

Optimum Fair Bandwidth Allocation Scheme for IEEE 802.16 Mesh Mode with Directional Antenna

Yanbin Lu*[†] and Guoqing Zhang*

*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China

Email: {yanbinlu, gqzhang}@ict.ac.cn

[†]Graduate School of Chinese Academy of Sciences, Beijing 100039, China

Abstract—The IEEE 802.16 standard is providing support for Mesh networks of which two scheduling mechanisms (i.e., centralized scheduling and distributed scheduling) are proposed. Centralized scheduling which builds on a routing tree behaves in a centralized manner and therefore presents more QoS guarantee than distributed scheduling. However, IEEE 802.16 standard doesn't specify how to schedule traffic in the mesh mode. In this paper, we focus on centralized scheduling and extensively explore its spatial reuse potential under the condition of directional antenna. In particular, we propose a scheduling mechanism which is optimum in terms of schedule length, and we introduce a bandwidth scaling algorithm in case the minimum schedule length is larger than the frame length. Simulation results show that our proposed scaling mechanism outperforms ordinary breadth first traversal.

Index Terms—Bandwidth Allocation, IEEE 802.16, Wireless Mesh Network.

I. INTRODUCTION

The IEEE 802.16 standard [1], also known as WiMax, enables easy and rapid worldwide deployment of low-cost BWA products, facilitates in broadband access, provides scalable alternatives to traditional wireline access, and accelerates the commercialization of BWA systems. There are two modes of operation in WiMax, Point-to-Multi-Point (PMP) that is mandatory and Mesh mode that is optional. In the PMP mode, Subscriber Stations (SSs) can only interact with the base station (BS) while, in mesh mode, SSs, without direct links with Mesh BS which serves as the interface to the backhaul, may route their traffic through other SSs to BS.

The Mesh mode, by permitting direct communications between SSs, allows multi-hop connections, the maintenance of which is provided by two specific scheduling mechanisms—centralized scheduling and distributed scheduling. Distributed scheduling is further classified into two types—coordinated and uncoordinated distributed scheduling. Coordinated distributed scheduling, by coordinating their transmissions in their extended two-hop neighborhood, suffers no collision. Uncoordinated distributed scheduling, used for setup of temporary bursts between a pair of neighboring nodes, acts more like ad-hoc. On the contrary, centralized scheduling relies on BS which determines the flow assignments and schedule transmissions for SSs.

This research work was supported by ICT innovation fund under contract number 20066033.

Centralized scheduling maintains a routing tree to schedule traffic to and from BS which is the root of the tree. Each SS periodically issue its flow request to BS by the order of its appearance in the routing tree. BS ensures collision-free scheduling over the links in the routing tree through warily granting flow assignments in the tree as per those requested by SSs. Some research work has identified that relaying traffic can increase the wireless throughput [2]. In addition, given a routing tree structure, the cycle time of SSs to relay new flow request is fixed, due to which, centralized scheduling is more suitable for QoS sensitive application than is coordinated distributed scheduling. Owing to these factors, we focus our attention on centralized scheduling mesh mode.

WiMax centralized scheduling mesh mode is based on TDMA, each frame of which is divided into 256 minislots. Mesh BS is responsible for assigning minislots to each link in the routing tree such that no collision occurs. However, how to schedule collision free transmission in the routing tree is left to the discretion of the implementor. Finding optimum link scheduling for general graph topology is NP-hard [3]. Furthermore, link scheduling is closely related to classic graph theoretic problem of vertex coloring [4], which is believed not to have bounded approximation algorithms [5]. But given a tree structure, reference [6] note there exists a optimum scheduling algorithm. What's more, by employing directional antenna, the problem of scheduling reduces to edge coloring for which P. Gupta and P. R. Kumar [7] has proposed a polynomial optimum scheduling algorithm, capable of dealing with any topology. Nevertheless, the high order time complexity makes it impractical for engineering implementation. Therefore, one objective of this paper is to propose an lightweight optimum scheduling algorithm by taking advantage of the combination of routing tree topology and edge coloring.

Moreover, if, after optimum scheduling, the duration of all resulting non-conflicting minislot allocation exceeds frame length, mesh BS must scale all the bandwidth requests fairly. Though standard [1] recommends scaling all allocations proportionally, yet it does not specify the process in detail. In this paper, by taking advantage of the benefits reaped from directional antenna, we propose a scaling scheme that can maximize minislots utilization while still treating fairly to each bandwidth request.

