PER NODE THROUGHPUT FAIRNESS IN A SINGLE BRANCH OF A MESH NETWORK

A Thesis by

Jana Grosboll

Bachelor of Science in Computer Engineering, Wichita State University, 2002

Submitted to the Department of Electrical Engineering and the faculty of the Graduate School of Wichita State University in partial fulfillment of the requirements for the degree of Master of Science

May 2007
© Copyright 2007 by Jana Grosboll

All Rights Reserved
I have examined the final copy of this Thesis for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

Coskun Cetinkaya, Committee Chair

We have read this Thesis and recommend its acceptance:

Kamesh Namuduri, Committee Member

Mehmet B. Yildirim, Committee Member
DEDICATION

To my parents, John and Evelyn for always being there for me, and encouraging me in my endeavors, my close friends for support and understanding, and above all to God for the gift of strength and life
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Coskun Cetinkaya, for his patience in working with me. I could not have done it without his expert work, advice, and time. Much appreciated!
ABSTRACT

It is important to provide throughput fairness in Wireless Mesh Networks (WMNs) so that each node has a fair chance at sending its packets through the network. Existing protocols for WMN’s do not provide throughput fairness for nodes that are more than one hop away from the gateway. In some cases, nodes that are further than one hop away from the gateway experience throughput starvation especially is this the case when the network load is increased [5], [7]. The purpose of this thesis is to simulate a new Mesh Fairness Algorithm (MFA) for a single branch of a mesh network and show that the MFA gives a much greater fairness as far as throughput for each node. In this MFA, the packet queue will be modified and the backoff counter in each node will be changed based on the actions and locations of other nodes in the network which will result in greater throughput fairness.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Wireless Mesh Networks (WMNs)</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Fairness Problem to Solve with WMNs</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Goals of this Thesis</td>
<td>3</td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>LITERATURE REVIEW</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Mesh Network Terms</td>
<td>5</td>
</tr>
<tr>
<td>2.2 IEEE 802.11 Medium Access Control (MAC) Protocol</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Cooperative Medium Access Control (C-MAC) Protocol</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Packet Queues</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Previous work on Wireless Mesh Networks</td>
<td>10</td>
</tr>
<tr>
<td>2.6 Summary</td>
<td>11</td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>MESH FAIRNESS ALGORITHM</td>
<td>12</td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>TESTING PROCEDURES AND IMPLEMENTATION</td>
<td>16</td>
</tr>
<tr>
<td>4.1 Simulator</td>
<td>16</td>
</tr>
<tr>
<td>4.2 Configuration</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Assumptions and Calculations</td>
<td>18</td>
</tr>
<tr>
<td>4.4 Mesh Fairness Code</td>
<td>18</td>
</tr>
<tr>
<td>4.4.1 Code Manipulations for the IEEE 802.11 MAC Protocol</td>
<td>18</td>
</tr>
<tr>
<td>4.4.2 Code Manipulations for the Queue</td>
<td>23</td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>RESULTS</td>
<td>27</td>
</tr>
<tr>
<td>6.</td>
<td></td>
</tr>
<tr>
<td>DISCUSSION AND CONCLUSION</td>
<td>33</td>
</tr>
<tr>
<td>6.1 Expected Results</td>
<td>33</td>
</tr>
<tr>
<td>6.2 Actual Results</td>
<td>33</td>
</tr>
<tr>
<td>6.3 Conclusion</td>
<td>34</td>
</tr>
<tr>
<td>6.4 Future Work</td>
<td>34</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>35</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

In recent years there has been an explosive increase in technology. Palm pilots, computers, Internet, Global Positioning System, and cell phones have become common household words. With this increase in technology has also come wireless networks which have drastically changed our world. We now have the freedom to connect to the Internet almost anywhere and anytime without the use of a wired link. Wireless networks, as the word implies, do not contain a physical medium of connect such as wired networks do. Many of the protocols in wireless networks have been taken straight from the protocols used with wired networks with some modifications to make them work with wireless networks.

1.1 Wireless Mesh Networks (WMNs)

Wireless Mesh Networks (WMNs) have received attention since they are able to provide reliable and robust wireless broadband service accessibility. WMN’s are predicted to solve some of the limitations and improve performance of other wireless networking methods such as adhoc networks, wireless local area networks, and wireless personal area networks [2]. In building a WMN, there is not a huge outlay of means and they can be expanded easily which adds to their popularity. Some applications may include medical systems, military operations, surveillance, emergency disaster areas, outer space, and wireless broadband service access.

A mesh network is a local area network (LAN) that allows for continuous connections and dynamic reconfiguration if a path breaks. WMN are part of the distributed wireless networks which are wireless nodes communicating with each other.
without any pre-existing infrastructure in place. There is no central administration so the network does not crash when one node goes down, other nodes just take over for that node [15]. Mesh networking is a sub category of ad hoc networking and the main difference between ad hoc networks and WMN is the traffic pattern. In mesh networks almost all traffic flows to and from an Internet-connected gateway [5], whereas in an ad hoc network traffic flows randomly between different pairs of nodes. The nodes in a WMN maintain and establish their own routes. Packets reach their destination by “hopping” from node to node meaning that each node is not only a host but also acts as a router. Even though all the protocols are in place for WMNs using the existing protocols from ad hoc networks, more work needs to be done on the protocols to enable them to work more efficiently on a WMN and throughput is not degraded by multi-hop forwarding and hidden terminals.

1.2 Fairness Problem to Solve with WMNs

Throughput fairness is an important aspect of a network. Throughput has to do with the fraction of the channel capacity used for data transmission [15]. Throughput per node is a comparison of the throughput of each node compared to every other node. Per node fairness in a MAC protocol is defined as not showing preference for a node when multiple nodes are contending for the channel [15]. In using the term throughput fairness it is meant that each node sends the same amount of packets through the network.

