A Scheduling and Call Admission Control Algorithm for WiMax Mesh Network with Strict QoS Guarantee

Pulkit Goyal and Anirudha Sahoo
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay, Powai, Mumbai, India, 400076
email: {pulkitgoyal, sahoo}@cse.iitb.ac.in

***Abstract*— The IEEE 802.16 standard (commonly known as WiMax) has emerged as a broadband wireless technology covering large geographical area while providing high speed data rates with native Quality of Service (QoS) support. In this paper, we study mesh mode of operation of WiMax with centralized scheduling for UGS and RTPS service classes. We briefly discuss two known routing algorithms (to find the path of a request) and propose two new routing algorithms. We present a novel scheduling and Call Admission Control (CAC) algorithm for UGS and RTPS service class. The scheduling and CAC algorithm make sure that each and every packet of admitted request strictly meets its delay and jitter constraints. Since an RTPS request can change its data rate requirement, we propose an efficient algorithm for computing extra bandwidth request for RTPS service class which perform much better in terms of average packet delay and packet drop percentage compared to some simple algorithms. We present simulation results comparing our scheduling algorithm with two other algorithms proposed in the literature. We also present results which show that our scheduling does provide strict QoS guarantee for every packet.**

I. Introduction

The IEEE 802.16 standard, *Air Interface for Fixed Broadband Wireless Access Systems* [1], has been ratified by IEEE as a Wireless Metropolitan Area Network (WMAN) technology. This technology is designed for broadband wireless *last-mile* access in a Metropolitan Area Network (MAN), with performance comparable to traditional cable, DSL, or T1 services. Although IEEE 802.11 technology is very widely used, it is limited to LAN environment, because of its limited transmission range. WiMax has a transmission range of few kilometers with high bandwidth and it also supports Quality of Service (QoS) by providing various service classes. The service classes in WiMax have been carefully designed to support real time applications like voice and video and non-realtime application like large file transfer. Thus, WiMax is a very attractive technology for providing integrated voice, video and data services in the last mile.

WiMax supports point-to-point, point-to-multipoint and mesh topology. In point-to-point mode two nodes, called subscriber station (SS), communicate with each other like peers without any control from any other node. In point-to-multipoint mode, it operates in a star topology where the base station (BS) remains at the root and makes the scheduling decision for all the SSs. In this mode, all the communication has to pass through the BS. In mesh mode, the SSs can communicate with each other in a multi-hop fashion and the data path may not pass through the mesh BS (MBS). Unlike PMP mode, there are no separate uplink and downlink subframes in the mesh mode. The mesh mode only supports Time Division Multiple Access (TDMA) for channel access among the MBS and SSs.

The mesh mode MAC supports both centralized and distributed scheduling [1]. In this study, we consider a centralized scheduling scheme. In mesh centralized scheduling, the MBS performs very basic function similar to a BS in PMP mode. The scheduling of all the nodes in the network is determined by the MBS. When an SS wants to set up a new connection to another SS, it sends the request to the MBS. The MBS runs a routing algorithm to find the path of the request (from source SS to the destination SS) and runs a scheduling algorithm to allocate slots along the path of the request. The MBS sends out scheduling information to all the SSs in the network periodically (known as *scheduling period*). SSs upon receiving the schedule, send data packets accordingly.

WiMax network supports four different service classes of traffic: (1) Unsolicited Grant Service (UGS) (2) Real-time Polling Service (RTPS) (3) Non-Real-time Polling Service and (4) Best Effort Service. The standard provides specification for these different services, but does not specify any scheduling algorithm. But WiMax system supports QoS guarantee in terms of bandwidth, delay and jitter. Hence scheduling and call admission control (CAC) play an important role in such a system. When a new request arrives, the CAC module makes sure that the new request as well as all the existing requests can be scheduled such that all of them meet their QoS requirements. Hence, the CAC module works closely with the scheduling module while admitting a new request.

In this paper, we present a load balanced centralized scheduling and call admission control algorithm for IEEE 802.16 mesh networks which provide strict delay and jitter guarantee to realtime applications. We discuss two routing algorithms known in the literature to find the path between source and destination SSs and propose two new algorithms which provides better performance over the known algorithms. Although there are many scheduling algorithms proposed in the literature for WiMax mesh networks (see Section II), to the best of our knowledge, none of them provide packet level strict QoS guarantee. We present a scheduling algorithm which ensures that the TDMA slots are allocated to the requests such that *every* packet of admitted requests strictly meets its bandwidth, delay and jitter requirements. Thus, our scheduling algorithm can be used for hard real-time applications. The scheduling algorithm uses a novel idea of logically partitioning a request into two parts. The first part spans from source node to the penultimate node and second part is from penultimate node to the destination node. The delay constraints of the partitioned requests are appropriately computed from the original delay constraint. The advantage of this partition is that the

978-1-4244-5489-1/10/$26.00

first part of the connection does not need to worry about jitter constraint, it only needs to meet its part of the delay constraint. The second part of the partitioned connection is responsible for meeting the jitter constraint. Decoupling the delay and jitter constraint by partitioning the request provides more flexibility to our scheduling algorithm. We propose a CAC algorithm which works closely with our scheduling module to ensure that QoS requirements of all the requests are fulfilled while admitting a new request. We present simulation results of our scheduling algorithm and compare it with two other algorithms proposed in the literature. Using simulation, we also show that our CAC does ensure that all the requests meet their respective QoS constraints at the packet level. Then we discuss bandwidth allocation of RTPS requests. Unlike UGS service class, RTPS service class has a variable bandwidth requirement. So, we present an efficient algorithm for RTPS requests to make extra bandwidth request and compare its performance with some simple algorithms.

II. Related Work

Algorithms for wireless mesh networks have been proposed in [2]–[4]. While these results are not specifically within 802.16 framework, the insights they provide are helpful. Many researchers have proposed centralized scheduling algorithms for WiMax mesh networks. In [5], the authors have proposed centralized scheduling and routing tree construction algorithms. In this paper, routing tree rooted at the BS is used while finding path from source to destination. They form the routing tree from the network graph by choosing paths which have minimum interfering links. In [6], routing tree is constructed using metric called blocking metric *B(k)*. The blocking metric B(k) of a multi-hop route indicates the number of blocked nodes by all the intermediate nodes along the route from BS to the destination node k. They also define blocking value for node i which is the number of nodes blocked when node i is transmitting. Blocking metric for any routing path is taken as summation of blocking values of all the nodes along the path. Thus, it uses a interference aware route construction algorithm and an enhanced centralized mesh scheduling scheme, which considers node level interference conditions. [7] is as an extension of the algorithm described in [5]. This paper uses multiple metrics while constructing the routing tree. The metrics considered are interference factor, load balancing of the links and priorities assigned to different QoS classes. All these three methods assumes that traffic is from SS to BS or from BS to SS only. But Mesh Networks allow transmission between any two SSs which may or may not involve the MBS. In [8] also the authors assume that traffic is between SS to BS but they admit the flow only if its end to end QoS requirement is guaranteed. The main problem with this algorithm described in [8] is that load is not balanced on the nodes. Hence there could be some heavily loaded nodes which may become bottlenecks in the network.

