose time synchronization.

At the end of stage one, we were able to accomplish the following tasks :

- Installation of voyage on Soekris board with madwifi driver
- Understanding madwifi driver basics
- Disabling MAC layer ACK
- Disabling beacons in Station-AP mode
- Understanding transmit and receive path for Monitor and Adhoc mode
- Using RAW packet of monitor mode and changing its content
- prototype TDMA system in Ad-hoc mode

3.4 Timeline For Stage Two

Our first priority was to enable monitor mode communication with complete removal of CSMA. We had set following action items for stage two of the project

- Monitor mode communication
- Deciding TDMA frame structure
- Implementation of TDMA in monitor mode
- TDMA schedule dissemination

In next section we will look at the working of the proposed TDMA system.

4 TDMA MAC Protocol

During the initial stages of stage two we had discussions on MAC protocol and simultaneously started work on monitor mode communication. After detail discussion we came up with three basic operations to be implemented for TDMA MAC. As of now we are not committing on final TDMA MAC protocol. It is important to note that, the proposed design will keep on evolving during the course of the project.

As mentioned earlier, we are implementing TDMA on Soekris platform which is running madwifi driver, hence the proposed TDMA MAC will be running in madwifi driver. Section 5 explains the details of plugging various functionality into madwifi driver. Before we go in to details of working of TDMA protocol, we need to clarify some terminology that we will be using in further discussion.

The **schedule** consists of **schedule header** followed by number of **scheduling elements**. Each scheduling element consists of start time and duration of the event. It also contains information about transmitter and intended receiver for current slot. The scheduling header and scheduling element frame are shown in Fig 5. Each **data frame** will have **data header** as shown in Fig 5. The data header filters the packets at every node as explained in section 5.6. The **contention slots** can be used by any node for sending information. Each schedule will have number of such slots. Also note that we are working in monitor mode and we are using custom frame format i.e. there are no 802.11 frames in our implementation.

Now, we will look at basic modules which constitute the proposed TDMA system.

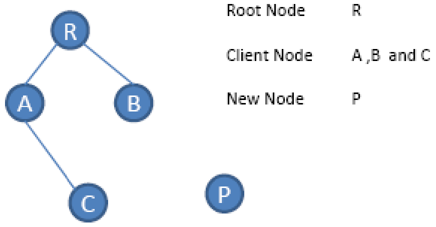


Figure 2: Topology

The system comprises of (as of now) four differrent task. For ease of understanding we will consider topology as shown in Fig 2. The **routing** and **flow request** mechanism for multihop TDMA is still under discussion.

Schedule Dissemination:

- Root creates and broadcasts schedule header along with scheduling element. Both A and B receives it.
- A and B are synchronized using the time stamp and offset.
- A and B independently look inside the scheduling elements and schedule their own send/receive time.
- Suppose A's schedule transmit time arrives. A transmits the entire schedule as a broadcast and the process continued thereafter. Scheduler will not schedule any slot for schedule broadcast for leaf node.
- Suppose A had another scheduling element for sending data from A to C. When A receives the original schedule it creates a queue of it's own event and switches to proper channel for transmission at correct time i.e. using the information send by root node in scheduling element.
- After one schedule ends, all nodes listen on channel 1 (which is, as of now, the default channel for schedule transmission by root node) for the next schedule.

In current implementation, we have limited functionality for schedule dissemination. As of now we are not sending schedule in multiple hop.

Node Join Operation:

- Node P joins the existing network.
- P listens on channel 1 for broadcast schedule.
- P gets multiple such packets, determines best strength possible parent. Synchronization is done with this parent and it also finds contention slot for sending node JOIN request.(contention slot information is in schedule packet that it receives).
- P sends node join request in the next contention slot. P sends a fixed (or variable) number of parent-RSSI pairs in the node JOIN packet.
- The Node JOIN may propagate towards the root in contention slot only. [Piggybacking on working flow might be an optimization]. Intermediate node changes the destination field to it parents.
- When root node receives the NODE JOIN request, it determines which of the parents in the parent-rssi list can be denoted as P's parent

- If P is an intermediate (non-leaf) node, the root puts a scheduling element for P to forward (broadcast) the schedule. This element contains transmitter = P, receiver = assigned parent and appropriate start at and duration field with type = schedule dissemination

- If p is an leaf node, similar element is put except that it has startat=0 and duration=0.