The rest of the paper is organized as follows. In section II, we provide some definitions and assumptions that we

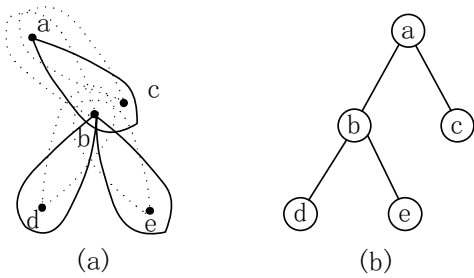


Fig. 1. Interference region

employ throughout the paper. In section III, we presents the optimum scheduling algorithm. Section IV presents the scaling algorithm in case the minimum schedule length is larger than frame length. Performance evaluation is described in section V. Finally, section VI concludes the paper.

II. PRELIMINARIES

In this paper, we assume that there is only one Mesh BS working in the network and providing the function of centralized scheduling in the routing tree. And we only account for two types of traffic—uplink traffic which transmits from SSs to BS and downlink traffic which transmits from BS to SSs—and these traffic may be relayed by other SSs.

We assume that all nodes work on a shared channel. Each node is assumed to be equipped with multiple directional antennas which can enhance or cancel out the radiating electromagnetic waves in certain directions [8]. A transmission is successful if and only if the transmitting node is in the reception beam of receiving node and the receiving node is in the radiation beam of the transmitting node. We assume that the number of antennas installed by one node is enough for covering all directions. With the advance of DSP technologies, an directional antenna is able to accurately control its beam pattern. So we assume that the beam of each antenna is adjusted such that the transmission between two nodes does not interfere with the transmission between two other nodes and that two nodes can communicate if and only if there is a link between them in the routing tree. The capacity of each link in the routing tree is not necessarily the same, and we assume it not to vary with time. Since SSs are generally mesh routers which are fixed at a specific position, we can simply steer the directions of antennas and control the beam pattern beforehand. We also assume that simultaneous transmission or reception by the same node is not allowed, which is justifiable for nodes under the control of single CPU.

In Fig. 1(a), the lines indicate the RF radiation beams and reception beams. The solid and dotted lines is of no difference, just for clarity. The resulting routing tree is shown in Fig. 1(b). From what has been discussed in last paragraph, the activeness of any two links with neither of their two ends in common can occur simultaneously. For example, transmission from node a to c and transmission from node d to b can happen concurrently though b is in the radiation beam of both node a and d and they are in the reception beam of node b . The reason is that the antenna at b for receiving data from a and the antenna for receiving data from d is not the same one.

Combining the assumptions discussed above, the resulting tree topology generated by directional antenna requires that any two links incident on the same node not be simultaneously active. Thus, the link scheduling problem reduces to edge coloring problem as introduced by [7].

III. OPTIMUM SCHEDULING

In this section, we give an algorithm of optimum scheduling for the routing tree formed by directional antenna. We represent the routing tree with a directed graph $T = (V, L)$ where V represents the set of nodes in the tree, and L denotes the set of links. We use $L(i)$ to represent the set of links incident on node i . We fix BS at node 0 with other nodes being SSs. Given a link l , we define c_l as the capacity of link l , $h(l)$ as the receiving end of l , and $t(l)$ as the transmission end of l . We use $l_{i,j}$ to indicate the link directed from node i to node j . We denote r_i^u and r_i^d as the uplink and downlink bandwidth request of node i respectively. $Children(i)$ indicates the set of children nodes of node i . $Subtree(i)$ stands for the set of subtree nodes of node i including i itself. i_p represents the parent node of i . A node located at k hops away from the root is at level k . A link is at level k if the higher level of its two endpoints is k . We define $UPLINK$ as the set of uplinks whose receiving end is at lower level than whose transmission end, $DOWNLINK$ as the set of downlinks whose transmission end is at lower level than whose receiving end. T_f signifies the total number of minislots of a frame.

The bandwidth request from all SSs can be converted to minislots requirement on each link by

$$t_l = \begin{cases} \sum_{a \in Subtree(t(l))} \lceil r_a^u \cdot T_f / c_l \rceil, & \text{if } l \in UPLINK \\ \sum_{a \in Subtree(h(l))} \lceil r_a^d \cdot T_f / c_l \rceil, & \text{if } l \in DOWNLINK \end{cases} \quad (1)$$

In the routing tree $T = (V, L)$, we split link l requesting t_l minislots into links l'_1, l'_2, \dots, l'_t , each requesting one minislot. The resulting multi-graph tree is $T' = (V, L')$. We use $L'(i)$ to represent the set of split links incident on node i . Note, in the multi-graph, there may be multiple links between a pair of nodes. The degree of a node will include both indegree and outdegree. Then the optimum scheduling algorithm is shown in Algorithm 1.