In WMNs each node must act as a router as well a node. It will be sending its own packets as well as packets that are forwarded to it by other nodes. All packets have to be forwarded between the clients and the gateway. Consider the “parking lot scenario” (Figure 1) where there is a network that is in a straight line starting with node
two then node one and lastly the gateway. Node one can completely starve node two by simply sending its own packets and not forwarding node number two’s packets. What is meant by starve is that node two will not be able to send any data since node one is sending all its own data to the gateway, therefore node two cannot get any data to the gateway. This problem is compounded as more nodes are added to the network that are more than one hop away from the gateway and as the network load increases [7].

The Medium Access Control (MAC) protocol contributes negatively to the fairness problem in WMNs, and MAC protocol determines when a node can send packets. Since data in a WMN must traverse multiple hops to reach the gateway, data must contend for access to the medium at each intermediate hop. This means that standard MAC protocols cannot provide fairness to each node in the network [5], [2].

![Figure 1: Parking Lot Scenario](image)

### 1.3 Goals of this Thesis

The goal of this thesis is to improve throughput fairness per node in a single branch of a wireless mesh network. This research focuses on the Mesh Fairness Algorithm (MFA) for a mesh network that uses the Cooperative Medium Access Control (C-MAC) protocol.
The following is what is to be accomplished:

1. Improve the fairness of a single branch of the a WMN using simulation by:
   a. Modifying the C-MAC protocol to identify who a packet belongs too in order to
      improve throughput fairness for packets that do not belong to the current
      node. The backoff counter is to be modified based on how far a node is from
      the gateway and what node sent the packet.
   b. Modifying the queue in order to improve throughput fairness among packets
      by giving a higher priority to packets that do not belong to the current node.
   c. Show by simulation that this has been accomplished.

2. Comparison of the MFA to the existing protocol IEEE 802.11.

3. Conclusions from the results and future work that could be done.

The rest of the paper is organized as follows: Chapter 2 is a review of literature
on wireless networking MAC protocols as well as wireless mesh networks. Chapter 3
describes the Mesh Fairness Algorithm proposed in this paper and how it works with the
MAC protocol. The testing methods of the Mesh Fairness Algorithm and setup are
described in chapter 4 and the results of simulation are noted in chapter 5. In chapter 6,
the author discusses the results of the simulation, what conclusions can be drawn from
the results, and finally what future work needs to be done.
CHAPTER 2
LITERATURE REVIEW

Research is constantly seeking ways to improve Mesh Networks and the protocols used in order to have more efficient and effective networks. In this section the author looks at some of the work that has been done with wireless networks especially MAC protocols to better understand mesh networks, throughput, and fairness.

2.1 Mesh Network Terms

First, some of the terms that are used for Mesh Networks must be defined. A gateway is a connection to the Internet and in a mesh network, all the traffic flows to and from the gateway [5]. A mesh network has two topology connections--full mesh topology and partial mesh topology. In full mesh topology all the nodes are directly connected to each other. With partial mesh topology nodes are only directly connected with some other nodes but not directly connected with all the nodes. Partial mesh networks is the most practical and most used.

Mesh Networks are normally located in a fixed location such as rooftops, therefore the topology is rather static. Topology changes are mostly based on the addition of more mesh nodes. Since mesh nodes are mostly fixed in location, they can be plugged into the power grid eliminating the need to use batteries. The traffic pattern is generally much the same as mentioned previously.

2.2 IEEE 802.11 Medium Access Control (MAC) Protocol

Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) is the MAC protocol on which the IEEE 802.11 MAC is based. The MAC protocol is used with either a basic access method or with a four-way hand-shake method. The four-way hand-shake method uses two extra packets---request-to-send (RTS) and clear-to-send
These packets let the sender reserve bandwidth for transmission of its' packet. The four-way hand-shake also minimizes the bandwidth wasted when there is a collision and this is illustrated in Figure 2.

When a node has a packet to send it first senses the channel. If the channel is idle for Distributed Interframe Spacing (DIFS), the node will send a packet. When the channel is busy, the node waits until the channel is idle for DIFS amount of time, after which the node generates a random backoff counter uniformly from the range of \([0, CW - 1]\), \(CW\) is the contention window size. When a node transmits or attempts to transmit a packet, the \(CW\) is set back to \(CW_{\text{min}}\) which is the minimum contention window size. As long as the channel is idle, the backoff counter is decremented.

When the channel becomes busy, the backoff counter is paused until the channel is idle again. Once the backoff counter reaches zero, the node will send a RTS packet for the four-way hand-shake. The receiving node will respond with a CTS packet after Short InterFrame Spacing (SIFS) amount of time. All other nodes hearing the RTS and CTS will defer their transmissions and set their network allocation vector (NAV) based on information contained in the RTS and CTS packets. When the sending node receives the CTS packet, it responds by sending the actual data packet after SIFS amount of time has elapsed. The sending node then waits for the receiving node to send the acknowledgement (ACK) packet. The sender will go into exponential backoff procedure when a collision has happened. If the sender does not hear the CTS or ACK (CTS if using the four-way hand-shake and ACK if using the basic access method) packet, it assumes that a collision has occurred and goes into exponential backoff. A collision usually occurs because either a node believed the channel to be idle when it was actually busy, (called the "hidden terminal" problem), or multiple nodes sensed the
channel idle and started transmitting at the same time. Exponential Backoff is where the CW value is doubled after each unsuccessful transmission until $CW_{\text{max}}$ is reached [4].