This ensures that p knows about its parent and if it is a leaf, it does not broadcast the schedule.

Flow Request: When node wants to send the data it will send flow request message destined for root node. It's up to the root node to allocate transmission slot for requesting node in next schedule. As of now we have not come up with exact mechanism for this. At implementation front, we have allocated slot for every node in the network such that every node has at least one slot for transmission in given schedule.

Routing : Design and Implementation of multi-hop routing will be done in third stage of the project.

The next section explains the work done during second stage to achieve some of the above design specifications.

5 Work Done

In stage two we have looked at several issues in preparing base for TDMA implementation. We started with enabling communication in monitor mode which took considerable time as it required us to look at bit level information in packets. Once monitor mode changes were done, we prepared skeleton of structures required for TDMA implementation. It includes schedule header, scheduling elements and data header. Once scheduling structures were ready, we were looking for a way to send such custom packets from MAC layer itself as sending it from upper layer would incur additional delay. Section 5.5 explains how we achieve generation of custom packets form MAC layer. Apart from these, there were other issues such as filtering packets at each node such that every node processes it's own packets and discards others. We also place a check on receiving side such that packets with CRC and PHY error will never reach network layer. Subsection 5.1 to 5.8 list the work done during second stage of the project. Fig 3 is a state diagram of currently implemented system. The different sub-components of the system are explained in the sub-sections below.

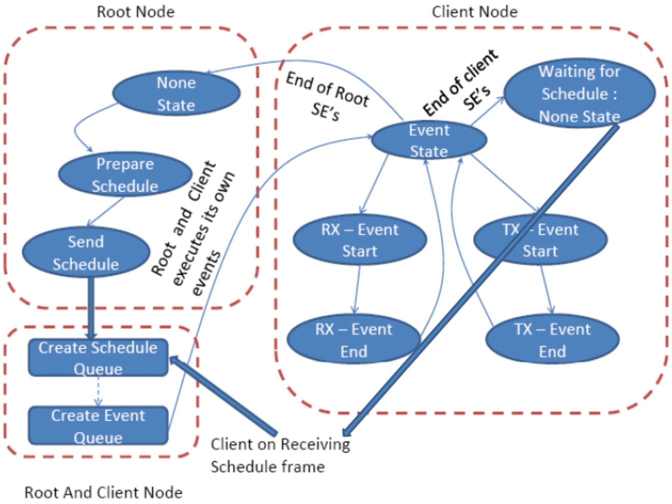


Figure 3: TDMA State Diagram

5.1 Monitor Mode Changes

At the end of first stage we were struggling to enable communication in monitor mode. As explained earlier, when adapter is set in monitor mode, the hardware was changing few bytes before sending packets on air. These changes were different for different type of packets as shown in Fig 4. In case of normal packets, the hardware was changing byte 23 and 24 before sending it on air. To fix it we replicated 2 bytes at position 23 and 24 at position 25 and 26 and on receiving side, we replaced those two changed bytes with replicated bytes. For ARP packets, the hardware was changing bytes 31 and 32. To fix it, we replicated byte 31 and 32 at position 33 and 34. When packet is received, the receiver removes byte 31 and 32 and shrinks the packet. Thus, on receiving side we get the original packet. In case of normal packets, above behavior was due to retry flag i.e hardware was printing sequence number at position 23 and 24. For ARP packets we still don't know the reason for such behavior.

When packet is received at MAC layer, it goes through various checks and headers are added or removed until it is handed over to network layer. Fig 6 shows the partial flow of packet on sending and receiving side. We also changed few elements of **skb** structure before sending packets to network layer. The **skb** structure is a single entity that flows across MAC and network layer.

After doing above changes, we were able to ping two nodes running in monitor mode. Now, the next step was to design and implement TDMA frame structure for scheduling and data frame.

