

An Efficient Call Admission Control for IEEE 802.16 Networks

Sarat Chandra and Anirudha Sahoo
Kanwal Rekhi School of Information Technology
Indian Institute of Technology, Bombay, Powai, Mumbai, India
Email: {sarat, sahoos}@it.iitb.ac.in

Abstract—IEEE 802.16 is a standard recommended by IEEE as a Wireless Metropolitan Area Network (WMAN) technology to provide connectivity to the *last mile* of a network. Not only does it provide high bandwidth but also it has a long transmission range (upto 30 miles). In addition, 802.16 MAC and physical layer are carefully designed to support different types of real time application by providing Quality of Service (QoS). But without proper scheduling and connection admission control (CAC), the system cannot provide promised QoS to the real time applications. Hence, scheduling and CAC play a vital role in 802.16 network. But the standard does not specify any scheduling or CAC. Although there are few 802.16 based QoS scheduling architectures proposed in the literature, most of them assume a conventional bandwidth based CAC (BW-CAC). But BW-CAC does not take QoS requirements (e.g., deadline) of connections into account. Hence, they would not be appropriate for real time application. In this paper, we propose an efficient QoS CAC that ensures that QoS guarantee of the connections are met. Using simulation, we present performance of our CAC in different scenarios and show that our CAC performs better than BW-CAC.

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 aims at providing broadband wireless *last-mile* access in a Metropolitan Area Network (MAN), with performance comparable to traditional cable, DSL, or T1 services. The most widely used Wireless technology, IEEE 802.11, can only be used in a LAN environment because its transmission range can only cover up to few hundred meters. While 802.16 has a transmission range of few kilometers, it also supports Quality of Service (QoS) by providing various service classes and by having high bandwidth. The service classes in 802.16 have been carefully designed to support real time applications like voice and video and non-realtime application like large file transfer. Besides, 802.16 based systems are becoming increasingly more feasible because of ease of deployment in remote areas where wireline connectivity would be prohibitively expensive. Thus, 802.16 network is a very attractive technology for providing integrated voice, video and data services in the last mile.

Different kinds of traffic supported by 802.16 network are classified into one of the following Services:(1) Unsolicited Grant Service (2) Real-time Polling Service (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 architecture. There have been few scheduling architectures reported in

the literature for 802.16 network [2], [3]. In addition to scheduling, Connection Admission Control (CAC) is also a very important part of 802.16 network, since the system is supposed to provide QoS guarantee. Without efficient CAC, 802.16 network will not be able to provide QoS guarantee to realtime applications like voice and video. To the best of our knowledge, there has been no architecture that clearly describes a CAC for IEEE 802.16 networks. Though some authors have suggested implicit conventional bandwidth based CAC, our study presented here, shows that such simple CAC cannot guarantee QoS to application services. Hence such primitive CAC may make the implementation non-compliant as well as unsuitable for application using different services of 802.16. Therefore, in this paper, we present an efficient CAC algorithm which not only provides bandwidth guarantee, but also ensures QoS guarantees to connections as per their service types. We compare performance of our CAC algorithm with the conventional bandwidth based CAC and show that our CAC performs much better than the bandwidth based CAC.

The remainder of this paper is organized as follows. Section II outlines the related work in the literature. Section III describes the overview of 802.16 network. Section IV introduces our QoS architecture followed by notations used in this paper in Section V. Section VI presents overview of QoS-CAC followed by pseudocode in Section VII. Section VIII describes a bandwidth estimation mechanism for RTPS and NRTPS connections. Section IX presents our simulation results followed by conclusion in Section X.

II. RELATED WORK

There have been few proposals presented in the literature to support QoS in IEEE 802.16 networks. For example, Chu et al. [4], proposed a QoS scheduling architecture for the MAC protocol in 802.16 networks. It is based on priority scheduling and dynamic bandwidth allocation. They have chosen GPSS (explained in Section III-A) mode for bandwidth grants. Though they proposed an architecture, the authors have not provided any simulation results that shows the effectiveness of the proposed architecture. In [5], the authors have proposed an uplink scheduling architecture to support QoS guarantees like bandwidth, delay for both DOCSIS and IEEE 802.16. They have chosen GPC mode for bandwidth grants. It is a centralized approach wherein all the scheduling decisions are taken at the base station (BS). Authors have claimed the architecture to be simple and capable of supporting diverse QoS requirements for various service flows. No simulation results are presented to show