Figure 2: IEEE 802.11 Four-way Hand-Shake

2.3 Cooperative Medium Access Control (C-MAC) Protocol

The C-MAC protocol [3] is based on the MAC 802.11 protocol in which the throughput depends on three main things: 1.) successful transmissions, 2.) collisions, and 3.) idle slots due to backoff at the contention periods. Throughput is a measure of the amount of bits sent to a node on a communication link divided by the amount of time in which they were sent usually measured in bits per second (bps). To reduce time wasted in idle slots collision resolution needs to be expedited. In the C-MAC protocol, idle slots are reduced through means of a constant window size. Collisions among collided nodes are resolved more quickly by assigning the collided nodes a shorter Distributed Inter-Frame Space (DIFS) than the other nodes.
When a node has a packet to send, it first senses the channel. If the channel is idle for DIFS (110 μsec in C-MAC) amount of time, the node will send the packet. If the channel is busy, the node will defer sending the packet until the channel is idle for DIFS amount of time. After DIFS amount of time, the node will generate a random backoff counter chosen from the range of \([0, CW -1]\), where \(CW\) is the contention window size. The backoff counter then starts decrementing as long as the channel is sensed to be idle. If the channel becomes busy, the backoff counter is stopped until the channel is idle again for DIFS amount of time. Once the backoff counter reaches zero, the node sends the request-to-send (RTS) packet if it is using the four way handshake the same as in the IEEE 802.11 protocol. When the packet is received by the receiving node, the receiving node waits Short Inter-Frame Space (SIFS, 10 μsec) amount of time and then sends the clear-to-send (CTS) packet. Any nodes that have heard the RTS and CTS packets will have sent their network allocation vector (NAV) so they will wait to send their packets. When the sending node receives the CTS packet, it responds by sending its data packet. When the receiving node receives the data packet, it responds by sending an acknowledgment (ACK) packet. If the sending node’s packet was successfully transmitted, it will then choose a new backoff counter uniformly from the range of \([CW, 2 * CW – 1]\) where \(CW\) is the contention window. The reason for the different range for the backoff counter of a node that has successfully transmitted is to give other nodes that have been waiting a higher chance of capturing the channel and sending their packets. If the sending node does not receive a CTS or ACK packet it will assume that a collision has occurred.

When a collision occurs, the sending node generates a new backoff counter based uniformly on the range of \([0, 3]\). It then senses the channel, and if the channel is
idle for a duration of Priority Inter-Frame Space (PIFS, 30μsec) instead of the normal DIFS amount of time, the node starts decrementing its backoff counter. DIFS is equal to PIFS plus four times the SlotTime (20μsec). Since the collided nodes have a shorter range from which their backoff counter is chosen, they have a higher priority for packet transmission than the other nodes. If there is another collision on the channel before the collided node’s backoff counter reaches zero, then the sending node sets its backoff counter to zero and waits for the channel to become idle for DIFS amount of time after which transmission will occur. Thus the collided node has a higher priority than the nodes not involved in the collision. Nodes that do not receive a RTS or CTS correctly sense that the channel is busy since the interference power received is higher than the floor noise. Then the node assumes that a collision has occurred and sets its NAV to Extended Inter-Frame Space (EIFS). When the node has successfully transmitted a packet it will choose a new backoff counter uniformly from the range of \([CW, 2 * CW – 1]\).

2.4 Packet Queues

A packet queue is a structure to store packets that are to be sent. Queues are important in WMNs since a node in a WMN must act as a router. Since a node must be a router it must have a queue to store packets that it receives. The most well known queue which is also used in the IEEE 802.11 is the First-In-First-Out (FIFO) queue also known as the drop-tail queue. In a FIFO queue the first element in the queue will be the first one out. When an element (in this paper packets) is added to the queue it is added to the end of the queue, so the last added element will be the last out. A queue has two basic functions to enqueue elements which is adding elements to the queue and dequeue which is taking elements out of the queue to be processed. A queue is of
limited capacity so it must of necessity drop some packets when it is full. A queue can be configured to drop packets from either the head (the beginning) or the tail (the end). With drop-tail as the name implies they are dropped from the tail. The queue size is important since this determines how many packets the queue can hold, on the other hand we cannot have an infinite queue size. Overflow happens when trying to add packets to an already full queue. The packet queue affects the throughput for a couple of reasons. One, if a packet gets dropped at a queue before it reaches the destination then obviously that is one less packet that did not make it through. Secondly, if some packets need a higher priority than others and these packets get stuck at the end of the queue not only will a lot of packets be going before them, but the packets have a higher probability of being dropped.

A priority queue gives certain packets a higher priority than all other packets. A simple priority queue can be implemented by using a FIFO queue and inserting high priority packets at the head of the queue instead of the tail and all other packets are inserted at the tail.

2.5 Previous Work on Wireless Mesh Networks

Fairness and throughput have been a topic of research in wireless networks. Many methods have been proposed to create more fairness and better throughput. Among them have been proposed label switching for efficient packet forwarding [18], coordinating efforts between the MAC layer and packet scheduling algorithms [19], [16], methods for modifying the MAC protocol for faster collision resolution [14], and many other variations and methods [11], [13], [20].

A new MAC protocol with fast collision resolution is proposed in [14]. It is stated in this paper that in order to have an efficient MAC protocol, that protocol must provide
high throughput performance and fairness. The new Fast Collision Resolution (FCR) MAC protocol for wireless networks focuses on reducing collision resolution time by modifying the backoff counter on active nodes. The FCR also reduces idle slots by using smaller contention window sizes for nodes that have transmitted successfully. Two very important points are made in this paper that to have better throughput 1. collision resolution needs to be faster and 2. the amount of idle slots needs to be reduced. These points have been taken into consideration in the C-MAC protocol in the following MFA that is proposed in this paper.

