Masters Project Report: First Stage
To Design, Implement And Evaluate a Multi-Hop Wireless TDMA System

By 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. The 802.11 standard has its own limitations when used in long distance links. To overcome these, we propose TDMA style MAC implementation using standard low-cost wifi cards. This report describes initial work done to implement proposed system. Initially we will look at related work done by researcher in this field. Then we will discuss advantage of TDMA-MAC over Overlay layer and reasons for choosing such implementation. Later, we will look at prototype TDMA system between two devices with loose time synchronization. Finally to conclude, we will list action items for second stage of project, which includes designing TDMA frame structure and implementing multi-hop TDMA.

1 Motivation

Imagine special wifi platform that can send data from city to rural region 10 to 15 KMs away and connect widely distributed geographical region to internet. This is what our goal 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. These motivates us to use unlicensed frequency band and standard 802.11 protocol. But the fact is, 802.11 protocol was designed to work in small area, generally 50-100 meter and extending it for long range has ill effect on its performance. More precisely, doing carrier sensing in presence external interference and long range results in unpredictable protocol behavior. In such scenario, TDMA MAC can be thought as one of the alternatives. As of now, existing TDMA implementation are limited to single hop and in most cases between two devices. Our aim is design multi-hop TDMA with precise time synchronization to carter large user base.

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.

The rest of the report is organizes as follows. In section 3 we look at related work done in this field. In section 4 we list shortcoming of 802.11 in long distance link and outlines our approach for multi-hop TDMA MAC. Section 5 gives initial work done towards TDMA implementation and section 6 presents the work to be done in next stage of project.

3 Previous Work

In past, several research effort has been made to extend functionality of 802.11 protocol. Moreover, wide availability of commodity hardware and ease of programmability provided by open source driver has helped a lot. In this section we will look at some of the TDMA type extension to 802.11 protocol. Then we look at overlay implementation, which is a protocol on top of MAC layer. We will also look at collision free TDMA protocol, which claims to give near optimal TDMA schedule for given network.

softMAC[1] 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 disbale CSMA/CA including MAC ACK. 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 archive it.

MadMAC[2] 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 to do so. But our frame structure might be completely different from MadMAC to implement multi-hop system.

On other hand, MultiMAC[3] , which also extends softMAC but uses multiple MAC protocol. A special marker in each packet identifies the protocol to use to decode the packet and passes to the network stack. But this might incur some 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. Initially we are more focused on implementing TDMA system. We might consider to include CSMA/CA for short distance provided delay and throughput are within permissible bound.

FreeMAC[4] goes one step ahead and provide multichannel communication. It also uses hardware beacon interrupt timer in place of software kernel timer for fine control over packet transmission timing. It also 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[6], which uses TDMA with bulk ACK and FEC type loss recovery method. Unlike softMAC, the WiLDNet uses 802.11 frame format. It uses a) click[8] module on top of MAC layer to implement bulk ACK and FEC mechanism and b) does some MAC layer modification. Our system will be fundametally different from WiLDNet but this paper does give some insight in to 802.11 poor performance in long distance link. Section 4 explain it in more detail.

Overlay Layer[5] 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 can use 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 to do so. In section 4 we give explanation of choosing MAC-level modification over overlay layer.

Now moving on to TDMA schedule, Spatial TDMA proposed by Nelson *et al.*[7] gives algorithm to design collision free near optimum TDMA schedule. Idea is to form a max clique from given adjacency matrix of link. In our case, we are considering 2 hop network and moreover network is tree like structure which might help in reducing the complexity of this problem.

4 Our Approach

In this section we study the limitation of 802.11 standard as explain in[6]. Below are the three main reason for unpredictable behavior of 802.11 for long distance links,

- Inefficient link level recovery
- Collision on long distance link
- Multiple link interference

Patra *et al.*[6] also suggest that modifying driver-level parameter will not suffice to improve the performance. WiLDNet[6] uses click[8] module on top of MAC layer which might increase complexity and may incur extra packet delay.