efficiency and performance of the proposed architecture. In [6], the authors proposed a hierarchical structure for bandwidth allocation to decide whether QoS for a particular connection can be satisfied at the BS. They use a simple admission control mechanism as described in equation(1). Bandwidth allocation is the only QoS criterion used in this architecture. However such scheme may fail to satisfy the QoS requirements for service classes which not only require bandwidth guarantee but also need guarantee in terms of delay and jitter. IEEE 802.16 standard [1] neither specifies any Admission Control nor Scheduling Architecture, and leaves them to the implementors.

III. OVERVIEW OF IEEE 802.16 BROADBAND ACCESS SYSTEMS

A. Basic Operation

IEEE 802.16 standard provides specification of the MAC and physical layer. 802.16 physical layer operates at 10-66 GHz and 2-11 GHz with data rates between 32 and 130 Mbps, depending on the channel frequency and modulation scheme. Figure 1 shows a typical 802.16 network. It consists of a central radio Base Station (BS) and a number of Subscriber Stations(SS) which communicate with the BS. The BS is typically connected to wireline network to extend the connectivity of SSs to Internet. SSs are user premise network equipments usually installed at Small Office Home Office (SOHO) or residential customer sites.

The network access to the buildings is achieved through exterior antennas communicating with the BS. Each SS can have a number of users accessing the network. Both BS and SS are fixed, but inside the SS, users can be either static or mobile. Communication can happen in two modes: Point-to-Multipoint (PMP) mode and Mesh mode. In PMP mode, all communications happen through the BS and the BS acts as the central entity that decides the transmission and reception schedule of the SSs. In Mesh mode SSs can communicate with each other without the need of BS. Our study is based on the PMP mode of operation. In PMP mode, a BS controls all the communication in its coverage area. The communication path between SS and BS has two directions: Uplink (from SS to BS) and Downlink (from BS to SS), multiplexed either with Time Division Duplex(TDD) or Frequency Division Duplex (FDD) [7]. Transmission parameters, including the modulation parameters and coding schemes, may be adjusted individually for each SS on a frame-by-frame basis. SS uses *Bandwidth Request* mechanisms to specify Uplink bandwidth requirement to the BS. There are two modes for granting bandwidth requested by SS.

- *Grant Per Connection* (GPC): Each connection is treated separately and bandwidth is allocated to each connection explicitly. SS then transmits in the order specified by the BS.
- *Grant Per Subscriber Station*(GPSS): All connections from a single SS are treated as single unit and bandwidth is granted accordingly by the BS on a per SS basis. An additional Scheduler in the SS determines the service order among its connections in the granted slots.

The architecture and Admission Control mechanism described in this paper assumes TDD mode. In 802.16, a TDD frame has a fixed duration which may take one of the three values: 0.5, 1 or 2 msec. Each frame is divided into a Downlink subframe, and an Uplink subframe. The bandwidth allocated to each of the above subframes can be adaptive. Each subframe consists of an integer number of Physical Slots (PSs), which represents the minimum unit of bandwidth allocation. Each frame starts with a Preamble, used for synchronization. This is followed by the frame control section, containing the Down Link Map (DL-MAP) and UpLink Map (UL-MAP) fields, which describe the usage of PSs in downlink and uplink directions respectively. Each SS receives and decodes the control information of the downlink direction contained in the DL-MAP and looks for MAC headers indicating data for itself in the remainder of the Downlink subframe.

Through the UL-MAP, the BS informs the transmission opportunities of SSs, based on the bandwidth requests made by each SS. Bandwidth requests are transmitted through special purpose information elements called *BW-Requests*. Each SS having decoded the corresponding control information contained in the UL-MAP, knows exactly which PSs of the Uplink subframe it is allowed to transmit in.

B. Uplink Scheduling Services

Scheduling services represent the data handling mechanisms supported by the MAC Scheduler on a connection. Each connection is associated with a single data service [1]. Each data service is associated with a set of QoS parameters that quantify aspects of its behavior. 802.16 standard supports four types of services: *Unsolicited Grant Service* (UGS), *Real-time Polling Service* (RTPS), *Non-real-time Polling Service* (NRTPS), and *Best Effort* (BE).