In [9], authors describe both centralized and distributed broadcast scheduling algorithms. Both distributed and centralized algorithms are implemented in two stages. In the first stage, each node is assigned exactly one slot in the frame. In the second stage, each slot is filled with extra nodes such that each slot has a maximal broadcasting set. This works well if the traffic is not bursty, but it becomes difficult to handle bursty traffic using this algorithm. In [10], authors describe interference free distributed broadcast scheduling algorithm. They do not provide details of the algorithm. The frame length is always equal to the number of nodes in the network topology graph. If the number of nodes are very large then frame length becomes a problem. This algorithm also produces a non-uniform schedule and thus results in unfairness.

In [11], authors have presented an efficient fair scheduling algorithm for IEEE 802.16 multi hop mesh networks according to a new fairness model in which the bandwidth allocation is contingent on the actual traffic demands in such a way that the capacity region is not sacrificed by imposing the fairness constraints. They formulated the scheduling problem to maximize the system throughput subject to the fairness condition. By exploiting the characteristics of WiMax mesh networks, they developed an efficient algorithm to find the optimal schedule. [12] presents a scheduling algorithm which provides per flow QoS guarantees to real and interactive data applications by utilizing the network resources efficiently. Authors propose scheduling algorithm for real-time and non-real-time (data) traffic. Real time application uses UDP whereas data traffic uses TCP. UDP traffic is considered as high priority traffic over TCP traffic since it deals with real time applications such as VoIP, video conferencing etc.

III. Network Model

We represent a WiMax mesh network as a Graph G having N nodes and E links. The nodes are numbered 1 to N. A link between two nodes i and j, denoted as l_{ij}, is said to exist if node i can communicate with node j. We assume that a link is bidirectional, i.e., when link l_{ij} exists, link l_{ji} also exists.

A link l_{xy} is said to be blocking link l_{ij}, iff any one of the following conditions hold:

- Node $x \in \{i, j, Nbr(j)\}$
- Node $y \in \{i, j, Nbr(i)\}$

where $Nbr(i)$ represents neighbors of node i. Note that the above definition considers a link to *block* another link if there is a common node between the two links. This means there cannot be simultaneous communication over the two links either because there is only one transceiver (hence the common node cannot transmit and receive at the same time) or the common node cannot communicate with two neighbors at the same time. The other scenario of one link blocking another is the classical interference of a link with another due to hidden node.

We consider a Time Division Duplex (TDD) WiMax system [1]. We assume all the links to be of same capacity and that the channel conditions do not change. Although these assumptions may not be valid in practice, we want to first solve the CAC and scheduling problem under these assumptions (the problem is quite difficult even with this assumption) and relax the restriction as part of our future work.

IV. Routing in WiMax Mesh Network

In a mesh network, the routing protocol has to determine the path between source and destination. Once the path is determined, then the scheduling module needs to make sure that a request is schedulable on each link along the path. There are different approaches by which the path can be determined. We outline some of the approaches used in the literature and some we propose in this paper.

A. Routing Tree Rooted at the BS (RTRB)

A routing tree rooted at the mesh base station is generated by having subscriber station join the tree by choosing its

neighbor based on some metric like minimum hop count or some other metric described in [5]–[7], [13]. For any request, the path from source to destination is found using this routing tree. The major drawback of this method is that because of tree structure, some links which are physically present in the network do not participate in communication. This may lead to traversal of longer path even though shorter paths may be available. Also, the missing links in the routing tree do not share any load, leading to reduced capacity of the network.

B. Simple Weighted Shortest Path (SWSP)

This is the traditional algorithm used to find the shortest path from the source to the destination. The weight metric for the links could be hop count or any other suitable metrics like minimum interference path etc. This method solves the problem of longer path being selected.

In this method, only single shortest path is taken into consideration while scheduling a request. Hence, if there is no schedule available along that path then the request is simply rejected. But there may exist some other path along which the request may be schedulable. Thus performance of this method may be affected due to this constraint.

C. Weighted Shortest Path with Retry (WSPR)

The above two methods were proposed in the literature. Now we propose this method in which shortest path algorithm is used to find the path between the source and destination. But if the request is not schedulable along the path, then the bottleneck link is removed from the topology and shortest path is recalculated. This process continues until a schedulable shortest path is found or the source and destination are not connected anymore. Bottleneck link is the link with maximum number of interfering links. This algorithm is shown in Algorithm 1.

Algorithm 1 Weighted Shortest Path with Retry

```
1: while source node and destination node is connected do
2:     find the shortest path from source to destination for the new
       request.
3:     if CAC accepts the request then exit
4:     else remove the bottleneck link
5: end while
6: reject the request
```

D. Load Balanced Weighted Shortest Path with Retry (LB-WSPR)

This method, proposed by us, is similar to the *Weighted Shortest Path with Retry* algorithm. But the link weights are dynamically changed to account for the change in load on the links. Weight of a link l_{ij} is given by

$$w_{ij} = L_{ij} * B_{ij} \tag{1}$$

where L_{ij} is the load on link l_{ij} and B_{ij} is the number of links which block simultaneous communication of link l_{ij}. Load on a link is defined as the fraction of slots in a TDD frame used in transmission over the link. We provide an example to compute B_{ij} using Figure 1. Let us find out B_{12}, i.e., the number of blocking links for l_{12}. Links interfering with l_{12} are $l_{21}, l_{18}, l_{17}, l_{81}, l_{71}, l_{68}, l_{24}, l_{42}, l_{23}, l_{32}, l_{45}$. So, $B_{12} = 11$.

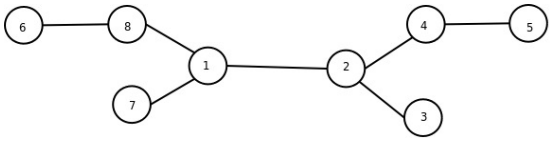

Fig. 1. Example Illustrating Interference

V. System Model for UGS and RTPS Requests

WiMax supports four types of service classes [1]. Out of them two are most important for QoS-based applications. We study these two service classes and present details of these two service classes. For details of other service classes please refer to [1].

1) **Unsolicited Grant Service (UGS)** : The UGS is designed to support real-time data streams consisting of fixed-size data packets sent at periodic intervals, such as T1/E1 and Voice over IP without silence suppression. It has the following four QoS parameters: Nominal Grant Interval (NGI), Tolerated Grant Jitter (TGJ), Maximum rate, Maximum Latency.
2) **Real-time Polling Service (RTPS)** : The RTPS is designed to support real-time data streams consisting of variable-sized data packets that are sent at periodic intervals, such as MPEG video. It has the following QoS parameters: Nominal Polling Interval (NPI), Tolerated poll Jitter (TPJ), Maximum rate, Minimum rate, Maximum Latency.

A request specifies a source and destination node along with its QoS requirements. The k^{th} UGS request is denoted as C_{ijk}^{ugs} and its source node is i and destination node is j. The path it takes is represented by P_{ijk}. P_{ijk} is an ordered list of nodes starting with i and ending with j. Any two consecutive nodes in the list represent the link (joining the two nodes) which is part of the path. Thus, the path is defined by the ordered list of links represented by taking pair of consecutive nodes in the list. Note that path P_{ijk} is determined by the routing algorithm used in the network (described in Section IV). Corresponding QoS parameters of the request, nominal grant interval, tolerated grant jitter, maximum rate and maximum latency are denoted as ngi_k^{ugs}, tgj_k^{ugs}, $R\_max_k^{ugs}$ and $d\_max_k^{ugs}$ respectively.