2.6 Summary

In this section mesh networking basics were discussed. The C-MAC protocol is summarized and used in conjunction with the MFA that is proposed in this paper. Also discussed are queues which play an important factor in how many and when a packet gets through the network. To understand MFA it is therefore important to understand mesh networks, queues, and MAC protocols.
CHAPTER 3
MESH FAIRNESS ALGORITHM

In this thesis, a solution for per node throughput fairness in Mesh Networks is built on a mesh fairness algorithm along with the use of the underlying MAC protocol, Cooperative MAC (C-MAC). The C-MAC protocol provides pro flow throughput fairness in a wireless (Local Area Network) LAN, but cannot provide per node throughput fairness in a WMN as is seen in chapter 5, Figure 12. Therefore the MFA protocol is proposed to provide per node throughput fairness and be used in conjunction with the C-MAC protocol as will be shown in chapter 5 Figure 10. The idea behind the Mesh Fairness Algorithm (MFA) is to achieve per node throughput fairness in a single branch of a Mesh Network. In the following section the MFA Protocol will be described. With this algorithm, medium access is coordinated among nodes to reduce medium contention as well as giving priority to packets from other nodes. In order to do this, nodes adjust their backoff counter based on the activities of other nodes.

Each node inserts its own packets last at the queue and other node’s packets first at the MAC queue. The MAC queue is basically the lineup of packets to be sent stored by the node in the order that they need to be sent. This gives other nodes a higher priority to help prevent a node near the gateway from starving other nodes that are further away. By giving arriving packets from multi-hop nodes a higher priority, bandwidth is actually being saved. Since multi-hop nodes have to send their packets through several hops, if their packet is dropped somewhere along these hops the packet has to be sent again wasting time and bandwidth. By inserting a multi-hop node’s packet at the beginning or head of the queue we basically make sure it will not get dropped when the packet queue gets full since the queue drops packets from the tail.
and not the head. When a packet is successfully transmitted, the MAC layer informs the transport layer to get the next packet. Then, nodes are differentiated based on their hop count away from the gateway, i.e. their distance from the gateway. There are two types of differentiation: single-hop and multi-hop nodes. Packets are distinguished based on whether they are the node’s own packet or a packet which the node is forwarding. The buffer scheduling used in the MFA is Last-In-First-Out (LIFO). The queue buffer has also been increased to help accommodate more packets. Buffer management at the intermediate nodes is important so that the multi-hop flow performance is not degraded.

If a node that is a single-hop away from the gateway has a packet to transmit, a random backoff counter is chosen uniformly from the range of $[2 \times CW, 3 \times CW - 1]$ for its own packets and from the range of $[CW, 2 \times CW - 1]$ for forwarding packets instead of the normal C-MAC backoff range. For a multi-hop node, the range from which the backoff counter is chosen for its own packets is $[CW, 2 \times CW - 1]$ and for forwarding packets, the range for the backoff counter is $[0, CW - 1]$. This gives the multi-hop nodes a higher priority to access the channel than the single-hop nodes and helps to solve the problem of a single hop node starving multi-hop nodes creating unfair throughput. Throughput for multi-hop nodes is thus increased.

Since this method reduces the throughput for single-hop nodes, there needs to be a way to limit the data rate of multi-hop nodes. To accomplish this, a multi-hop node is limited from contending for the channel if it has a packet of its own at the head of the packet queue. This is the situation unless the multi-hop node’s parent transmits the parent’s own packet. The multi-hop node then waits a packet transmission time which is RTS plus CTS plus DATA plus ACK so that the grandparent can transmit the parent’s packet. After this time period, the backoff counter continues to work as described.
above. When a node receives a packet, it resets it’s backoff counter since the packet is inserted at the head of the packet queue. This additional criteria limits the data rate of multi-hop nodes thus making for throughput fairness for a single branch of a wireless mesh network.

It is assumed that a node always has a packet to send, but what if this is not true? If a node has a packet to send but its parent node does not have a packet to send, the node could be waiting forever for the parent node to send a packet and never get to send its own packet. One solution is if a node does not have a packet to transmit it inserts a dummy packet in the queue. Then if a node really has a packet to send before the dummy packet is sent, the dummy packet can be replaced by the new packet. If this does not happen, the dummy packet is transmitted without the RTS or CTS part of the four-way hand-shake. The dummy packet does not contain any data so its size is a very small 28 bytes which is the size of the MAC header. For the children nodes to be activated, a parent must transmit a dummy packet which will not be put in the parent’s queue. The parent will reset its backoff counter when it receives a new packet. Another solution would be to have a timer at the child node so that if the parent does not send a packet for a certain amount of time the child will then be allowed to send a packet.

Some initial C programming code shows that the MFA will achieve a per node throughput fairness. The overall throughput is less than the IEEE 802.11 MAC, but this is because this protocol does not achieve per node throughput fairness. Rather node 1 is sending all of the packets and starving the other nodes. The results of the initial programming code are show in Figure 3.
The MFA algorithm does not add any extra information to the packets on the network neither does it add extra overhead packets. This is the valuable feature of the MFA when compared with other methods for making WMNs fair. The MFA simply monitors the network traffic and based on the traffic makes decisions about how to handle it.
CHAPTER 4
TESTING PROCEDURES AND IMPLEMENTATION

4.1 Simulator

To test the Mesh Fairness Algorithm (MFA) with the modified C-MAC protocol, the ns-2 simulator version 2.27 was used [6]. The ns-2 simulator was created at the University of California Berkley in the 1990’s and continues to be used and revised. In the simulator a combination of the C programming language with an OTcl frontend interpreter is used. The IEEE 802.11 MAC protocol was modified to the specifications of the C-MAC protocol. The MFA protocol was implemented by modifying the existing code for the MAC protocol and MAC queue, by differentiating nodes based on their hop count from the gateway and by differentiating packets based on their source. In this section the simulation setup and procedures are discussed.