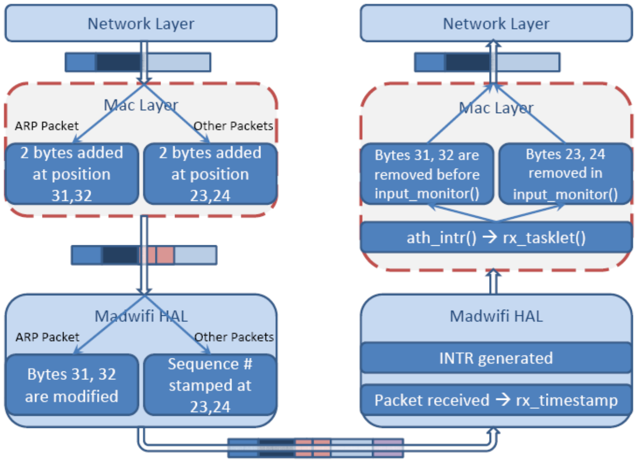
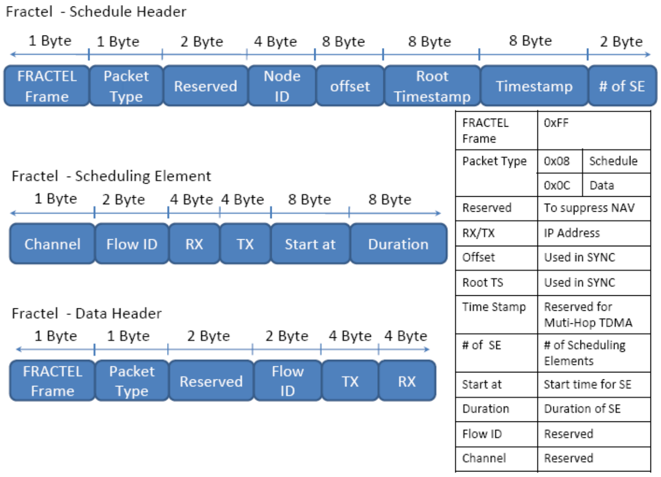

Network Layer
Mac Layer
ARP Packet
Other Packets
2 bytes added at position 31,32
2 bytes added at position 23,24
Madwifi HAL
ARP Packet
Other Packets
Bytes 31, 32 are modified
Sequence # stamped at 23,24
Network Layer
Mac Layer
Bytes 31, 32 are removed before input_monitor()
Bytes 23, 24 removed in input_monitor()
ath_intr() → rx_tasklet()
Madwifi HAL
INTR generated
Packet received → rx_timestamp

Figure 4: Monitor Mode TX/RX


Fractel - Schedule Header

1 Byte	1 Byte	2 Byte	4 Byte	8 Byte	8 Byte	8 Byte	2 Byte
FRACTEL Frame	Packet Type	Reserved	Node ID	offset	Root Timestamp	Timestamp	# of SE

Fractel - Scheduling Element

1 Byte	2 Byte	4 Byte	4 Byte	8 Byte	8 Byte
Channel	Flow ID	RX	TX	Start at	Duration

Fractel - Data Header

1 Byte	1 Byte	2 Byte	2 Byte	4 Byte	4 Byte
FRACTEL Frame	Packet Type	Reserved	Flow ID	TX	RX

FRACTEL Frame	0xFF	
Packet Type	0x08	Schedule
	0x0C	Data
Reserved	To suppress NAV	
RX/TX	IP Address	
Offset	Used in SYNC	
Root TS	Used in SYNC	
Time Stamp	Reserved for Muti-Hop TDMA	
# of SE	# of Scheduling Elements	
Start at	Start time for SE	
Duration	Duration of SE	
Flow ID	Reserved	
Channel	Reserved	

Figure 5: FRACTEL Frames

5.2 TDMA Frame Format

Our implementation consists of schedule header, data header and many scheduling elements. After every schedule header there will be number of scheduling elements. This number is specified in schedule header.