Similarly, we denote k^{th} RTPS request as C_{ijk}^{rtps} which takes a path P_{ijk} where its source and destination nodes are i and j respectively. Corresponding QoS parameters of the request, nominal polling interval, tolerated poll jitter, maximum rate, minimum rate and maximum latency are denoted as npi_k^{rtps}, tpj_k^{rtps}, $R\_max_k^{rtps}$, $R\_min_k^{rtps}$ and $d\_max_k^{rtps}$ respectively. The total number of ongoing UGS and RTPS requests are denoted as N^{ugs} and N^{rtps} respectively.

Much of the analysis of UGS and RTPS requests are same. Hence, we will sometimes denote parameters without having ugs or $rtps$ superscript, which would indicate that the treatment is same for both the type of requests.

VI. Scheduling of Requests

Since requests in WiMax have stringent QoS requirement, they have to be carefully scheduled. In this work, we concentrate on TDD based WiMax system and hence the resource to be allocated by the scheduling module is time slots in a WiMax frame. Finding an optimal feasible centralized scheduling in a mesh network is an NP-hard problem. Hence, in this section we describe an efficient scheduling algorithm. Before we go into the details of the slot allocation procedure, we present some preliminary concepts behind the slot allotment process.

A. Preliminaries

First, we need to define few terms used in our analysis.

HyperInterval : HyperInterval of UGS (RTPS) requests is the LCM of ngi (npi) of all UGS (RTPS) requests in the network. That is,

$$HyperInterval^{ugs} = LCM(ngi_k^{ugs}) \quad 1 \leq k \leq N^{ugs} \quad (2)$$

$$HyperInterval^{rtps} = LCM(npi_k^{rtps}) \quad 1 \leq k \leq N^{rtps} \quad (3)$$

HyperInterval of all the requests (UGS and RTPS) is given by

$$H = LCM(HyperInterval^{ugs}, HyperInterval^{rtps}) \quad (4)$$

Since different UGS requests will have different ngi and the scheduling algorithm has to make sure that the packets arriving in every ngi meet their QoS constraints, HyperInterval is used while computing the schedule. Essentially, if the QoS requirement of a request is satisfied in every ngi within a HyperInterval, then QoS constraint of every packet belonging to any ngi can be guaranteed.

Offset : A subscriber station should transmit a packet of a request at the the beginning of ngi to minimize its delay. But the initial slots in a ngi may not always be available for the request because it may have been allotted to some other request. Thus, the packet of the request is scheduled in the first time slots available from the start of ngi which satisfies some scheduling constraints (described later). The distance (measured in slots) between its first scheduled timeslot and the beginning of ngi (when a packet arrived) is referred to as *offset*. In each ngi we leave tgj number of slots (from end of ngi going backwards) so that there is room for changing the slot allocation within the allowable jitter of the request[1]. As we will describe later, offset is only applicable to the penultimate node along the path of the request. Offset of request C_{ijk} is denoted by $offset_k$. Figure 2 illustrates a typical frame structure in which a request has offset 2, ngi 10 and tgj 5. Maximum value of offset allowed for request C_{ijk} is $offset_k^{max} = ngi_k - tgj_k$.

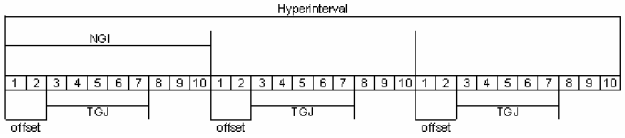

Fig. 2. Typical UGS request

When a new request C_{ijk} arrives, its path P_{ijk} is determined by the routing algorithm. We divide the path into two parts: path from source node i to penultimate node p (denoted as P_{ipk}), and path from penultimate node p to the destination node j (denoted as P_{pjk}). If i is also the penultimate node (j is the immediate neighbor of i in P_{ijk}), then the request is handled little differently. When link l_{xy} is in the path of the request C_{ijk} and when we deal with scheduling on link l_{xy} for the request, then the request is denoted as $C_{ijk}\{xy\}$.

Partitioning of the path implies that the original request C_{ijk} is also logically partitioned into two parts, C_{ipk} (request from source i to penultimate node p) and C_{pjk} (request from penultimate node p to destination j). The advantage of this partition is that the first part of the request does not need to worry about jitter constraint, it only needs to meet its part of delay constraint (partitioning of delay constraint is explained in the next paragraph). The second part of the partitioned connection is responsible for meeting the jitter constraint. Thus, dividing the request into two parts, it is easy to take care of the QoS constraints: intermediate nodes only need to fulfill the delay requirement whereas the penultimate node needs to only make sure that the jitter at the receiver is within the required limit. This decoupling of delay and jitter constraint by partitioning the request provides more flexibility to our scheduling algorithm.

Both UGS and RTPS requests have maximum latency constraint (d_max), which means end-to-end delay should be less than d_max. Let us denote d_{ipk} to be the delay from source node i to penultimate node p and d_{pjk} to be the delay from penultimate node p to destination j. Let d_max_k be the maximum latency of request C_{ijk}. Then the maximum latency constraint can be expressed as

$$d_{ipk} + d_{pjk} \leq d\_max_k \quad if \ i \neq p \quad (5)$$

$$d_{ijk} \leq d\_max_k \quad if \ i = p \quad (6)$$

Since end-to-end delay bound is divided into two parts, one for the path from source to penultimate node and the other from the penultimate node to the destination node, we need to find the maximum delay that a request can incur in each part so that the resultant end-to-end delay and jitter satisfies the QoS constraints of the original request.

For request C_{ijk}, If source node is also the penultimate node (i.e., $i = p$), then

$$d_{ijk}^{max} = d\_max_k \quad if \quad i = p \quad (7)$$

But for request C_{ijk} if source node is not the penultimate node ($i \neq p$), then

$$d_{ipk}^{max} = \begin{cases} d\_max_k - tgj_k & \text{if } (d\_max_k \% ngi_k) > tgj_k \\ n * ngi_k - tgj_k & \text{otherwise} \end{cases} \quad (8)$$

$$d_{pjk}^{max} = tgj_k \quad (9)$$

where $n = \lfloor d\_max_k / ngi_k \rfloor$.

Note that the maximum delay above is based on the fact that the jitter at the destination has to be less than tgj_k. Thus, it leaves a slack of duration tgj_k for the penultimate node within which it can choose the required number of slots for the request without violating jitter constraint.

Knowing the delay requirement of each partitioned part of the request C_{ijk}, we now look at the limits in slot allotment. Specifically, we look at what is the allowable start slot and end slot so that the request satisfies its QoS requirements. For the path P_{ipk}, let the allowable start and end slot be denoted as S_{ipk} and E_{ipk} respectively. This means that the required number of slots have to be allocated to this part of the request along each link of path P_{ipk} between S_{ipk} and E_{ipk}. Note that the slot allocation has to happen for every ngi (since packets arrive in every ngi). For the m^{th} ngi,

$$S_{ipk}^m = m * ngi_k \quad (10)$$

$$E_{ipk}^m = S_{ipk}^m + d_{ipk} \quad (11)$$

Note that in the above equation we have used d_{ipk} which indicate the value of actual delay (not the maximum delay) after slot allocations are done. In the CAC algorithm, d_{ipk} is set to d_{ipk}^{min} at the beginning and changed in every iteration until a schedule is obtained (see CAC algorithm in Algorithm 2). Once d_{ipk} is set then its value cannot be changed throughout the lifetime of the request. Otherwise it may lead to violation of delay jitter constraint.