Given the failure of CSMA/CA and stop-and-wait type MAC level ACK mechanism for long distance links and resisting ourself from using click software, we decided to use TDMA type system. Now we explain reason of choosing TDMA-MAC type implementation over overlay layer.

In section 3, we looked at different TDMA-MAC systems and one overlay type implementation. The distinction between between them is in the level of functionality they offers. In TDMA-MAC user get direct access to MAC layer and can control individual packet transmission, while in case of overlay, user can only work with packet buffers and have no control over packet timing. Also overlay works on top of MAC, so it can work only with interface provided by MAC layer. In case of TDMA-MAC, it get direct access to hardware through HAL giving better control over actual packet.

Our objective was to implement TDMA system on easily available commodity hardware with open source madwifi 0.9.4[9] wireless driver. Madwifi works on top proprietary HAL provided by Atheros. We consider every node in TDMA system is a Soekris[10] board

with Linux running on it. Madwifi is written in C language, so one can easily modify Madwifi code to suite its requirement. Since HAL is in binary, it prevents us from accidently modifying any proprietary firmware code. Purpose of HAL is to provide interface to madwifi driver to access actual hardware with some restriction.

After considering pros and cons of both method, we decided to use TDMA-MAC type implementation. This means we are now going to work with madwifi code in rest of the project. Now we will look at envisioned architecture of proposed project. In Fig. 1, central node is the master node and also the internet gateway. The node inside the sector antennas range are local wifi devices. They communicate with local gateway marked with red color to send their traffic to Master node. All node will run modified TDMA code.

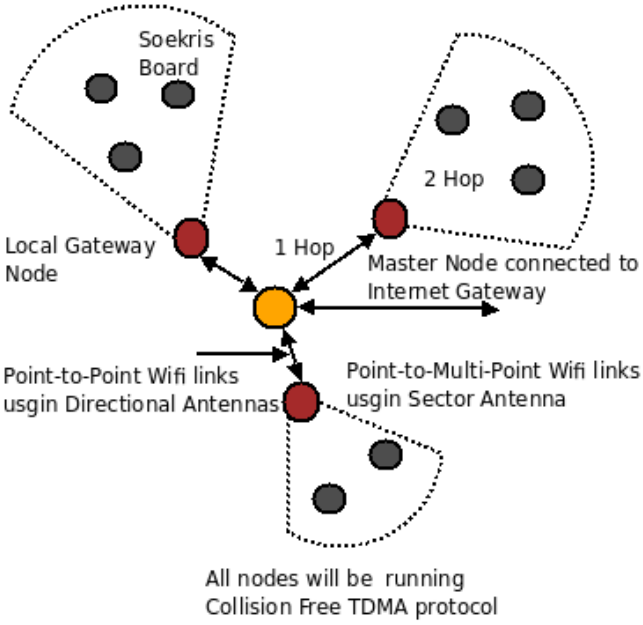

Figure 1: Multi-Hop TDMA Architecture

The node in sector antenna is soekris machine running linux. Soekris machine has eth0 and wifi0 interface. In our proposed system, device must run modified TDMA code to talk to other devices. It might be the case that end user might not have wifi interface. To facilitate non wifi device, we can configure routing entry in soekris board and make it as gateway for non wifi devices. Now all non-wifi device can be connected to ethernet port of soekris. Data from such devices will get buffered in soekris and will get transmitted during its transmission slot using wifi interface.

All TDMA-MAC approach discussed in section 3 are designed for single hop network and all uses static TDMA schedule. In contrast, in our implementation we are planning to address following issues,

- Implementing Muli-Hop TDMA functionality
- Making custom frame format
- Intelligent TDMA schedule dissemination

Before going in to our implementation of TDMA system lets look at the performance of a live system which uses single hop TDMA type mechanism. This system was setup in mountain region of Venezuela[11] and it uses WiLDNet[6] framework. They have establish single-hop link with distance greater than 300 kilometers. They have use directional antennas for point to point link. According to them they were able to achieve sustained throughput of 3 mbps, which is sufficient for handling audio/video and web traffic.