4.2 Configuration

In the ns-2 simulator a default configuration file called ns-default.tcl is used for setting default values for variables used in the simulation. To begin, CWmin is set to 4 and Cwmax to 16. Simulations are run for 30 seconds and data is taken from 15 seconds to 45 seconds. We chose to start collecting data at 15 seconds since by that time the network should have completed all the initial setup and the traffic should have settled down to a normal flow. The link capacity has been set to 1 Megabytes per second (Mbps). The IEEE 802.11 four-way hand-shake is used in all simulations.

The network has been set up using three nodes in a row to simulate a single branch of a WMN. A node can only send for a distance 250 meters in our simulation. Each node is spaced 200 meters apart on the x axis and fixed distance of 400 meters on the y axis so that the nodes are in a row for a branch of a WMN. Since node 1 is 400
(it is 200 meters to node 2 and another 200 meters to node 1 for a total of 400 meters) meters from node 3, node 3 can only reach node 2 with packets. All packets from node 3 destined for the gateway must pass through node 2 to node 1 and then to the gateway. Node 2 is far enough from the gateway that it cannot send traffic straight to the gateway but must send traffic through node 1 and the same is true for all nodes except node 1 which is only one hop from the gateway. Each node generates CBR (Constant Bit Rate) traffic destined for the gateway. The packet interval is set to half a second and packet size is set to 1000 bytes. This size of 1000 bytes is about the max packet size that can be used before packets start getting fragmented because they are too large. The data rate at which the packets are sent is in kilobytes per second (kbps) and the values used range from between 25 to 500. Node 0 was configured as the gateway for the network. The routing protocol that we are using in the simulation is Destination-Sequenced Distance Vector (DSDV) with each node stationary during the simulation. The ShortRetryLimit_ for the MAC protocol has been changed to 20 which is the number of retransmissions for a packet. DIFS has been set to 110μsec, SIFS to 10 μsec, and PIFS to 30μsec. Other variables are left at the default values in the simulation for a wireless network.

For the queuing, the drop-tail queue was used along with priority queuing. The queue buffer size was increased from the default of 50 to 250, enabling the queue to hold more packets and to accommodate the changes that have been made to the queue.
4.3 Assumptions and Calculations

We assume that each node on the network always has a packet to send. The case where this assumption does not hold is when a node does not have a packet to send.

For the throughput per node calculation the following formula is used. The per node throughput equals the number of packets received at the gateway from a certain node multiplied by the packet size and then divided by the data rate. To find the network throughput, total packet transmissions must be used because in order to reach the gateway, node 3’s packets must be transmitted over the link three times. The total throughput is calculated by \( Th_n = 3 \times Th_3 + 2 \times Th_2 + Th_1 \) where \( Th_n \) is network throughput and \( Th_3 \) is node 3’s etc.

4.4 Mesh Fairness Code

4.4.1 Code Manipulations for the IEEE 802.11 MAC Protocol

The code for the IEEE 802.11 MAC protocol had to be modified to meet the new specifications of the CMAC protocol (which had been done previously), but more modifications were necessitated to implement the MFA. The first modification that had to be made was a variable called mfa_var which was used to determine if the packet was the node’s own packet or another node’s packet and how many hops (single hop or multi-hop) away from the destination it was. This step differentiates each node in terms of hop count away from the gateway. This added variable is shown in bold. The following code is located in the mac-802_11.cc file.

```c
struct hdr_mac802_11* pkt_hdr = HDR_MAC802_11(p);
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);

int mac_addr = addr();
    //current node mac address
```
int pkt_src = ETHER_ADDR(pkt_hdr->dh_ta);
// packet current source address
int pkt_dst = ETHER_ADDR(pkt_hdr->dh_ra);
// packet current destination address
int orig_dst = (nsaddr_t)Address::instance().get_nodeaddr(ih->daddr());
// packet final destination address
int orig_src = (nsaddr_t)Address::instance().get_nodeaddr(ih->saddr());
// packet original source address

// Determine Mesh Fair Algorithm variable mfa_var
int mfa_var = 3; // default value
// mfa_var = 0 = "single-hop node, own packet";
// mfa_var = 1 = "single-hop node, other packet";
// mfa_var = 2 = "multi-hop node, own packet";
// mfa_var = 3 = "multi-hop node, other packet";

if ( (orig_src > -1) && (pkt_src > -1) )
{
    if ( mac_addr == orig_src ) // own packet
    {
        if ( (orig_dst > -1) && (pkt_dst > -1) )
        {
            if (orig_dst == pkt_dst) // single hop, own packet
            {
                mfa_var = 0;
            }
        }
        else // multi hop, own packet
        {
            mfa_var = 2;
        }
    }
    else // other nodes packet
    {
        if ( (orig_dst > -1) && (pkt_dst > -1) )
        {
            if (orig_dst == pkt_dst) // single hop, other nodes packet
            {
                mfa_var = 1;
            }
            else // multi hop, other nodes packet
            {
                mfa_var = 3;
            }
        }
    }
}