The corresponding limits at the penultimate node is given by

$$S_{pjk}^m = m * ngi_k + offset_k \quad (12)$$

$$E_{pjk}^m = S_{pjk}^m + tgj_k \quad (13)$$

[1]slot allocation for existing requests may need to be changed so that new requests can be admitted.

B. Details of Slot Allotment

In this section, we present our slot allocation procedure which will find a feasible slot assignment for the requests and will also satisfy their respective delay and delay jitter requirements.

When a new request arrives and the CAC algorithm is executed, then all the existing requests along with the new request have to go through the slot allocation procedure. For existing requests there is already a slot assignment which was used in the last scheduling interval and we refer to it as *previous slot allotment*[2].

While determining scheduling of a WiMax frame on a particular link, a slot is assigned to a particular request for which the link is in the path of the request. A slot can be assigned to multiple links (belonging to the same request or different requests) as long as links are non-blocking with respect to each other. Thus, the transmitter nodes of the links would transmit at the same time (in the same slot) and the data would be received correctly by the receiver nodes of the links. We will refer to the slot as *eligible slot* for the links.

We illustrate this by an example. Consider the mesh network shown in Figure 3. Table I shows slot assignment at some instant. Now, consider a new request C_{454} arrives. So we have to find slot for $C_{454}\{45\}$. For this request, Slot 1 is ineligible slot but Slot 2 and Slot 3 are eligible slots. Because in Slot 1 link l_{12} is active which blocks link l_{45}. Note that any empty slot is trivially an eligible slot. So, Slot 4 and Slot 5 are also eligible for $C_{454}\{45\}$.

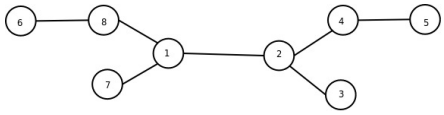

Fig. 3. Example Illustrating Eligible Slot

Slot 1	Slot 2	Slot 3	Slot 4	Slot5
$C_{131}\{12\}$	$C_{131}\{23\}, C_{862}\{86\}$	$C_{713}\{71\}$		

TABLE I

AN EXAMPLE SLOT ASSIGNMENT

Our slot allocation procedure is a four step process as described below.

1) Calculate the Hyperinterval (H) of existing and the new request H.
2) Every request C_{ijk} is logically split into C_{ipk} and C_{pjk}. For all $m = 1\ to\ (H/ngi_k)$, compute S^m_{ipk}, E^m_{ipk} and S^m_{pjk}, E^m_{pjk}.
3) For all request k sort all the E^m_{ipk} and E^m_{pjk} in the non-decreasing order.
4) Starting from the request having smallest E^m, allocate required number of slots to the corresponding request. Note that if E^m_{ipk} is being considered, then slot allocation is done for the request for the m^{th} ngi on all the links from source i to the penultimate node p starting at slot location S^m_{ipk}. Allocation over all the links on the paths should be done before E^m_{ipk}, otherwise the request is not schedulable. Similarly, if E^m_{pjk} is considered, then the slot allocation on link l_{pj} should start from S^m_{pjk} and should end before E^m_{pjk}.

[2]Slots belonging to the requests which left the system are marked free in *previous slot allotment*.

Step 4 discussed above is actually a complex step and needs to follow certain constraints. The allocation for request C_{ijk} starts from S^m_{ipk} and each slot from the starting point is looked at and is either allocated to the request or left out depending on the following constraints.

- If the request under consideration is a new request, then slot is allocated to the request if this slot is an eligible slot for the new request.
- If a slot under consideration was allocated to the current request in the previous slot allotment and this slot is eligible slot for this request, then the slot is allotted to the request.
- If in the previous slot allotment this slot was not allocated to this request, but the total number of slots allocated to this request until this slot in the current allotment is more than (or equal to) that in the previous allotment, then the slot will be allocated to this request if this is an eligible slot in the current as well as in the previous slot allotment. This is so because it is known that the number of remaining alloted slots in the previous slot allotment is more (or equal) than current deficiency of the request. Hence this request will at least be able to grab those slots from previous allocation to satisfy its slot requirement. So, this slot will be alloted to a request only if it does not adversely affect the slot allotment of any other request.
- If in the previous slot allotment this slot is not allotted to this request but the total number of slots allocated to this request until this slot in the current allotment is less than that in the previous allotment, then the slot will be allocated to this request if it is an eligible slot in current allocation[3]. The number of alloted slots is less because some of the slots assigned to this request in previous allotment are taken away by new request directly or indirectly, so this request should be allowed to fulfill its requirement as early as possible.

The reason for these constraints is to ensure that all the admitted requests get minimum number of required slots in every round of slot allotment. By these conditions, every request is guaranteed to get slots which was allotted to the same request in previous slot allotment round until a new request takes away its slot directly or indirectly[4].

Above constraints are considered while allocating slots for both the parts (C_{ipk} and C_{pjk}) of the request C_{ijk}. But there are some more conditions which are specific to the first part of the request (C_{ipk}) as described below.

- The path of C_{ipk} may have more than one link. So, demand for all the links along the path has to be satisfied. Let us assume that the path of C_{ipk} is $i - x - p$. First, slot is allotted for link l_{ix} (starting from S^m_{ipk}). When its demand is fulfilled then we move on to next link l_{xp}. For this link, slots after the last slot allocated to link l_{ix} are considered for link l_{xp}.
- Delay of C_{ipk} as given by (8), for some cases may extend beyond the limit of HyperInterval. Especially, packets which arrive towards the later part of Hyperinterval (note that there will be multiple ngi's in one HyperInterval) may have their delay bound beyond the HyperInterval. Allocation of slots to such packets may happen in the

[3]Slot which was ineligible in the previous slot allotment becomes eligible now because some requests may have left the system.

[4]A new request can take away a slot of an existing request C_{ijk} and the request C_{ijk} can take away a slot from another existing request C_{xyz}.

subsequent line of the HyperInterval as shown in Table II and III. A slot which is allocated to a request in the second line indicates that the packet being transmitted in that slot was actually generated in the previous HyperInterval.

- When slots are allotted across multiple lines then there are certain conditions which need to be satisfied. Table II and III show two slot allotment for a request along the links $1-2-3$. In the tables, an X means the slot cannot be allocated to this request. In the previous slot allotment shown in Table II, node 1 sends data in slots 5 and 6 and then node 2 sends the packets in next Hyperinterval in slots 1 and 2. Thus, maximum delay observed by packets is equal to 8 slots. Let us assume maximum allowed delay for the request is 10 slots. Now, in the current slot allotment shown in Table III, some of the busy slots (marked as X in the previous allotment) are no longer busy. Hence, node 1 is allocated slots 3 and 4 and node 2 is allocated slots 5 and 6. Both the slot allotments might look fine separately. But when viewed together, one can observe violation of delay constraint. Consider a packet generated in the previous slot allotment (Table II) is at node 2 at the end of the Hyperinterval. Now in the next Hyperinterval, this packet will get slot 5 and 6 to be transmitted to node 3. This lead to a total delay of 12 slots which is more than the maximum allowed delay. So, we cannot allow such allotment, i.e., when a slot is allotted to a link of a request at i^{th} line of the HyperInterval, in the new allotment it cannot take up slots in the lines above the i^{th} line.