Schedule Header consists of special marker 0xFF indicating itself as a FRACTEL frame followed by packet type i.e indicating whether it is a schedule or data packet. The three timestamp field are used for synchronization purpose. The last entry gives information about number of scheduling elements following the current schedule header. The use of 2 byte reserved filed in both schedule and data frame is explained in section 5.6.

Scheduling Elements contains the information about transmitter and receiver for given slot. Start at field gives information about when to start and duration field gives duration of current transfer. Every schedule frame will consists of one scheduling header and number of scheduling elements. On receiving the schedule frame, each node will create its own event queue. The creation and working of event queue is explained in section 5.3. Every node can either be transmitter or receiver in a given scheduling element but can't be both. The Flow ID and channel field are reserved and will be use during third stage of project.

Data Header is attached to every data frame. It consists of special marker 0xFF to indicate it as Fractel frame followed by packet type as 0xOC indicating it as data frame. It consists of transmitter and receiver IP address. The receiver IP field if used on client side to filter out non-destined packet for that node. Filtering based on MAC address will be discussed in section 5.6. The flow ID field will be used during third stage of our project. Data header is attached to every incoming packet at MAC layer.

5.3 Event Queue Creation

The event queue is used by each node to keep track of its transmitting/receiving slot. Initially client nodes will listen for schedule frame. Once the schedule frame along with scheduling elements is received by client node, it will create schedule queue. This schedule queue will be used during third stage of our project when intermediate client node will be required to send scheduling frame. After creating scheduling queue, client node will create node specific queue by looking at TX and RX field in each scheduling elements. This queue is called event queue and will consists of only node specific scheduling elements. During event queue creation, we also perform node synchronization and discard events that are old. We update start_at field of every scheduling element during queue creation (in accordance with synchronization). Every scheduling element is encapsulated in fractel_event_list queue, which consists of next_at timer value set to start time of next element in queue. This helps us in setting next timer for upcoming event. We will be using fractel_event_list queue for getting scheduling element along with the next_at timer.

Upon firing of timer, **fractel_event_handler** is called, which is a state machine for our TDMA system. Depending on state of the node it will either prepare schedule, send schedule or initiate transmit event or start receiving event. The state diagram is explained in Fig 3. Once the fractel_event_list queue is empty, the node will wait for next schedule and process repeats.

It is the responsibility of the root node to prepare valid schedule along with scheduling elements and transmit across the network. Both root node and client node will be in FRACTEL_NONE_STATE when they boot up. After specified time, root node will prepare first schedule and send it on air. As soon as root sends the schedule, it will start creating fractel_schedule_list queue and then fractel_event_list queue. On the other hand, when client receives the schedule it will first do synchronization and then create same queues as created by root node. Once fractel_event_list queue is ready node can set initial timer and process events in this queue one by one(at specific time).

5.4 TDMA Implementation

For working of TDMA we needed a way to buffer all out going traffic and transmit packets such that cumulative sum of total bytes transfered in a slot does not over shoot the **(slot size $*$ data rate)** product. Fig 6 shows working of prototype TDMA system. The final call in madwifi for tansmitting packet on air is **ath_hal_txstart()**. Initially, we had plug-in TDMA queuing mechanism between ath_txqaddbuff() and ath_hal_txstart(). As explained in section 6, this was not the proper place to do so. Right now we are working on changing TDMA queuing module as shown in Fig 6. The goal is to buffer all incoming traffic and on arrival of interrupt (i.e. send slot for given node - set from even_queue_list()) it will send slot size $*$ data rate number of bytes on air. However, there is small problem with large MTU with small data rate. Section 5.7 addresses this issue.