Theorem 1: Algorithm 1 allocates the minimum number of minislots.

Proof: The proof is by induction on the link levels, k , in the tree network. When $k = 1$, it follows immediately that algorithm 1 will generate minimum number of minislots since it is at least the degree of root r .

Assume the theorem holds for all $k - 1$ level links. Without loss of generality, we focus on an arbitrary node of level $k - 1$, say node a . Assume that the total number of slots node a needs to assign to its level k links is γ , that the total number of slots allocated to the tree so far is ω , and that the total number of minislots allocated to links between a and its parent is θ . Then there are two cases.

Case 1: $\gamma \leq \omega - \theta$. It is obvious that the assignment is minimum since no new minislots are needed.

Algorithm 1: Optimum link scheduling

Input: A multi-graph tree network $T' = (V, L')$ with root r
Output: A slot allocation $sa: L \rightarrow \{1, 2, \dots\}$
1 Assign the links incident on r minislots $1, 2, \dots$, degree(r);
2 $Queue.push(u), \forall u \in Children(r) \setminus \{r\}$;
3 **while** $Queue$ is not empty **do**
4 $v \leftarrow Queue.pop()$;
5 $E \leftarrow \{sa(l') : l' \in L'(v) \cap L'(v_p)\}$;
6 **foreach** $l' : l' \in L'(v)$ **and** $l' \notin L'(v_p)$ **do**
7 Let s be the least minislot $\notin E$;
8 $sa(l') \leftarrow s$;
9 $E \leftarrow E \cup \{s\}$
10 **end**
11 $Queue.push(u), \forall u \in Children(v) \setminus \{v\}$;
12 **end**

Case 2: $\gamma > \omega - \theta$. The total number of minislots utilized, including those allocated to node a , is $\omega + [\gamma - (\omega - \theta)] = \gamma + \theta$, which is the degree of node a . Therefore, the assignment is minimum since the minimum number of minislots is at least the degree of node a . \square

Lemma 1: The minimum scheduling length yielded by Algorithm 1 is $\max_{a \in V} (\sum_{l \in L(a)} t_l)$.

Proof: When $k = 1$, the total number of minislots assigned to level 1 links is the degree of the root. And it follows from theorem 1 that each time the total number of allocated minislots, ω , increases, it increases to a new node's degree. Therefore the final ω is equal to or less than $\max_{a \in V} (\sum_{l \in L(a)} t_l)$. However, $\max_{a \in V} (\sum_{l \in L(a)} t_l)$ is the lower bound on minimum scheduling length. So the lemma follows. \square

IV. FAIR BANDWIDTH ALLOCATION

Algorithm 1, be it producing a minimum scheduling length, is meaningless if the minimum scheduling length is larger than the frame length T_f . In this section, we will deal with the case that minimum scheduling length is greater than T_f and introduce an algorithm that fairly scales each bandwidth request, as well as maximizing the total throughput. By fairness, we mean, at each node, a subtree node cannot increase its minislots allocation without diminishing the minislots allocation of another node that has a smaller ratio of bandwidth provisioned by its minislots allocation to the bandwidth it requests.

Since the uplink and downlink bandwidth request of a node is of no difference to our algorithm, we use r_i to represent the bandwidth request of node i . In other words, we only consider the uplink or downlink bandwidth request. But this is totally for the purpose of brevity and we will show later how to deal with the case that both uplink and downlink bandwidth request exist. The scaling factor of r_i is called α_i . In order to reference the bandwidth request corresponding to a specific scaling factor α , we use $\alpha.n$ to indicate the node issuing the request, so the corresponding request can be referenced by $r_{\alpha.n}$. For terseness, we define vector $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_{|V|}]$. Every

Algorithm 2: Scale(a, α)