Another example is Arvind Eye Hospital[12] in India, which uses WiLDNet framework to enable video conferencing between doctor and patient.

Our goal is to get such performance for multi-hope TDMA system with multiple device competing for resource. To start with, we have implement prototype TDMA system between two node which is explained in detail, in section 5.

5 Work Done

This section describe initial tasks carried out in order to implement prototype TDMA system between two devices. Here we assume to have loose time synchronization and we use software kernel timer to trigger various action.

This prototype TDMA system has helped us in understanding madwifi driver in better way. Madwifi uses various structure to store device(dev), packet(skb) and software(sc) related configuration for physical device. It uses tasklet mechanism to process various interrupt it receives. *ath_intr()* handles all interrupt and returns handle to a function for each interrupt. More precisely, it calls *rx_tasklet()* routine when it receive data through interrupt. One has to initialize tasklet in *ath_attach()* using ATH_INIT_TQUEUE and schedule it in *ath_intr()* using ATH_SCHEDULE_TQUEUE. The tasklet has all necessary information such as which function to call and what data to pass to it in event of interrupt. In case of madwifi driver data passed is dev structure.

5.1 General Setup

To start with, we use small-form factor soekris board having mini-pci wifi card. It has AR5212 Atheros

chipset which works well with madwifi 0.9.4. Our first step was to setup soekris board with Linux and to install madwifi driver on it. We use Voyage Linux with 2.6.25.4 kernel and madwifi 0.9.4. Installing voyage Linux was bit tricky, but one can find detail documentation on web. Voyage linux takes around 80 MB and gives full functionality as of any Linux distribution. Next step was to integrate readlog functionality in madwifi 0.9.4. Readlog was implemented by undergraduate student at IIT Kanpur and it gives us way to sniff packet with detail per packet information. Since voyage linux has shell interface, this tool comes handy for monitoring wifi traffic. The original readlog was implemented in madwifi 0.9.3 and making it work in 0.9.4 was straight forward. At this stage we had

1. Seokris board with Voyage Linux and 2.6.25.4 kernel
2. Madwifi 0.9.4 with readlog functionality

Madwifi provides five mode of operations and one of them is monitor mode. When card is set in monitor mode, one can sniff the channel using sniffer software like wireshark. Since voyage linux runs at run level 3, readlog is used to dump the traffic. When we use readlog, we found there were large number of entry with *packet_type = 0, packet_sub_type = 0* and *len =0*. This specific packet_type / packet_sub_type pair was not fitting in any 802.11 packet specification. So we decided to modify readlog to discard such entry from appearing in sniffed log.

We then tested with various operation mode and found Ahdemo mode was crashing intermittently. At this stage we know

1. How to create VAP for adhoc,monitor,ahdemo, station and AP mode
2. How to setup card in adhoc and station-AP mode
3. How to use wlanconfig, iwlist, iwconfig
4. Verified communication with *ping* and *scp* for above modes

Next we tried to disable immediate MAC level ACK. We got some pointers from madwifi mailing list and made some changes in code including writing specific value in register. To verify it, we monitored traffic between two soekris running modified code and found there were no packet with *packet_type = 1* and *packet_subtype = 13*. Which means ACK packets were not generated. Once MAC layer ACK are disabled, it is up to the upper layers to take care of error. At this stage we decided to keep MAC layer ACK disabled for all future experiments.