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6
X	X	X	X	$C_{131}\{12\}$	$C_{131}\{12\}$
$C_{131}\{23\}$	$C_{131}\{23\}$	X	X		

TABLE II

AN EXAMPLE OF PREVIOUS SLOT ALLOTMENT

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6
X	X	$C_{131}\{12\}$	$C_{131}\{12\}$	$C_{131}\{23\}$	$C_{131}\{23\}$

TABLE III

AN EXAMPLE OF CURRENT SLOT ALLOTMENT

VII. CALL ADMISSION CONTROL (CAC)

In WiMax mesh network the CAC module needs to make sure that a request gets enough number of slots at each link along the path from source to destination so that QoS requirements of the request is satisfied. One of the algorithms described in Section IV is used to determine the path from source node to the destination node. Thus, CAC and scheduling modules work closely with each other to meet the above requirement.

A. CAC Module

In this section, we present our CAC algorithm shown in Algorithm 2 which is executed when a new request C_{ijk} arrives. The CAC module runs on all previously accepted requests and the new request.

First the CAC module finds the path for C_{ijk}. Then, as discussed earlier, the new request is split into two parts, C_{ipk} and C_{pjk}. The CAC module first deals with the partitioned request C_{ipk}. Then d_{ipk} is initialized to d_{ipk}^{min} which is given by

$$d_{ipk}^{min} = R\_min_k^{inslots} * N_{ipk} \quad (14)$$

where $R\_min_k^{inslots}$ denotes the minimum number of slots required by the request C_{ijk} along each link (which is $R\_max_k^{ugs}$ if the new request is an UGS request or $R\_min_k^{rtps}$ if it is an RTPS request), and N_{ipk} denotes the number of links in the path from node i to p (P_{ipk}).

Now CAC module finds a feasible schedule for all existing requests as well as for the partitioned new request C_{ipk} using the slot allotment procedures discussed in the previous section. If CAC module does not find the feasible schedule then it means that at least one request had a shortage of slots. Hence, it increases the value of d_{ipk} by an amount equal to the maximum of these shortages. Then it repeats the process of slot allotment again. If $d_{ipk} > d_{ipk}^{max}$, then CAC module removes edge corresponding to the bottleneck link from the graph and repeats the whole process.

If a feasible schedule is found for C_{ipk}, then the CAC module tries to find feasible slot allotment for C_{pjk}. Recall

Algorithm 2 Call_Admission_Control Algorithm

```
1: while source node i and destination node j is connected do
2:   find the path from source to destination for the new request (using a routing algorithm)
3:   Let p = penultimate node in the path of C_ijk
4:   if i ≠ p then
5:     Divide C_ijk into two parts C_ipk and C_pjk
6:     Set d_ipk = d_ipk^min /* as per Eqn (14) */
7:     while d_ipk ≤ d_ipk^max do
8:       Find Slot Allotment for all existing request and C_ipk
9:       if all the requests are schedulable then
10:        Store the current values of d_ipk and exit while Loop
11:      else
12:        /* at least one request had a shortage of slots, so it could not be scheduled */
13:        max_shortage = maximum of all the shortages (of different requests)
14:        d_ipk += max_shortage
15:      end if
16:    end while
17:  else
18:    d_ipk = 0
19:    d_ipk^max = 0
20:  end if
21:  if d_ipk ≤ d_ipk^max then
22:    d_pjk = tgj_k
23:    offset_k = d_ipk % ngi_k ; flag = 0
24:    while ((offset_k + flag * ngi_k) < (ngi_k + d_ipk%ngi_k)) AND (d_ipk + d_pjk ≤ d_max_k) do
25:      Find the Slot Allotment for all existing requests and C_ipk and C_pjk
26:      if all the requests are schedulable then
27:        Store the values of d_ipk & offset_k.
28:        /* These parameters will not change in future for this request*/
29:        Accept the new request and exit
30:      else if i ≠ p then
31:        increment offset_k by one
32:        if offset_k > (ngi_k − tgj_k) then
33:          Set offset_k = 0 ; flag_k = 1
34:        end if
35:      end if
36:      if i ≠ p then
37:        d_ipk = ⌊d_ipk/ngi_k⌋ * ngi_k + offset_k + flag * ngi_k
38:      end if
39:    end while
40:  end if
41:  Remove the edge corresponding to the bottleneck link from the graph
42: end while
43: Reject the request
```

that offset is used for scheduling at the penultimate node only. First, $offset_k$ is initialized to $d_{ipk}\%ngi_k$. This is because data would arrive at the node p only at d_{ipk} and hence the transmission of data cannot begin before that. Thus, the initial offset value is set to the point at which d_{ipk} ends. This is depicted in Figure 4. But the offset value can only go up to $ngi - tgj$ as discussed earlier and hence the allowable offset value is between A and B.

After initializing $offset_k$, CAC module tries to find the feasible schedule considering all the existing requests and the two components of the new request, C_{ipk} and C_{pjk}. If it finds the schedule then CAC module accepts the request. Otherwise, it increments the value of offset by one and repeats the slot allotment process again. If a feasible allocation is not found (until offset takes the maximum possible value), then offset is initialized to zero (Step 33) and incremented until $d_{ipk}\%ngi_k$ (this is between C and D in Figure 4). Since this interval was not explored in the previous iteration, the CAC tries to find if this interval would result in a feasible slot allocation. In this case, the data transmission starts in the next ngi and this fact is captured by setting $flag$ to 1 (Step 33). Finally, d_{ipk} is extended to the end of $offset_k$ (as depicted in Figure 5), since data transmission does not start until end of $offset_k$. This increases the delay bound for C_{ipk} and hence provides more flexibility for slot allocation of C_{ipk}.

VIII. Methods for Allocating extra bandwidth to RTPS requests

Any RTPS request C_{ijk}^{rtps} has two bandwidth parameters $R\_min_k^{rtps}$ and $R\_max_k^{rtps}$. When an RTPS request is admitted, the CAC algorithm (presented in the previous section) makes sure that the new request is considered for admission with $R\_min_k^{rtps}$ and all the existing requests are schedulable. But the request C_{ijk}^{rtps} can change its bandwidth requirement between $R\_min_k^{rtps}$ and $R\_max_k^{rtps}$ during its lifetime.

A. Bandwidth Estimator

We propose a Bandwidth Estimator module in the WiMax system which can monitor the RTPS request queues and based on the queue length can predict the bandwidth requirement of the requests and then the SS can send extra bandwidth request to the BS. It calculates the number of packets generated in a particular npi by a request. If number of packets generated is more than what the currently granted rate, then then it calculates the extra bandwidth required to meet the current packet generation rate. This newly calculated bandwidth request is then sent to the BS. If the request is granted then the extra bandwidth (extra slots) are allocated to the request in the next scheduling period.

Now, we describe some of the methods to fulfill extra bandwidth requirement of RTPS requests.