Figure 4: Modified Code in the mac-802_11.cc file

The mfa_var is then used in the mac_timer.cc file to determine how to set the backoff counter. Based on the hop count information from mfa_var, the backoff timer was set. If the node is a single hop node meaning one hop away from the gateway, the backoff counter was chosen uniformly from the range of [2 * CW, 3 * CW – 1] for it's own packets and [CW, 2 * CW -1] from forwarding packets. For multi-hop nodes, the
Backoff counter was chosen uniformly from \([CW, 2 \times CW - 1]\) for its own packets and \([0, CW - 1]\) for forwarding packets. The file with the backoff timer is mac-timers.cc. In the header file for the MAC protocol we have multiplied the contention window by 2 when we receive an ACK. So in order to obtain the original value of the contention window, we must divide it by 2 as is listed in bold in the code. Following is the code that was modified. The mfa_var was passed to this function from the mac-802_11.cc file. Again the mfa_var is shown in bold.

```c
void BackoffTimer::start(int cw, int idle, int colstat, int mfa_var)
{
    u_int32_t bcnew;
    if (colstat==2){ rtime=0;}
    else {
        Scheduler &s = Scheduler::instance();
        assert(busy_ == 0);
        busy_ = 1;
        paused_ = 0;
        stime = s.clock();

        if ( mfa_var == 0 ) // single-hop, own packet
            { if (cw > mac->phymib_.getCWMax())
                {bcnew = (Random::random() % (cw/2))+(cw/2);} else {
                if (colstat == 0) {bcnew=(Random::random() % cw)+2*cw;} else {bcnew = (Random::random() % cw);} }
        }
    else if ( mfa_var == 1 ) // single-hop, other packet
        { if (cw > mac->phymib_.getCWMax())
                {bcnew = (Random::random() % (cw/2))+(cw/2);} else {
                if (colstat == 0) {bcnew = (Random::random() % cw)+cw;} else {bcnew = (Random::random() % cw);} }
        }
    else if ( mfa_var == 2 ) // multi-hop, own packet
        { if (cw > mac->phymib_.getCWMax())
                {bcnew = (Random::random() % (cw/2))+(cw/2);} else {
                if (colstat == 0) {bcnew = (Random::random() % cw)+cw;} else {bcnew = (Random::random() % cw);} }
        }
    else if ( mfa_var == 3 ) // multi-hop, other packet
    {  
```
if (cw > mac->phymib_.getCWMax())
   {bcnew = (Random::random() % (cw/2));}
else{
   if (colstat == 0) {bcnew=(Random::random() % cw);}
   else {bcnew = (Random::random() % cw);}
}

Figure 5: Modified Code in the mac-timers.cc file

Next, to limit the rate of multi-hop nodes, the backoff counter was modified again. If a node has one of its own packets to send but has just sent one of its own packets, it must wait until the node’s parent sends a packet before it can send another packet. For example, if node 3 had just sent a packet and has another one of its own packets to send, it would wait to send its packet until its parent node 2 sends its own packet. During the time that a node is waiting for its parent to send a packet, the node’s backoff counter is not decremented. Once the parent node has sent a packet normal backoff counter function returns.

int BackoffTimer::pause(int cw, int recvDataj[4], int ActiveNode[4], int MyPacket)
{
   ...  

   int MyPack = 0; //my Packet
   int cnode;  //current Node

   cnode = mac->index_;

   if (cnode == mac->srcAddr_)
      MyPack = 1;

   if ((ActiveNode[cnode] == 2) && (MyPack == 1))
      rtime = rtime;
   else
   {
      rtime -= (slots * mac->phymib_.getSlotTime());
   }

Figure 6: Modified Code in the mac-timers.cc file for the parents packet

To identify when a node has just sent a packet the vector ActiveNode has been used see Figure 6. In the timer file “rtime” is used for the backoff counter so if
ActiveNode is set to 2 and it is my packet the backoff counter “rtime” is not decremented which essentially keeps the backoff counter from decrementing and reaching zero. A node will not send a packet until the backoff counter reaches zero.

```cpp
inline void Mac802_11::checkBackoffTimer()
{
    if(is_idle() && mhBackoff_.paused())
    {
        if (colstat_==1)
            mhBackoff_.resume(phymib_.getPIFS(), colstat_);
        else
            mhBackoff_.resume(phymib_.getDIFS(), colstat_);
    }
    else
    {
        if(! is_idle() && mhBackoff_.busy() && ! mhBackoff_.paused())
            mhBackoff_.pause(cw_, recvDataj, ActiveNodej, MyPacket);
    }
}
```

```cpp
int Mac802_11::check_pktTx() //this function sends Data Packet
{
...
    if((u_int32_t)ETHER_ADDR(mh->dh_ra) != MAC_BROADCAST) {
        //not a broadcast packet
        if(index_ == srcAddr_) //My Packet
            if ((index_ != 1) && (index_ != 0)) //node 0 or 1
                if (ActiveNodej[index_] != 2)
                    ActiveNodej[index_] = 1;
    }
...
}
```

```cpp
void Mac802_11::recv_timer()
{
...
    if(dst != (u_int32_t)index_ && dst != MAC_BROADCAST) {
        //not a broadcast packet
        if ((index_ != 1) && (index_ != 0)) //node 1 or 0
        {
            if (srcAddr_ == Parent_[index_]) //parent sending packet
                ActiveNodej[index_] = 0;
            //reset Active Node parent has transmitted packet
        }
    } /*
     * We don't want to log this event, so we just free
     * the packet instead of calling the drop routine.
     */
    discard(pktRx_, "---");
    goto done;
} 22
```
Mac802_11::recvACK(Packet *p)
{
...  
if ((index_ != 0) && (index_ != 1))
    if (ActiveNodej[index_] == 1){  //make sure it was My data packet

        ActiveNodej[index_] = 2;  //received ACK from my data packet
        
    }
...  

Figure 7: Modified Code in the mac-802_11.cc file for the parents packet

Figure 7 contains the code for identifying when a parent has sent its own packet as well as when the current node has sent its own packet. The vector ActiveNodej is set to one for a node when it sends its data packet when it receives the ACK packet it is then set to two. When the ActiveNodej vector for the current node is set to two this means the node has successfully sent one of its own packets and received the acknowledgement for the packet. The reason that ActiveNodej is set to one and then two is to determine when the ACK is received if it was my packet that was sent. The ActiveNodej vector is reset to zero when the node overhears its parent sending the parents own packet.