Input: node a and α which is both input and output variable
1 **if** $\sum_{i \in Subtree(a)} (\lceil \tau_{l_{F_a(i)}}^{\alpha_i r_i} \rceil + \lceil \tau_{l_{S_a(i)}}^{\alpha_i r_i} \rceil) \leq T_f$ **then**
2 **return**
3 **end**
4 Scale $\tau_{l_{F_a(i)}}^{r_i}, \tau_{l_{S_a(i)}}^{r_i}, \forall i \in Subtree(a)$ proportionally so that the sum of them equals T , and then round down the fraction with the result being $t_{l_{F_a(i)}}^{r_i}, t_{l_{S_a(i)}}^{r_i}, \forall i \in Subtree(a)$;
5 **foreach** $i \in Subtree(a)$ **do**
6 $\beta_i \leftarrow \min (b_{l_{F_a(i)}}(t_{l_{F_a(i)}}^{r_i}), b_{l_{S_a(i)}}(t_{l_{S_a(i)}}^{r_i})) / r_i$;
7 $\beta_i.n \leftarrow i$;
8 **if** $\beta_i \geq \alpha_i$ **then**
9 $\beta_i \leftarrow \alpha_i$;
10 $B \leftarrow B \cup \{\beta_i.n\}$;
11 **end**
12 $t_{l_{F_a(i)}}^{r_i} \leftarrow \lceil \tau_{l_{F_a(i)}}^{\beta_i r_i} \rceil$;
13 $t_{l_{S_a(i)}}^{r_i} \leftarrow \lceil \tau_{l_{S_a(i)}}^{\beta_i r_i} \rceil$;
14 **end**
15 $T_{rem} \leftarrow T_f - \sum_{i \in Subtree(a)} (t_{l_{F_a(i)}}^{r_i} + t_{l_{S_a(i)}}^{r_i})$;
16 **while** $T_{rem} > 0$ **and** $Subtree(a) \setminus B \neq \emptyset$ **do**
17 select β' , the smallest β where $\beta.n \in Subtree(a) \setminus B$ and denote $\alpha_{\beta'.n}$ to be α' . When there are more than one smallest β , select arbitrary one;
18 **if** $b_{l_{F_a(\beta'.n)}}(t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}}) < b_{l_{S_a(\beta'.n)}}(t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}})$ **then**
19 $t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}} \leftarrow t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}} + 1$;
20 $T_{rem} \leftarrow T_{rem} - 1$;
21 **else if** $b_{l_{F_a(\beta'.n)}}(t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}}) > b_{l_{S_a(\beta'.n)}}(t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}})$ **then**
22 $t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}} \leftarrow t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}} + 1$;
23 $T_{rem} \leftarrow T_{rem} - 1$;
24 **else if** $b_{l_{F_a(\beta'.n)}}(t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}}) = b_{l_{S_a(\beta'.n)}}(t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}})$ **then**
25 **if** $T_{rem} \geq 2$ **then**
26 $t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}} \leftarrow t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}} + 1$;
27 $t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}} \leftarrow t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}} + 1$;
28 $T_{rem} \leftarrow T_{rem} - 2$;
29 **else**
30 $B \leftarrow B \cup \{\beta'.n\}$;
31 **continue**;
32 **end**
33 **end**
34 $\beta' \leftarrow$
35 $\min (b_{l_{F_a(\beta'.n)}}(t_{l_{F_a(\beta'.n)}}^{r_{\beta'.n}}), b_{l_{S_a(\beta'.n)}}(t_{l_{S_a(\beta'.n)}}^{r_{\beta'.n}})) / r_{\beta'.n}$;
36 **if** $\beta' \geq \alpha'$ **then**
37 $\beta' \leftarrow \alpha'$;
38 $B \leftarrow B \cup \{\beta'.n\}$;
39 **end**
40 $\alpha_i \leftarrow \beta_i, \forall i \in Subtree(a)$;

Algorithm 3: Breadth first determining α

Input: A tree network $T = (V, L)$ with root r

- 1 $\alpha \leftarrow 1$;
- 2 $Queue.push(r)$;
- 3 **while** $Queue$ is not empty **do**
- 4 $a \leftarrow Queue.pop()$;
- 5 Scale (a, α);
- 6 $Queue.push(v), \forall v \in Children(a)$;
- 7 **end**