Now our next step was to disable beacon. In adhoc mode, we suspect that beacon are generated from hardware and we were enable to find place to disable it. Though it calls *ath_beacon_start_adhoc()* but then is never comes to it again. At this stage we don't know the path from where beacons are generated in adhoc mode. While in station and AP mode, we modified *ath_beacon_start_adhoc()* to return immediately and we where able to disable beacons. We verified it with monitor mode and readlog. We then tried to disable backoff by forcing *cw_min* and *cw_max* to zero and setting HAL_TXQ_BACKOFF_DISABLE flag. After detail experimentation we were not sure of whether backoff was indeed disabled.

After lot of experimentation and unsuccessful attempt to disable beacon transmission and backoff in adhoc mode, we shift our focus on implementing prototype TDMA. We were looking at TDMA type communication between two device. We had two mode to choose from for TDMA implementation. We decided to use Adhoc mode over monitor mode. Advantage of monitor mode was it disables RTS/CTS mechanism and MAC level ACK's (which we already disabled) but on other hand it generate RAW packets. We tried to generate our own packet format, but once packet reaches *ath_hal_txstart*, it stamps hardware sequence number at byte number 23 and 24. This behavior was corrupting our data payload. We did tried to copy those two byte in next location. Also in Adhoc mode byte number 31 and 32 are padded and are removed on receiver side, same thing happens for Monitor mode. In Adhoc mode, driver take care of it but in monitor mode byte 31 and 32 are data payload and it get deleted. To avoid it, we again copied this byte in next location. After doing that, we where able to go past ARP request, but somehow every time destination MAC address got corrupted. Due to that, during ping test, ARP was going through but ping reply had wrong MAC address. Destination node was sending ping reply to corrupted MAC. This was again a frustrating result for us. Though we were close to solve this problem we were keen on doing TDMA MAC. So finally we decided to implement TDMA in Adhoc mode keeping standard 802.11 frame format with beacon and RTS/CTS mechanism enabled.

5.2 Prototype TDMA

The aim was to build TDMA system between two device with loose time synchronization. The setup is as follows. Initially both device will use CSMA for communication. At some time (by running UDP program in user space on node having ID = 0) the

master node will send magic packet with six A's and will start it's TDMA timer which expires at every SLOT_INTERVAL + node_id * 100. Were SLOT_INTERVAL is 1000 msec for example. On timer expiration it will call *TDMA_send_triggered()* routine which again activate timer for next slot and send data on air from queue one packet at a timer for specified slot length i.e 100 msec or empty queue condition, whichever occur first. Here queue is a link list where all incoming data from network layer is buffered till next transmission slot.

Once the allocated time slot for node is over, it will wait for next timer expiration. In this way every node will send data in specific time slot. Below figure explain the working of prototype TDMA. The TDMA slot will overlap with each other on long run due loose synchronization and clock drift. We have plan to address this issue in second stage of project.

Now we will look at the call sequence in Adhoc mode for transmitting and receiving data. From fig. 2, on transmit side we modified *ath_tx_txqaddbuf()* to buffer data coming from upper layer. We delay call to *ath_hal_txstart()*, which is the place from where packets are actually transmitted on air, till *TDMA_send_triggered()* is schedule by timer. It is the job of *TDMA_send_triggered()* to check for slot boundary and send packet from queue by calling *ath_hal_txstart()*. On receiving side, we modified *rx_tasklet()* to start TDMA timer up on receiving magic packet and on timer expiration it calls *TDMA_send_triggered()* to send packet. *TDMA_send_triggered()* adds timer every time it is called to mark next TDMA cycle. Purpose of this setup was to find places to hookup TDMA code in madwifi. Synchronization and multi-hop TDMA are immediate goal in second stage.