- Grant Request Only R_min (GRMin): In this method, RTPS requests are eligible to get bandwidth equivalent to its R_min only even if they demand for higher bandwidth. This would enable the system to admit more number of requests, but at the cost of higher latency and packet loss of the RTPS requests.
- Grant Request at R_max (GRMax):In this method RTPS requests are admitted at R_max. So, this method will result in lesser number of requests getting admitted, but RTPS requests will have lower latency and no packet loss.
- Grant only Extra Required Bandwidth (GERB): In this method, SS sends bandwidth request which exactly meets the extra requirement computed by the bandwidth estimator. Initially, the requests are admitted at R_min and later when bandwidth estimator detects a higher packet arrival, SS sends bandwidth request corresponding to the higher packet arrival rate. But the problem with this method is that the queue can build up by the time the request is granted by the BS. Hence, the packets may experience high delay.
- Grant Extra Bandwidth and Reduce Latency (GEBRL): In this method, the extra bandwidth requested by SS is calculated based on the extra demand due to higher packet arrival rate and considering reduction of data queue size of the request which builds up due to the lag in sending the extra bandwidth request and getting the corresponding bandwidth grant. So, basically the bandwidth request sent out to the BS is more than what would be required to meet the increase in packet arrival to account for flushing of data queue of the request. The algorithm for this method is shown in Algorithm 3. In this algorithm we denote *extra_demand* as the demand over and above the R_min required by the request. Currently granted rate is denoted as *current_rate* and the extra demand request which should be made to satisfy the $extra\_demand$ is represented as *extra_demand_req*. There are two tunable parameters A and B which are used to increase or decrease the extra bandwidth request respectively. Both the parameters have value between 0 and 1. q_size represents the number of packets in the queue of a particular request and $demand\_grant$ is boolean variable which tells whether $extra\_demand\_req$ is granted last time or not.

Obviously GEBRL is the most complex method but would provide better performance. This method is implemented using Algorithm 3. Whenever incoming packet rate increases, the algorithm sends a bandwidth request for the extra bandwidth, which if granted, makes the current rate equal to the packet arrival rate (Step 5). But, in the next round if the queue builds up and if the previous request was granted, then it asks for more bandwidth, which is a fraction of R_min. This is controlled by a tunable parameter A (Step 19). But if the previous request was not granted, then it means that the system is running at high utilization and hence it actually reduces the extra bandwidth request (Step 15). Here a tunable parameter B is used to reduce it by a fraction of R_min. Notice that as soon as the queue length becomes zero, the algorithm produces a negative $extra\_demand\_req$ which means the SS asks for a reduction in its bandwidth allocation (Step 9).

IX. Simulation Experiments

A. Simulation Parameters

We have developed a simulator using Java sdk 1.5 to evaluate the performance of various algorithms and methods presented in this paper. We created a mesh network of 25 nodes by choosing the nodes randomly. Links are placed in the network in a random manner until all the nodes are connected. We assume that we have 200 slots in a data subframe and duration of each frame is 30 msec. Arrival of requests is Poisson distributed with mean arrival rate of λ. Lifetime of requests is exponentially distributed with mean lifetime of μ. For RTPS requests, the extra demand bandwidth is uniformly distributed between R_min and R_max. The new

Algorithm 3 Grant_Extra_Bandwidth_and_Reduce_Latency

/* $bw\_to\_flush\_queue$, $current\_rate$ and $demand\_grant$ are global variables, initialized to 0, R_min and $FALSE$ respectively */
/* $extra\_demand$ is the extra bandwidth over R_min required by the request (computed by the bandwidth estimator) */
/* SS should set $demand\_grant$ to TRUE or FALSE depending on whether the previous bandwidth request was granted or rejected by the BS */
if $demand\_grant == TRUE$ **then**
  $current\_rate = current\_rate + extra\_demand\_req$
end if
if $(current\_rate > R\_min + extra\_demand)\ AND\ (q\_size = 0)$ **then**
  /* Send Request to decrease the $current\_rate$. $extra\_demand\_req$ would be negative in this case */
  $extra\_demand\_req = R\_min + extra\_demand - current\_rate$
else if $current\_rate < R\_min + extra\_demand$ **then**
  $extra\_demand\_req = R\_min + extra\_demand - current\_rate$
end if
if $demand\_grant == FALSE\ AND\ q\_size \neq 0$ **then**
  /* since previous request was not granted, reduce the additional bandwidth being asked to flush the queue */
  $bw\_to\_flush\_queue = max(0,\ bw\_to\_flush\_queue - B * R\_min)$
  $extra\_demand\_req = bw\_to\_flush\_queue$
else if $q\_size \neq 0$ **then**
  /*demand request was granted in the previous scheduling interval, so we can increase the request for additional bandwidth to flush the queue */
  $bw\_to\_flush\_queue = min(R\_max - currentrate,\ bw\_to\_flush\_queue + A * R\_min)$
  $extra\_demand\_req = bw\_to\_flush\_queue$
end if
if $extra\_demand\_req > 0$ **then**
  send a bandwidth request of amount $extra\_demand\_req$ to BS
else
  $demand\_grant$ = $TRUE$
end if

extra bandwidth demand stays in effect for a random duration. This duration is exponentially distributed with mean γ. Both the parameters A and B used in Algorithm 3 are set to 0.1. Simulation period for each simulation run is 500 sec. The parameters used for UGS and RTPS requests are shown in Table IV and Table V respectively.

ngi (in msec)	tgj (in msec)	R_max (in slots)	μ (in mins)	d_max (in sec)
20	10	10	3	100
30	15	10	3	150
40	20	10	3	150

TABLE IV
PARAMETERS USED FOR UGS REQUESTS

npi (in msec)	tpj (in msec)	R_min (in slots)	R_max (in slots)	μ (in mins)	d_max (in sec)	γ (in sec)
20	10	10	20	3	100	10
30	15	10	20	3	150	20
40	20	10	40	3	200	10

TABLE V
PARAMETERS USED FOR RTPS REQUESTS

B. Study of Different Routing Methods

We ran simulation to compare the performance of different methods for computing routing paths (discussed in Section IV). The performance metric used for comparison is *acceptance ratio*, defined as the ratio of number of requests admitted by the CAC to total number of requests. Figure 6 plots acceptance ratio versus request arrival rate for these different methods. Load Balanced Weighted Shortest Path with Retry (LBWSPR) method performs the best, whereas Routing Tree Rooted at BS (RTRB) method performs the worst. In LBWSPR method, paths are chosen such that links are balanced with respect to load. Hence, it leads to much better performance than other methods. In RTRB method, even if there may be shorter paths available, it has to choose a longer path passing through the BS. Thus, this method performs the worst.

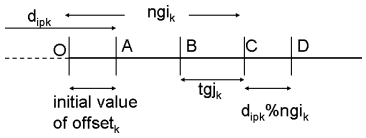

Fig. 4. Illustration of Calculation of $offset_k$

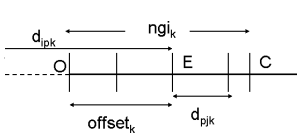

Fig. 5. Final Adjustment of d_{ipk} after $offset_k$ is Fixed

C. Comparison between different Scheduling Algorithms

In this section, we present comparison between different scheduling algorithms. Scheduling algorithms proposed in [6], [7] assume data is transmitted only from SS to BS station. So, for a fair comparison, we simulated our scheduling algorithm with traffic flowing from SS to BS. We label our scheduling algorithm as *Load Balanced Multipath Scheduling*. The scheduling algorithm proposed in [7] is labeled as *Interference Aware Routing* and that in [6] as *Multipath Scheduling*.