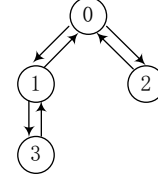


Fig. 2. $T_f = 8, c_{l_{0,1}} = c_{l_{1,0}} = c_{l_{0,2}} = c_{l_{2,0}} = 4m/s, c_{l_{3,1}} = c_{l_{1,3}} = 2m/s, r_1^u = 4m/s, r_2^u = 2m/s, r_3^u = 1m/s, r_3^d = 4m/s$

node i 's bandwidth request where $i \in Subtree(j) \setminus \{j\}, j \neq 0$ requires two links incident on j , of which we call the link farther to BS the first link and the link nearer to BS the second link, to allocate minislots. We denote the first link at node j for bandwidth request of node i as $l_{F_j(i)}$, the second link as $l_{S_j(i)}$. Note, in the case that $j = i$ or $j = 0$, there is only one uplink or downlink incident on j for bandwidth request of i , so we use $l_{F_i(i)}$ and $l_{S_0(i)}$ to represent them, and $l_{S_i(i)}$ and $l_{S_0(i)}$ don't exist. For example, in Fig. 2, the uplink bandwidth request of node 3, requires link $l_{3,1}$ and link $l_{1,0}$ incident on node 1 to allocate minislots, so we denote $l_{3,1}$ as $l_{F_1(3)}$ and denote $l_{1,0}$ as $l_{S_1(3)}$. The only link incident on node 1 for uplink request of 1 is $l_{1,0}$ and we call it $l_{F_1(1)}$. The only link incident on node 0 for uplink request of 2 is $l_{2,0}$ and we call it $l_{F_0(2)}$. We use τ_l^r to signify the fractional number of minislots allocated to l to meet bandwidth request r , in other words, $\tau_l^r = \frac{r \cdot T_f}{c_l}$. Function $b_l(t)$ will return $\frac{t \cdot c_l}{T_f}$, the bandwidth provisioned by l if given t minislots. Note we define $\tau_{l_{S_j(i)}}^r, \tau_{l_{S_0(i)}}^r, \forall j \in V$ to be zero and define $b_{l_{S_j(i)}}(t), b_{l_{S_0(i)}}(t), \forall j \in V$ to be ∞ for the sake of brevity of algorithm description.

Given lemma 1, we can guarantee that the minimum scheduling length not exceed T_f by scaling the number of minislots assigned to links incident on each node so that the sum of them doesn't exceed T_f , in other words, by requiring

$$\sum_{i \in Subtree(a)} \left(\lceil \tau_{l_{F_a(i)}}^{\alpha_i r_i} \rceil + \lceil \tau_{l_{S_a(i)}}^{\alpha_i r_i} \rceil \right) \leq T_f, \forall a \in V. \quad (2)$$

We propose Algorithm 2 to do the job of scaling bandwidth fairly at a specific node a . Steps 1–3 make sure whether to scale or not. Step 4 proportionally scales each minislot allocation and rounds them down so that they are integral. Step 6 takes the smaller bandwidth provided by the first and the second links, and calculate current scaling factor β . Note step 6 is correct when $a = i$ or $a = 0$ since $b_{l_{S_i(i)}}(t), b_{l_{S_0(i)}}(t)$ is set to ∞ . Step 7 stores the node label for later use. If other nodes have scaled the bandwidth and the scaling factor is smaller than the current one, steps 8–11 will make the scaling factor replace the current one and add this node to set B whose nodes are restrained from augmenting scaling factor later. In step 6, some minislots allocated to the link providing larger bandwidth is otiose, so steps 12, 13 and 15 are responsible for freeing these unnecessary minislots. Step 17 selects the most unfair (smallest scaling factor) node. If this node's first link provides less bandwidth than its second link under current allocation, step 19 increases the allocations of the first link by 1. If this node's first link provides more

bandwidth than its second link under current allocation, step 22 increases the allocations of the second link by 1. If the first link and the second link provides equivalent bandwidth, step 26–27 add one minislot to both of them. Step 30 prevents the node whose first and second link provide same bandwidth from being selected again when residual minislots are less than two. Steps 34–38 redo the job of steps 6–11. Note we need not redo step 12–13 since current minislot allocations can guarantee no unnecessary minislots exist. Step 40 finally affirm each scaling factor.