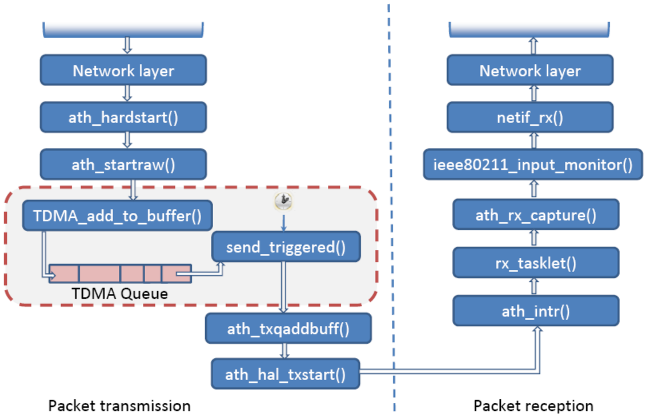


Figure 6: TDMA in Monitor Mode

5.5 Generation Of Packets At MAC Layer

All custom packets explained in section 5.2 can be generated from application layer, but it imposes additional overhead. In our case we were looking to generate such packets from MAC layer itself, so that we incur minimal overhead. We manage to achieve it through ieee80211_getmgtframe() function which allocate and setup a management frame of the specified size and return the sk_buff and a pointer to the start of the contiguous data area that's been reserved based on the packet length. Once we get hold of sk_buff we can use sk_buff's data field to push our data. We use skb_push() and skb_pop() function to manipulate skb→data[] field.

5.6 MAC Filtering And NAV Field

After generating schedule frame, scheduling elements and data frame, we were able to ping between two nodes running on TDMA schedule. But sometimes ping was giving weird reply. We found out that problem was with the way in which we allowed packet to reach network layer on each node. The fact was, we were passing all received packets to network layer even if that packet was not destined for that node. To fix this problem we created static map of IP and MAC address of all nodes.

When packet comes to MAC layer it has MAC address of destination machine. To find IP address we refer to static map of MAC and IP addresses and place corresponding IP address in RX field. When client receives the packet it first matches the RX field with it's own IP. If match is successful then we allow data packet to reach to network layer else we discard the packet.

Now, we will look at purpose of keeping two byte **reserved field** in both schedule and data frame. The position of reserved field is second byte. In IEEE standard, that byte correspond to NAV field. Which when set, tells hardware to defer transmission by corresponding amount. We encountered this problem when we got repeated **no xmit buff** error. The only explanation for such behavior was hardware queue was completely filled due to long delay in transmission. We fixed this problem by setting it to zero.

5.7 MTU For Small Slot Size

In our implementation, we buffer the packet until transmission slot occurs. Once transmission slot starts, we check whether current packet at the head of the queue is small enough to send in current slot. If it is not, then we stop transmission. This was causing problem at small data rate with small slot size. For example, consider 1 Mbps data rate with 1 msec slot size. In such setting maximum data that can be send in one slot is 125 Bytes. In such situation if packet having length more than 125 bytes comes into queue, then TDMA system stops working i.e permanently stopped at packet having size > 125 bytes. To fix this problem, we set MTU such that MAC layer never gets larger packets then what is possible to send in one slot at given rate. Till now we have not automated this calculation. We will plug-in this mechanism in third stage of the project.

5.8 Channel Switching

In TDMA system we need to maximize the available resources. Enabling simultaneous transmission of nearby node on different channel might help in increasing system throughput. Linux inbuilt iwconfig command provides a way to switch channel from command prompt. In order to achieve same functionality from driver code, we traced the command in madwifi through IOCTL call and found a way to switch channel from madwifi driver without using IOCTL call. To achieve it, we need to set ic→curchan structure with intended channel information and then calling ic→ic_set_channel(ic), which writes specific register and changes the operating frequency. It is visible that channel switching will have some overhead. As of now we haven't characterized the channel switching delay. In due course of project we will do the required characterization and also integrate channel switching functionality into madwifi code.

In next section, we will look at results obtained after making above changes in madwifi.

6 Experiments And Results

We carried out experiments by varying slot size and number of scheduling elements. Below table shows the observation obtained in one such experiment. Note that, throughput measurements are in Mbps.

Rate	Slot Size	SE	UDP	
			One-Dir	Bi-Dir
11	20msec	3	7.52	7.53
11	10msec	3	7.35	7.59
11	5msec	3	7.20	7.36

Table 2: UDP Experiment

We gave equal slot to three communicating nodes. The root node was only sending scheduling information. In theory, given equal chance for transmission to each nodes, the throughput should reduce by one thired. But in contrast, we were getting much more throughput than that. For example, in one direction UDP test with slot size 20msec, we were getting around 7.43 Mbps, while by theory we should not cross more than 2.98Mbps. This was a very surprising result for us. We narrow down the reason for such behavior as either transmission node was not obeying transmission timing or packets were getting send without TDMA queuing.