4.4.2 Code Manipulations for the Queue

First the droptail queue was modified to determine if a packet was the node’s own packet or a packet from another node that the current node was forwarding. If the packet was not current node’s own packet, it was inserted at the head of the queue. Packets that were the current node’s packets were inserted at the tail of the queue. This was done to assist in giving multi-hop nodes a higher priority then single hop nodes. In Figure 8 is the code that was added to do this in the drop-tail.cc file.
Variables that were added and the enqueue head function call are in bold. The variable orig_src is the original source node of the packet. To determine at which node the packet is at the moment, the variable pkt_src was used, and by comparing these two variables it can be determined if the packet is at the node which originated the packet and therefore whether or not it was the current node’s packet.

```c
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
struct hdr_mac *m = HDR_MAC(p);
struct hdr_mac802_11* pkt_hdr = HDR_MAC802_11(p);

int orig_src = ih->saddr(); //original source address
int pkt_src = ETHER_ADDR(pkt_hdr->dh_ta); //current source address

if (summarystats) {
    Queue::updateStats(qib_?q_->byteLength():q_->length());
}

int qlimBytes = qlim_ * mean_pktsize_
if (qib_ && (q_->length() + 1) >= qlim_ ||
    (qib_ && (q_->byteLength() + hdr_cmn::access(p)->size()) >= qlimBytes)){
    // if the queue would overflow if we added this packet...
    if (drop_front_){ /* remove from head of queue */
        q_->enque(p);
        Packet *pp = q_->deque();
        drop(pp);
    } else {  
        drop(p);
    }
} else {
    if (orig_src == pkt_src) //own packet enqueue at the tail
        q_->enque(p);
    else
        q_->enqueueHead(p);  //other nodes packet enqueue at the head
}
```

Figure 8: Modified Code in the drop-tail.cc file
The next step was to modify the priority queue to perform the same as the droptail queue. In Figure 9 is the code that was added to the priqueue.cc file. In bold are the variables that were added as well as modifications to the code.

```c
void PriQueue::recv(Packet *p, Handler *h)
{
    struct hdr_cmn *ch = HDR_CMN(p);
    struct hdr_ip *ih = HDR_IP(p);
    struct hdr_mac *m = HDR_MAC(p);
    struct hdr_mac802_11* pkt_hdr = HDR_MAC802_11(p);

    int orig_src = ih->saddr(); //original source address
    int pkt_src = ETHER_ADDR(pkt_hdr->dh_ta); //current source address

    if(Prefer_Routing_Protocols) {
        switch(ch->ptype()) {
            case PT_DSR:
            case PT_MESSAGE:
            case PT_TORA:
            case PT_AODV:
                recvHighPriority(p, h);
                break;
            default:
                if (orig_src != pkt_src) //other packet
                {
                    recvHighPriority(p, h); //enqueue at the head
                    break;
                }
                else
                {
                    Queue::recv(p, h); //own packet enqueue tail
                    break;
                }
        } //else
    } //if
    else {
        if (orig_src != pkt_src)
        {
            recvHighPriority(p, h); //other packet enqueue head
        }
        else
        {
            Queue::recv(p, h); //own packet enqueue tail
        }
    }
}
```

Figure 9: Modified Code in the priqueue.cc file
When the queue files were modified, it was observed that some packets from the current node were slipping in before packets from other nodes. One reason this could occur is when a node receives a packet on the physical layer, it must be forwarded up to the higher layers which takes a little time. If while a packet was passing up to the MAC layer, and the MAC layer sent a packet, then the node is inserting a packet of its own before packets from other nodes. If this new packet is then sent, then a packet is slipping in from the current node before the packet from the multi-hop node. To deal with this problem we have made a vector variable called recvDataj that is set to one when a node has received a packet. We passed this variable to the mac-timers.cc file where if this variable is set to one, we increased the backoff counter to give the node a longer amount of time before it sends another packet. For simulation this does not seem to help much with the problem so the code is not included. Another reason that a node’s own packet may be slipping in before other node’s packets is the scheduling scheme used in NS. From observation it seems that a packet is actually taken out of the queue, scheduled, and waiting to be sent before packets arrive from other nodes, therefore this one packet is still sent before the packets are sent from other nodes. After this first packet is sent, the packets are sent in the specified order from the queue.
CHAPTER 5
RESULTS

This chapter presents the results of the ns-2 simulations. All the results shown are based on a single branch of a WMNs consisting of three nodes (parking lot scenario) in a row with a gateway for the nodes. In Figure 10 is our network scenario for the simulations. The WMN is composed of 3 wireless routers (nodes) and a gateway router. Each of these routers is servicing a Wireless LAN which will be running the C-MAC protocol to provide per flow throughput fairness to the Wireless LAN. In turn each of the routers is running the MFA to provide per node throughput fairness to the WMN.

![Wireless Mesh Network for Simulation](image)

Figure 10: Wireless Mesh Network for Simulation

First we modified the C-MAC protocol code differentiating between nodes based on their hop count from the gateway. This was done by comparing the current node’s MAC address with the originating MAC address of the packet. The information gathered, based on hop count, is then used to determine the backoff counter. From the following Figure 13 it can be seen that this decreased the throughput. Now, the nodes
are being limited from sending based on hop count. The queue size has not been increased or modified so a lot of packets are not making it through the network but are being dropped instead. The following Figures compare the throughput for the IEEE 802.11 MAC protocol, C-MAC protocol, and the MFA protocol for the WMN simulation scenario. In Figure 12 it can be see that the C-MAC protocol cannot provide per node throughput fairness in a single branch of a WMN.