In Figure 7, acceptance ratios of different scheduling algorithms as arrival rate of requests increases, are compared. Our algorithm shows lower performance based on this metric compared to algorithms described in [6] and [7]. However, we found that requests meet all their QoS constraints when scheduled using our algorithm whereas many request fail to meet their QoS constraints using the other two algorithms. Hence acceptance ratio is not a fair metric for comparison. So, we define a metric, *compliance ratio*, as given below.

$$Compliance\ Ratio = R_d/R \tag{15}$$

Here, R_d is the number of requests that meet all the constraints of the respective requests. R is the total number of requests arriving to the system. Constraints of an UGS request are rate, latency and jitter, whereas an RTPS request has the constraint of meeting rate of R_min along with its latency and jitter. Comparison of different scheduling algorithm on the basis of Compliance Ratio is shown in Figure 8. It is clear from the figure that our scheduling algorithm performs much better compared to the other two algorithms. This is primarily because our scheduling algorithm provides hard guarantees to the requests and hence none of them violate their QoS constraints. But, since our algorithm is strict about QoS constraint it admits fewer requests compared to the other two. As an aside, readers can notice (from Figure 7 and Figure 8) that the acceptance ratio and compliance ratio are same for our algorithm, since our algorithm accepts connections only if it can guarantee it QoS.

D. Validation of QoS Guarantee

Next we present our results which shows that each and every packet belonging to different admitted requests meets its delay and jitter requirement. For this experiment, we have kept all other parameters of both UGS and RTPS requests same (as given in Table IV and Table V), but d_max of all requests

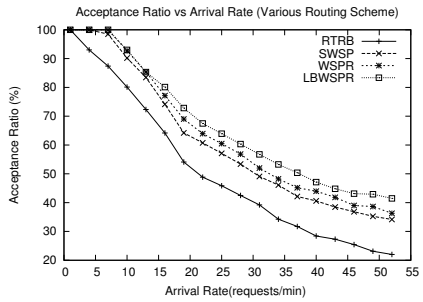

Fig. 6. Comparison Among Different Routing Path Methods

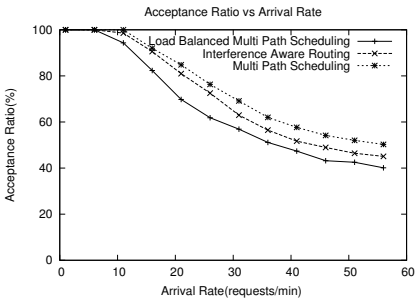

Fig. 7. Acceptance Ratio vs Arrival Rate

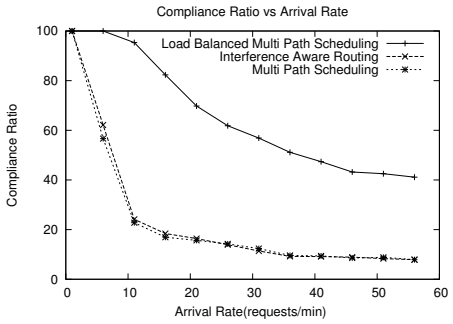

Fig. 8. Compliance Ratio vs Arrival Rate

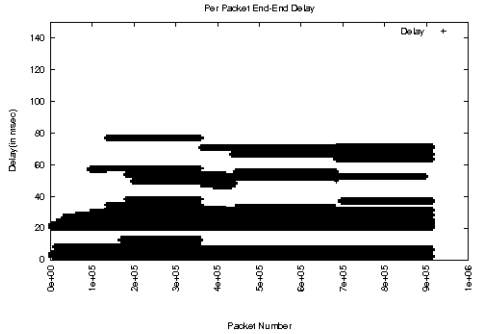

Fig. 9. Per Packet Delay Analysis

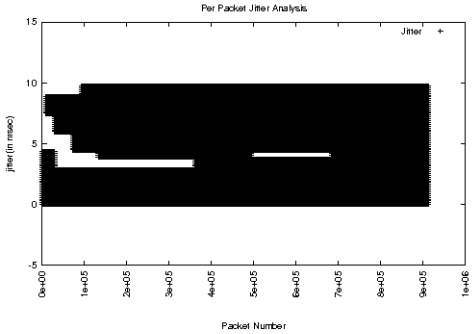

Fig. 10. Per Packet Jitter Analysis

were set to 100 msec and tgj and tpj were set to 10msec. It can be confirmed from Figure 9 and 10 that delays of all the packets are less than 100msec and jitter values of all the packets are less than 10msec. This validates the correctness of our CAC algorithm in terms of providing strict QoS guarantees to every packet.

E. Average Number of Active Links per Slot (ANALS)

In this section, we show the variation of Average Number of Active Links per time Slot (ANALS) at different request arrival rate for networks having different number of nodes (Figure 11). This graph essentially shows how our scheduling algorithm can increase the network capacity by having multiple simultaneous data transmissions. It is clear from the figure, as the size of network (N) grows, the value of $ANALS$ also increases which implies that the network throughput increases.

F. Comparison of Different Methods for Allocating Extra Bandwidth to RTPS Requests

In this section, we present comparison among different methods used for allocating extra bandwidth to RTPS requests. This comparison is done using different metrics as described below.

The first metric used is the Acceptance Ratio and the comparison is shown in Figure 12. If requests are admitted at R_min and they are not provided any extra bandwidth above R_min (GRMin method), then the Acceptance Ratio of this scheme is the highest. On the other hand if requests are admitted at R_max (GRMax method), then the Acceptance Ratio of this scheme is the lowest. Performance of the other two schemes lies in between these two schemes.

The next metric we use is the percentage of packet drop. Here packet drop refers to the packets that the receiver drops because the packet did not meet its delay or jitter constraint.

The simulation for this is done in two scenarios.

- Scenario 1: In the first scenario, there are RTPS requests with the following parameters :

npi (in msec)	tpj (in msec)	R_min (in slots)	R_max (in slots)	μ (in mins)	d_max (in sec)	γ (in sec)
20	10	10	20	3	200	10
30	15	10	20	3	200	10

- Scenario 2: In the second scenario, there are RTPS requests with the following parameters :

npi (in msec)	tpj (in msec)	R_min (in slots)	R_max (in slots)	μ (in mins)	d_max (in sec)	γ (in sec)
20	10	10	40	3	200	50
30	15	10	40	3	200	50

The main difference between the two scenarios is that in Scenario 2, the variation between between R_min and R_max is much higher than in Scenario 1.

From Figure 13 and 14 it is clear that for both the scenarios packet drop is maximum if the requests are admitted at R_min and no extra bandwidth is granted to the requests (GRMin method) and packet drop is zero when the requests are admitted at R_max (GRMax method). In Scenario 1, the performance of GERB is almost same as GEBRL. Since the variation in bandwidth requirement is not high, there is not much backlog when GERB method is used. But, in scenario 2, GEBRL performs much better than GERB, since it is able to reduce backlog by asking for more bandwidth than required by the request so as to flush the backlog.