- UGS: The UGS is designed to support real-time service flows that generate *fixed-size* data packets on a *periodic* basis, such Voice over IP. Thus, it eliminates the overhead and latency of SS requests and assures that grants are available to meet the real-time needs of the connections.
- RTPS: The RTPS is designed to support real-time service flows that generate variable sized data packets on a periodic basis, such as Moving Pictures Experts Group (MPEG) video. The service offers real-time, periodic, unicast request opportunities, which meet the connection's real-time needs and allow the SS to specify the size of the desired grant. This service requires more request overhead than UGS, but supports variable grant sizes for data transport efficiency. The BS should provide periodic unicast request opportunities so that the SSs can change their bandwidth requests.
- NRTPS: The NRTPS is designed to support non-real-time service flows that require variable sized data grants on a regular basis, such as high bandwidth FTP. The NRTPS offers unicast polls on a regular basis, but at a larger intervals than RTPS, assuring that the service flow receives request opportunities even during network congestion. The BS shall provide timely unicast request opportunities. NRTPS works

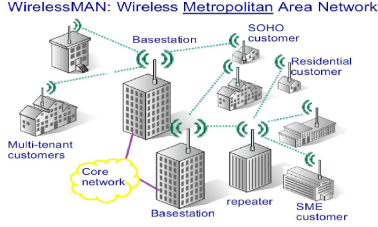


Fig. 1. Typical 802.16 Network

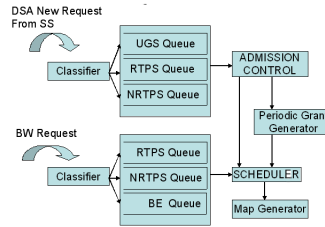


Fig. 2. Base Station Architecture

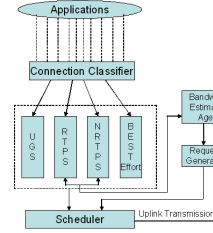


Fig. 3. Subscriber Station Architecture

similar to RTP Service but has larger values for its parameters.

- BE service: The intent of the BE service is to provide efficient service for Best Effort traffic like Web traffic. BS provides contention slots (in the UL-MAP) for SS to make new BE connection requests.

The traffic scheduler located at the BS decides on the allocation of PSs in each time frame by considering the following parameters for each of the active connections:

- the scheduling service specified for the connection
- QoS parameter values of the connections
- the availability of data for transmission (queue size)
- the capacity of the available bandwidth

By specifying a scheduling service and its associated QoS parameters, the BS Scheduler can anticipate the throughput and latency needs of the Uplink traffic and provide polls and/or grants at appropriate times.

IV. OUR QoS ARCHITECTURE

A. Base Station Architecture

Figure 2 depicts our proposed QoS architecture at the base station, that uses GPC mode for granting bandwidth to SSs. Our main goals in designing the architecture are to provide delay and bandwidth guarantees for various applications while still achieving high system Utilization. The architecture supports all types of services specified in IEEE 802.16 standard. Since IEEE 802.16 MAC protocol is *connection oriented*, any application must establish a connection with the BS as well as associate it to one of the service class types before it can start transmitting data. When a new flow generates/updates its parameters through Dynamic Service Addition, Change or Delete (DSA/DSC/DSD) requests, it sends a message to the BS. The classifier at the BS depending on the type of service request, classifies it into one of the Priority Queues (Priority Order: UGS Queue>RTPS Queue >NRTPS Queue). Best-Effort requests do not go through Admission Control process. If bandwidth is available at the end of each scheduling interval, the scheduler will allocate it to BE traffic.

The Priority Queues (UGS, RTPS, NRTPS) are accessed by the Admission Control module in order to check whether the requested QoS can be guaranteed in the current situation at the BS. If accepted, each connection will be allotted a unique connection identifier (cid) and the Admission control informs the scheduler to allocate bandwidth request slots in the next scheduling interval to that connection. Since

the Downlink and Uplink are broadcast channels, the only way that a SS can know that it has been admitted or not, is by looking at the Uplink Map message which contains the information of slot allocation to various SSs. If the connection is accepted by Admission control, the SS will then send its bandwidth request which will be classified and directed to the appropriate Priority Queue on the basis of cid. The periodic grant generator ensures allocation of requested data slots for all previously admitted UGS flows in each of the scheduling interval such that QoS guarantees are not violated. The Scheduler looks at the Priority Queues for bandwidth requests of different services and decides on the slot allocation, which is then fed to the Map Generator. The Map Generator generates an Uplink Map message. This generated Uplink Map message is used to generate the TDD Frame.