Rate	Slot Size	SE	TCP	
			One-Dir	Two-Flow
11	20msec	3	3.81	4.92
11	10msec	3	4.31	5.2
11	5msec	3	4.40	4.73

Table 3: TCP Experiment

After detailed analysis of our TDMA code, we found implementation bug in our system. To give clear picture, as explained in section 5.4, the TDMA queuing was happening after calling ath_txqaddbuff(). The purpose of ath_txqaddbuff() is to push packet on hardware queue. It then calls ath_hal_txstart(). The purpose of ath_hal_txstart() is to enable transmission on specified hardware queue. As we were queuing packet for TDMA after it was added to hardware queue, all the packets that a node got for transmission were in hardware queue rather than in TDMA queue. So once we call ath_hal_txstart(), it was sending all packets that were on hadware queue on air. So effectively, TDMA was not happening. This was a major bug in our code. We are currently working on it and will fix the bug as soon as possible. Once we are done with the fix, we will do detail throughput measurements and will compare it with CSMA protocol.

7 Stage 3 Timeline

In stage two, we tried to prepare base for implementation of TDMA protocol for stage three. We addressed many issues as listed in section 5. In stage three, we are looking at extending TDMA to multi-hop network. Implementation of TDMA in Multihop setup has it's own set of issues. Following are the key issues that we will address in stage 3

- Multihop routing
- Schedule dissemination across network
- Flow request mechanism
- Indoor and Outdoor benchmarking of Multihop system

Apart from above objective, we might migrate to different hardware platform. The new platform has 680 MHz CPU compared to current 233 MHz CPU. It will require us to cross compile madwifi driver, as it has different processor architecture.

8 Conclusion

Challenges: One of the main challenges of the work was understanding and changing device driver code (particularly when no hardware specification is available). For example, it took us almost one month to figure out that a single flag needs to be set to generate hardware-timestamped packet.

Milestones: A few milestones reached in the second stage include:

- Enabling monitor mode communication
- Generating packets from MAC layer
- Designing a TDMA frame structure and implementing it in madwifi driver code

Though TDMA queuing mechanism needs a major fix, we are optimistic that we can implement multihop TDMA system in third stage of the project.

References

[1] Kameswari Chebrolu and Bhaskaran Raman. FRACTEL: A Fresh Perspective on (Rural) Mesh Networks, ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR'07), A Workshop in SIGCOMM 2007, Aug 2007, Kyoto, Japan.

[2] Michale Neufeld, Jeff Fifield, Christian Doerr, Anmol Sheth and Drik Grunwald. softMAC-Flexible Wireless Research Platform, HotNets-IV, Nov 2005.

[3] Ashish Sharma, Mohit Tiwari, Haitao Zheng. MadMAC: Building a Reconfigurable Radio Testbed Using Commodity 802.11 Hardware, WSDR 2006.

[4] Christian Doerr, Michael Neufeld, Jeff Fifield, Troy Weingart, DC Sicker, Dirk Grunwald. MultiMAC - An Adaptive MAC Framework for Dynamic Radio Networking, DySPAN 2005.

[5] Ashish Sharma, Elizabeth M. Belding FreeMAC: Framework for Multi-Channel MAC Development on 802.11 Hardware, PRESTO 2008

[6] Ananth Rao, Ion Stoica. An Overlay MAC Layer for 802.11 Networks, MobiSys 2005.

[7] Rabin Patra, Sergiu Nedevschi, Sonesh Surana, Anmol Sheth, Lakshminarayanan Subramanian, Eric Brewer. WiLDNet: Design and Implementation of High PerformanceWiFi Based Long Distance Networks, NSDI 2007

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. TOCS 2000.

[9] http://www.madwifi.org

[10] http://www.soekris.com/