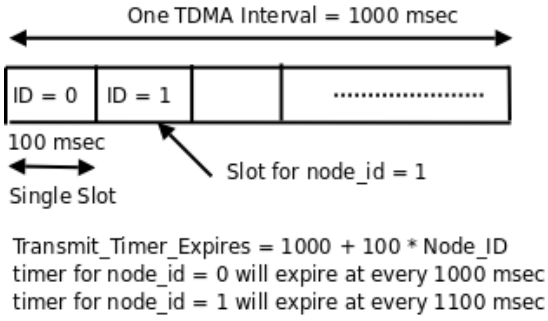

Figure 2: TDMA Cycle

Also, nodes will receive data when they are not in transmitting state. We have disable HAL_INT_RX flag just before transmission and re-enabled it after transmission. This prevents nodes from going in to rx_tasklet() routine in event of receiving HAL_INT_RX interrupt while they are transmitting. As said earlier, it has loose time synchronization, right now we are not in position to comment on overall behavior of prototype TDMA system. But we had successfully tested ping command and it was giving around 5% packet loss.

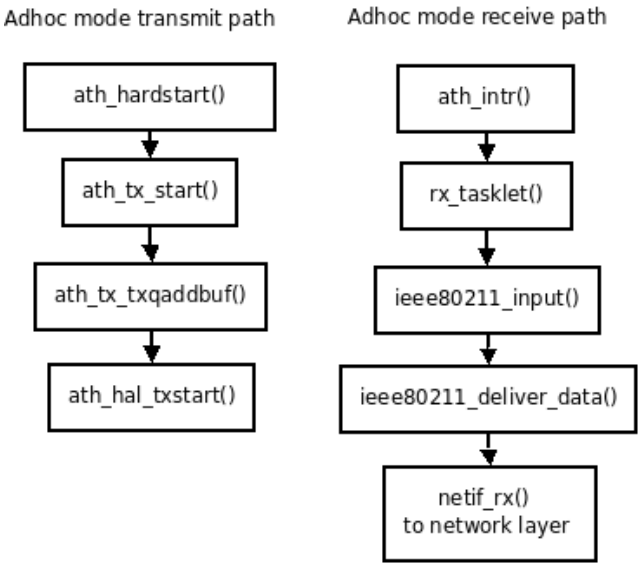

Figure 3: Transmit and Receive Path in Adhoc mode

We also created new athdebug mode named TDMA, which can be used to print TDMA related output. This might come handy for second stage.

6 Stage Two Timeline

In stage one we were able to accomplish following things

1. Understanding Madwifi driver basics
2. Disabling MAC layer ACK
3. Disabling beacons in Station-AP mode
4. Understanding transmit and receive path for Monitor and Adhoc mode
5. Using RAW packet of monitor mode and changing it's content
6. Implementing kernel timer for TDMA operation

and we were able to implement simple TDMA system for two devices. This was a first step in building fully functional multi-hop TDMA system.

In second stage we are going to explore monitor mode. More precisely we want current Adhoc mode TDMA to work in Monitor mode. Advantage of using monitor mode is it allows RAW packet to be send to / from MAC layer. Using this, one can send custom frame format to destination. This is crucial for TDMA working. Since, it is most likely that we are going to use our own MAC frame format for TDMA working, which might include synchronization information with guard time to name few. As of now we haven't come up custom frame format.

We will address following action items in stage two,

- TDMA in monitor mode
- Finalizing TDMA frame structure
- TDMA schedule dissemination

Then we are planning to extend it to multi-hop network. This is the major goal of second stage. Bellow is the task list given in softMAC[1] to disable CSMA/CA in madwifi.

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)

When we put card in monitor mode it achieves first three by default. We did try to disable Beacon and Backoff but were not sure of it's working. We will work on it in second stage. We also need to come up with collision free near optimum TDMA schedule algorithm for static network. Later we might extend it for dynamic workload.

References

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

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

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

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

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

[6] 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

[7] Nelson, R., Kleinrock, L. Spatial TDMA - A collision-free multihop channel access protocol, IEEE Transactions on Communications, vol. COM-33, Sept. 1985, p. 934-944

[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/

[11] www.eslared.org.ve/articulos/Long Distance WiFi Trial.pdf

[12] http://tier.cs.berkeley.edu/wiki/Aravind