Given Algorithm 2, we can easily meet Constraint 2 through breadth first traversal as shown in Algorithm 3. Here we use an example whose configuration is shown in Fig. 2 to illustrate this algorithm. Fig. 2 shows the capacity of each link and the request of each node. Since there are both uplink and downlink request in this example, we use β_i^u and β_i^d to indicate their corresponding β_i in Algorithm 2. First Algorithm 3 visits node 0 and runs Algorithm 2 at node 0. At step 4, $\tau_{l_{1,0}}^{r_1^u} = 8, \tau_{l_{2,0}}^{r_2^u} = 4, \tau_{l_{1,0}}^{r_3^u} = 2, \tau_{l_{0,1}}^{r_3^d} = 8$ and after scaling, $t_{l_{1,0}}^{r_1^u} = 2, t_{l_{2,0}}^{r_2^u} = 1, t_{l_{1,0}}^{r_3^u} = 0, t_{l_{0,1}}^{r_3^d} = 2$. After steps 5–14, $\beta_1^u = 0.25, \beta_2^u = 0.25, \beta_3^u = 0, \beta_3^d = 0.25$ and t_l^r doesn't change during steps 12–13. At step 15, $T_{rem} = 3$, and steps 16–38 will run in three iterations. In the first iteration, step 17 selects β_3^u , and $t_{l_{1,0}}^{r_3^u}$ becomes 1 and β_3^u becomes 0.5. In the second iteration, β_1^u is selected and, $t_{l_{1,0}}^{r_1^u}$ becomes 3 and β_1^u becomes 0.375. In the third iteration, β_2^u is selected and, $t_{l_{1,0}}^{r_2^u}$ becomes 2 and β_1^u becomes 0.5. So the scaling factor for each request after visiting node 0 is $\alpha_1^u = 0.375, \alpha_2^u = 0.5, \alpha_3^u = 0.5$ and $\alpha_3^d = 0.25$. When Algorithm 3 invokes Algorithm 2 at node 1, at step 4, $\tau_{l_{1,0}}^{r_1^u} = 8, \tau_{l_{3,1}}^{r_3^u} = 4, \tau_{l_{1,0}}^{r_3^u} = 2, \tau_{l_{1,3}}^{r_3^d} = 16, \tau_{l_{0,1}}^{r_3^d} = 8$ and after scaling, $t_{l_{1,0}}^{r_1^u} = 1, t_{l_{3,1}}^{r_3^u} = 0, t_{l_{1,0}}^{r_3^u} = 0, t_{l_{1,3}}^{r_3^d} = 3, t_{l_{0,1}}^{r_3^d} = 1$. After steps 5–14, $\beta_1^u = 0.125, \beta_3^u = 0, \beta_3^d = 0.125$, and $t_{l_{1,3}}^{r_3^d}$ becomes 2 at step 12. At step 15, $T_{rem} = 4$ and steps 16–38 will run in four iterations. In the first iteration, step 17 selects β_3^u , both $t_{l_{3,1}}^{r_3^u}$ and $t_{l_{1,0}}^{r_3^u}$ becomes 1, and β_3^u becomes 0.25. In the second iteration, step 17 selects $\beta_1^u, t_{l_{1,0}}^{r_1^u}$ becomes 2 and β_1^u becomes 0.25. In the third iteration, β_3^d is selected but step 31 will be reached since $T_{rem_u} = 1$. In the fourth iteration, β_1^u will be selected again, and $t_{l_{1,0}}^{r_1^u}$ becomes 3 and β_1^u becomes 0.375. So the scaling factor for each request after visiting node 1 is $\alpha_1^u = 0.375, \alpha_2^u = 0.5, \alpha_3^u = 0.25$ and $\alpha_3^d = 0.125$. These will be the final scaling factor because no further scaling is needed when Algorithm 3 visits node 2 and node 3.

Algorithm 4: Proposed approach to determining α

Input: A tree network $T = (V, L)$ with level k ;
Bandwidth request of each node;

Output: the scaling factor

```
1  $\psi \leftarrow k$ ;  
2 while  $\psi \geq 0$  do  
3   foreach level  $\psi$  node  $a$  do  
4      $\alpha_a \leftarrow 1$ ;  
5     Scale ( $a, \alpha$ );  
6   end  
7    $\psi \leftarrow \psi - 1$ ;  
8 end
```

The above example also shows that we can deal with the case that both uplink and downlink bandwidth request exists simply by treating them fairly. Although breadth first traversal can guarantee Constraint 2 at each node, yet it doesn't fully use the available bandwidth. In the above example, when Algorithm 3 visits node 1 and scales the bandwidth request in *Subtree*(1), there will be one minislot spared at node 0 which has to be wasted. The reason for this is that the scaling factors generated by node 1 for bandwidth requests from 3 is smaller than that generated by node 0 and the traversal order makes it impossible for breadth first traversal to reuse the spared minislots. If we assign this spare minislot to node 2, the scaling factor of node 2 can increase to 0.75 without undermining other requests' scaling factors, therefore keeping fairness to other requests. In order to fully take advantage of available minislots at each node, we recommend down-top traversal of the nodes in the routing tree as shown in Algorithm 4. By changing the traversal order, it is easy to see that Algorithm 4, with respect to the configuration of Fig. 2, increases α_2^y from 0.5 to 0.75 while keeping other scaling factors from deteriorating, compared to Algorithm 3.