B. Subscriber Station Architecture

Figure 3 depicts the QoS architecture of SS for making bandwidth requests periodically, depending on the service class type. Applications once admitted into the network, are classified into various service class types by the connection classifier at the MAC layer. This results in application data being directed to one of the priority Queues (UGS Queue>RTPS Queue>NRTPS Queue>BE Queue) at SS. Within the Queue, we follow First-come-First-Serve policy. The MAP messages will inform the SSs when to transmit data and when to transmit bandwidth request. The SSs will inform the BS about their bandwidth requirements by making specific bandwidth requests for each connection. We assume that the application informs the SS about its traffic characteristics such as *minrate* and *maxrate*. While making DSA, the SS informs the BS about the traffic characteristics that it is going to request in the future. A Bandwidth Estimator Agent (BEA) monitors the queue length of each RTPS and NRTPS connections to estimate the bandwidth requirement of the connection and subsequently make the appropriate bandwidth request for such connections. Details of working of BEA is presented in Section VIII.

V. NOTATIONS AND DEFINITIONS

The number of connections in a service class admitted into the network is denoted by $N_{service_class}$. The total number of connections (of all classes) in the system is denoted as N . We denote i^{th} ($1 \leq i \leq N_{service_class}$) connection request (received at the BS) of

a *service_class* as $C_i^{service_class}$. Service class can be one of the four services defined in 802.16 i.e. $service_class \in \{UGS, RTPS, NRTPS, BE\}$. The service parameters of a connection are denoted in brackets as given below.

- A UGS connection request comes with the following parameters: *nominal grant interval, tolerated grant jitter, maximum_rate*. The i^{th} UGS connection is represented as C_i^{UGS} and its associated parameters are denoted as $C_i^{UGS}[ngi]$, $C_i^{UGS}[tgj]$, $C_i^{UGS}[maxrate]$ respectively.
- An RTPS connection request comes with the following parameters: *nominal polling interval, tolerated poll jitter, minimum_rate, maximum_rate*. So the i^{th} RTPS connection is denoted as C_i^{RTPS} whose parameters are represented as $C_i^{RTPS}[npi]$, $C_i^{RTPS}[tpj]$, $C_i^{RTPS}[minrate]$, $C_i^{RTPS}[maxrate]$ respectively.
- An NRTPS connection request comes with the following parameters: *nominal polling interval, tolerated poll jitter, minimum_rate, maximum_rate, traffic_priority* which are represented as $C_i^{NRTPS}[npi]$, $C_i^{NRTPS}[tpj]$, $C_i^{NRTPS}[minrate]$, $C_i^{NRTPS}[maxrate]$, $C_i^{NRTPS}[priority]$ for the i^{th} connection C_i^{NRTPS} respectively.
- Finally, a best effort connection request comes with the following parameters: *maximum_rate, traffic_priority* which are represented as $C_i^{BE}[maxrate]$, $C_i^{BE}[priority]$ for the i^{th} connection C_i^{BE} .
- HyperInterval: Since connections come with different parameters, HyperInterval is used for testing admissibility of connections. This makes sure that QoS requirements are met at every periodic interval of the corresponding service type. HyperIntervals of different service types are defined as follows.

$$HI^{UGS}(N) = \forall i \text{ LCM}(C_i^{UGS}[ngi]), 1 \leq i \leq N^{UGS}$$

where LCM is the Least Common Multiple. For RTPS connections, the HyperInterval is defined as:

$$HI^{RTPS}(N) = \forall i \text{ LCM}(C_i^{RTPS}[npi]), 1 \leq i \leq N^{RTPS}$$

Similarly, for NRTPS connections, the HyperInterval is defined as:

$$HI^{NRTPS}(N) = \forall i \text{ LCM}(C_i^{NRTPS}[npi]), 1 \leq i \leq N^{NRTPS}$$

Finally, the HyperInterval of all the connections across the three service categories are calculated as follows:

$$HI(N) = \text{LCM}(HI^{UGS}, HI^{RTPS}, HI^{NRTPS})$$

Note that the HyperInterval changes as the number of connections in the system changes. $HI(N)$ is the parameter used to check admissibility of a connection by the QoS-CAC module.