Figure 11: Throughput IEEE 802.11

Figure 12: Throughput C-MAC
The network throughput is about the same for each node and each protocol until we reach about 125 Kbps for the data rate. After 125 Kbps in the IEEE 802.11 protocol, node 1 starts to dominate throughput while nodes 2 and 3, but especially 3 start to degrade on the amount of throughput until they are getting little or no packets through which is basically zero throughput. The C-MAC protocol has higher throughput than the IEEE 802.11 but shows a similar pattern in degradation of throughput. The new MFA protocol has a very similar throughput pattern as the IEEE 802.11. The reason for this is because the queue has not been modified at this step. So even though node 2 and node 3 are given more opportunities to send packets, their packets get dropped before they reach the gateway. There are two reasons their packets get dropped. First, the queue size is set to fairly small at this point and secondly, because the packets are being inserted at the end or tail of the queue. Since packets are dropped from the tail of the queue, packets from nodes 2 and 3 are the first to get dropped.

The next step that was taken was to modify the queue by inserting packets from other nodes at the head of the queue and inserting packets from the current node at the
tail. In Figure 14 is the throughput for the MFA with the modified queue giving priority to multi-hop nodes. Here it can be seen that instead of node 3 dropping drastically in throughput, it actually starves node 2 and now has a much higher throughput than node 1. Node 2’s throughput starts going down at about 125 Kbps data rate again. Node 1’s throughput does not go completely down since it is only one hop away from the gateway. The range of the throughput values has also gone down since node 1 has been limited where as before node 1 was starving the rest of the network.

![Figure 14: Throughput for MFA with the Modified Queue](image)

Now it can be observed that node 3 has taken over the network. Node 3 needs to be limited so that node 2 can send some packets and get a higher throughput. At this point we have added the concept of a parent. Node 3’s parent is node 2, node 2’s parent is node 1 and so forth. We now limit node 3 by requiring node 3 to wait to send its own packet until node 2 sends one of its own packets. From the Figure 15 it can be seen that now the throughput is starting to even out between all nodes.
There are two reasons that the throughput is not totally fair between the nodes even though it has been improved and each node is getting a fairly steady amount of throughput. The first reason being that which has been mentioned before in chapter 4 under section 4.4.2 where there are still some packets from the current node slipping in first before we get the next packet from the queue. It is believed that this first reason will not affect the total throughput fairness too much. The second reason is the way the backoff timer is setup in NS. In the coding for the backoff timer the counter is not actually physically decremented instead the packet is scheduled to be sent at the time when the backoff timer should be zero. This creates a problem since if the packet is already scheduled to be sent, the backoff timer will not be checked unless it has to be paused for another node that is sending a packet in which case the packet will be rescheduled when the backoff timer is not paused. When the backoff timer is paused this will delay the sending of the packet since in the code we set “rtime” (the backoff counter value) equal to what it was before. This helps node 2 to not completely starve.
and to be able to send some packets. It also limits node 3 some, but it does not totally implement the parent concept. In order to totally implement the parent concept, a packet has to keep getting rescheduled until the parent node actually sends one of its own packets. If the packet is not scheduled it will be dropped which is not desirable either. Some method whereby to keep delaying the packet until the parent sends its packet and not just temporarily delaying the packet in scheduling is needed to finish solving this problem.

Using the formula in chapter 4 for the total network throughput, the total network throughput for IEEE 802.11 MAC protocol is compared to the MFA protocol. In Figure 16 it can be seen that to begin with MFA has a lower network throughput than the IEEE 802.11 MAC but in the long term MFA has a steady throughput that exceeds IEEE 802.11.
CHAPTER 6
DISCUTION AND CONCULSION

6.1 Expected Results

At the beginning of this work it could be seen that the current IEEE 802.11 MAC protocol does not provide a fair throughput in a WMN for all nodes especially nodes that are located a distance from the gateway. These nodes tended to get starved. Solutions that have been proposed mostly take more resources to implement and produce more overhead or are harder to implement, whereas the MFA is rather simple to implement and does not take a lot of overhead or resources. We expected that when we determined if a packet was from a multi-hop node or a single hop node that the throughput would not change much for multi-hop nodes since their packets would still get dropped at the queue. When the queue was modified to insert at the head for multi-hop nodes we expected that node 3 would dominate the throughput. We expected that when we added limits on the multi-hop nodes that the throughput for each node would even out fairly. We expected that MFA would produce a much more fair MAC protocol than is currently in place.

6.2 Actual Results

To begin with the actual results are what was expected. The results from implementing the parent concept are not totally what was expected. The reasons for this have been discussed at the end of chapter 5. Even though the results are not totally what was expected it can still be said that the MFA thus far still produces a better throughput than the IEEE 802.11 MAC protocol. This is because no node is starved for throughput and each node is getting a fairly steady amount of throughput even though each node is not at the same level of throughput.
6.3 Conclusion

The goal of this work was to achieve per node throughput fairness in a single branch of a WMN. The MFA provides an efficient and effective solution which has been demonstrated from the results that have been shown. This new protocol, MFA, is easily implemented via the C-MAC protocol for WMNs. An advantage of the MFA is that there is not a lot of overhead to implement the protocol and performance is considerably better than the base MAC protocol IEEE 802.11. Another advantage to the MFA is that it does not require a central authority or a lot of computation ability which is not a good option in WMNs since there is limited power, memory, and infrastructure. The key to the MFA MAC coordination is that the backoff timer is adjusted based on its relation to the gateway and the actions of its neighbors. The MFA provides a steady network throughput. It has been shown that the MFA algorithm actually does improve the performance of the MAC protocol and offers a better degree of fairness than IEEE 802.11 MAC protocol.

6.4 Future Work

This thesis leads to more possible areas of research. As discussed in chapter 5, improvements need to be made to the NS code for the backoff timer in conjunction with the packet scheduling to simulate more accurately the throughput performance of the MFA protocol. This protocol has been tested for CBR traffic in one direction but the effects of other types of traffic with flows in both directions has not been evaluated. This protocol could be tested on a longer branch to see how it performs as well as actually setting up a physical network to implement this protocol. Additionally, a mathematical model and quality of service model could be developed for this MFA protocol.
LIST OF REFERENCES