Finally, we compare various methods using average end-to-end delay as the metric (Figure 15 and 16). Here we use the same two scenarios as used while comparing packet drop. When the requests are admitted at R_max (GRMax method) then the avg end-end delay is minimum. The avg end-to-end delay for the case when requests are admitted at R_min (GRMin method) is also low. In GRMin method, there are packets which do not meet the delay constraints which are

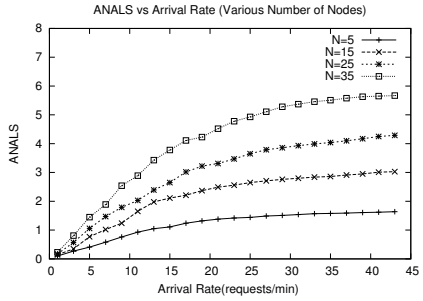

Fig. 11. Average Number of Active Links per slot ($ANALS$)

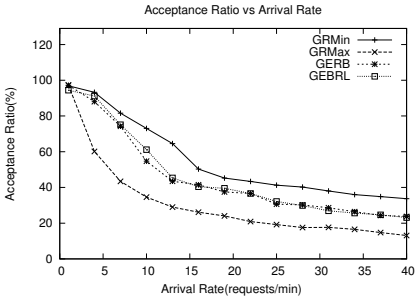

Fig. 12. RTPS Extra Bandwidth Allocation Scheme: Acceptance Ratio vs Arrival Rate

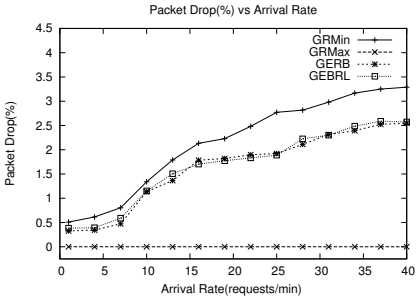

Fig. 13. RTPS Extra Bandwidth Allocation Scheme: Packet Drop vs Arrival Rate (Scenario 1)

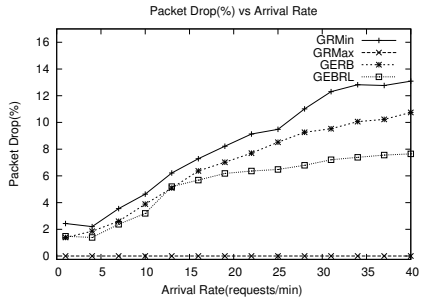

Fig. 14. RTPS Extra Bandwidth Allocation Scheme: Packet Drop vs Arrival Rate (Scenario 2)

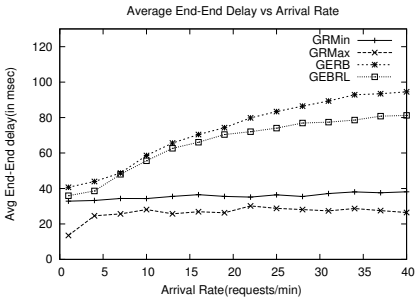

Fig. 15. RTPS Extra Bandwidth Allocation Scheme: Avg. Delay vs Arrival Rate (Scenario 1)

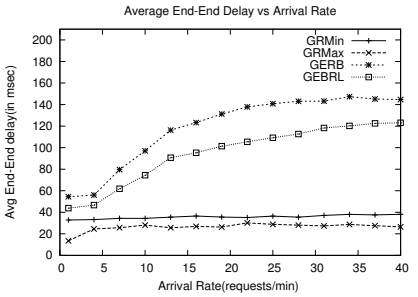

Fig. 16. RTPS Extra Bandwidth Allocation Scheme: Avg. Delay vs Arrival Rate (Scenario 2)

dropped by the SS and hence, the average delay is computed over the packets which meet the delay constraints. The relative performance of GEBRL over GERB method is better in both the scenarios due to the fact that GEBRL tries to flush the queue by asking for more bandwidth than required by the requests.

X. Conclusion and Future Work

In this paper, we have presented two efficient routing algorithms and a centralized scheduling algorithm for IEEE 802.16 mesh networks for UGS and RTPS service class. The proposed centralized scheduling algorithm uses a novel idea of splitting a request into two parts. The first part takes care of delay constraint and not worry about jitter constraint whereas the second part is responsible for jitter constraint. We have proposed a CAC algorithm which works closely with the scheduling algorithm to make sure that requests are admitted such that every packet of the request meets its delay and jitter requirements. For RTPS requests, we have also proposed different methods for allocating extra bandwidth to RTPS requests.

In this study, we have assumed that channel condition does not change and that the nodes are fixed. We intend to extend this study to varying channel condition and mobile nodes as future work. We have not considered node or link failures in the current study. It will be interesting to see how our proposed algorithm can be modified to handle such failures.

References

[1] "IEEE Standard for local and metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems," *IEEE Standard 802.16-2004*, October 2004.

[2] P.-H. Hsiao, A. Hwang, H. Kung, and D. Vlah, "Load-balancing routing for wireless access networks," *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 986–995, 2001.

[3] J. Gao and L. Zhang, "Load-balanced short-path routing in wireless networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 4, pp. 377–388, April 2006.

[4] A. Raniwala and T. cker Chiueh, "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 2223–2234 vol. 3, March 2005.

[5] J. Tao, F. Liu, Z. Zeng, and Z. Lin, "Throughput enhancement in WiMax mesh networks using concurrent transmission," *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, vol. 2, pp. 871–874, Sept. 2005.

[6] H.-Y. Wei, S. Ganguly, R. Izmailov, and Z. Haas, "Interference-aware IEEE 802.16 WiMax mesh networks," *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 5, pp. 3102–3106 Vol. 5, May-1 June 2005.

[7] Y. Cao, Z. Liu, and Y. Yang, "A centralized scheduling algorithm based on multi-path routing in WiMAX mesh network," *Wireless Communications, Networking and Mobile Computing, 2006. WiCOM 2006.International Conference on*, pp. 1–4, Sept. 2006.

[8] D. Ghosh, A. Gupta, and P. Mohapatra, "Admission control and interference-aware scheduling in multi-hop WiMAX networks," *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007. MASS 2007.*, pp. 1–9, Oct. 2007.

[9] R. Ramaswami and K. Parhi, "Distributed scheduling of broadcasts in a radio network," *INFOCOM '89. Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, IEEE*, pp. 497–504 vol.2, Apr 1989.

[10] A. Ephremides and T. Truong, "Distributed algorithm for efficient and interference-free broadcasting in radio networks," *INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pp. 1119–1124, Mar 1988.

[11] M. Cao, V. Raghunathan, and P. Kumar, "A tractable algorithm for fair and efficient uplink scheduling of multi-hop wimax mesh networks," *Wireless Mesh Networks, 2006. WiMesh 2006. 2nd IEEE Workshop on*, pp. 93–100, 2006.

[12] H. Shetiya and V. Sharma, "Algorithms for routing and centralized scheduling to provide QoS in IEEE 802.16 mesh networks," *in WMuNeP '05: Proceedings of the 1st ACM workshop on Wireless multimedia networking and performance modeling*, pp. 140–149, 2005.

[13] J. B. Chalke, "Scheduling and call Admission control (CAC) in IEEE 802.16 mesh networks," MTech Thesis, IIT Bombay, July 2008.