After scaling and recalculating the total feasible minislots allocated on each link, we can get the optimum scheduling by running Algorithm 1.

V. PERFORMANCE EVALUATION

In this section, we compare the performance of Algorithm 4 with respect to that of Algorithm 3 in random configuration. We use randomly generated tree topology whose root is also randomly selected. Each node has 2 units of both uplink and downlink bandwidth request. The capacity of each link is randomly distributed between 10 and 60 units. We set $T_f = 256$, the default value of number of minislots in a frame. We vary the number of nodes in the routing tree from 10 to 50 and measure the ratio of Algorithm 3 to Algorithm 4 in terms of the sum of all bandwidth requests' scaling factors. We present our result in Fig. 3. Each data point is the average of the ratio attained over 10 random trees with the maximum degree varying from 3 to 12.

In Fig. 3, the result shows that Algorithm 3 averagely performs within 85% of Algorithm 4. Since the capacity of each link is randomly selected, the variation of number of nodes seems not to have significant impact on the result.

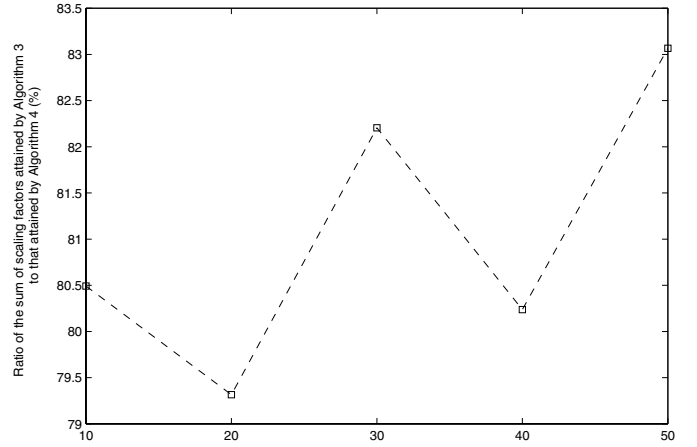


Fig. 3. Algorithm 3 vs. Algorithm 4

VI. CONCLUSION

We modeled the scheduling problems for IEEE 802.16 centralized mesh mode with directional antenna as edge coloring problems and design a lightweight optimum scheduling for it. In addition, we introduced a fair scaling algorithm that works in case the minimum scheduling length is longer than frame length. We also identified the impact of traversal order of scaling algorithm on the total throughput utilization and showed that down-top traversal can make use of available minislots better. Simulation result showed that breadth first traversal performs within 85% of down-top one.

REFERENCES

- [1] *IEEE Standard for Local and metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Std. 802.16, 2004.
- [2] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. Inform. Theory*, vol. 46, no. 2, pp. 1244–1245, Mar. 2000.
- [3] E. Arikan, "Some complexity results about packet radio networks," *IEEE Trans. Inform. Theory*, vol. 30, pp. 681–685, July 1984.
- [4] S. Ramanathan and E. L. Lloyd, "Complexity of certain graph coloring problems with applications to radio networks," Dep. Comput. Sci., Univ. Delaware, Tech. Rep. 92-18.
- [5] M. R. Garey and J. D. S., *Computers and brtracrability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
- [6] S. Ramanathan and E. L. Lloyd, "Scheduling algorithms for multihop radio networks," *IEEE/ACM Trans. Networking*, vol. 1, no. 2, pp. 166–177, Apr. 1993.
- [7] P. Gupta and P. R. Kumar, "Link scheduling in polynomial time," *IEEE Trans. Inform. Theory*, vol. 34, no. 5, pp. 910–917, Sept. 1988.
- [8] L. Bao and J. J. Garcia-Luna-Aceves, "Transmission scheduling in ad hoc networks with directional antennas," in *Proc. ACM MOBICOM*, Sept. 2002, pp. 48–58.