VI. QoS CALL ADMISSION CONTROL (QoS-CAC) ALGORITHM

In this section we first explain the conventional bandwidth based CAC (BW-CAC) and then give the details of our QoS-CAC algorithm.

A. Bandwidth based CAC (BW-CAC)

Most of the literature we surveyed for 802.16 system, we found that they have implicitly assumed a conventional bandwidth based CAC (BW-CAC). BW-CAC admits flows as long as there is enough bandwidth to satisfy the incoming request, but it does not consider the deadline constraints of the connections. The BW-CAC receives all the DSA/DSC/DSD requests and updates the available bandwidth after admitting new connection or deleting an outgoing connection or honoring bandwidth change request of a connections. The available bandwidth (BW_{avail}) is given by

$$BW_{avail} = BW - \sum_{s \in \{UGS, RTPS, NRTPS\}} \sum_{i=1}^{N^s} C_i^s[rate] \quad (1)$$

where $C_i^s[rate] = C_i^s[maxrate]$ when $s \in UGS$, $C_i^s[rate] = C_i^s[minrate]$ otherwise and BW is the total link bandwidth. Note that available bandwidth is calculated based on minimum rate, although a connection may have been allocated more than *minrate*. This is because for variable rate connections (e.g., RTPS), only the minimum rate is guaranteed.

B. Overview Of QoS-CAC

Our QoS-CAC algorithm (running at the BS) admits connections such that QoS guarantee is provided to all the admitted connections. A new connection request is classified into a particular queue depending on the associated Service Class type. QoS-CAC services the *UGS* connection queue first, followed by *RTPS* and then by *NRTPS* queues. Thus, it provides highest priority to UGS connections requests followed by RTPS and NRTPS connection requests. There is no need for Admission Control to Best-Effort connections since it does not require any guarantees. In every scheduling interval, the QoS-CAC algorithm scans the new request queues of different service classes in the order mentioned and decides whether it can guarantee the requested QoS of the new connection as well as the existing connections, if the connection is admitted.

1) *Admission Decision of UGS Connection*: If the request is of type UGS then, it should satisfy the necessary condition: *the requested slots based on its maxrate within its ngi should be less than or equal to the total number of slots that can actually be accommodated within the tgj based on BW*. i.e

$$\begin{aligned} & \lceil (C_i^{UGS}[ngi] * C_i^{UGS}[maxrate]) / slot_size \rceil \\ & \leq \lfloor (C_i^{UGS}[tgj] * BW) / slot_size \rfloor \end{aligned} \quad (2)$$

Once the necessary condition (2) is satisfied then the next step is to ensure that the required number of data slots are available within *tolerated grant jitter* (tgj) for each nominal grant interval (ngi) in a period equal to the HyperInterval $HI[N]$. Our CAC uses a helper routine *search(no_of_slots, initial_slot, final_slot)* (used in Algorithm 1 and Algorithm 2) to complete this task. This routine searches for *no_of_slots* in an interval between $[initial_slot, final_slot]$. Figure 4 shows allocation

of data slots to a connection. The connection requires 2 data slots (see Figure 4(a)) in every nominal grant interval (ngi). These two data slots are allocated starting from tgj and moving to the left. This task is done by $allocate(no_of_slots, initial_slot, final_slot, cid)$. This routine allocates no_of_slots starting from $final_slot$ from right to left to connection with identifier cid .

Figure 4 represents the case where a new request arrives with same ngi and tgj which requires one data slot. The previously allocated two data slots (see Figure 4(a)) is shifted towards the left by one slot to make room for the new connection(see Figure 4(b)). Our algorithm always allocates slots to the new request such that the new request gets its slots starting from its deadline (tgj) to the left.

In the previous example, the connections' slots were allocated contiguously. But this may not be the case always. If a new request has the same ngi as the previously admitted request but with a different tgj (or with different ngi and tgj) then the allocation of slots may become non-contiguous as shown in Figure 5. Since tgj of the second connection (see Figure 5(b)) falls on one of the allotted slots of the first connection (Figure 5(a)), one slot of the first connection is shifted to the left to make room for the slot required by the 2nd connection.

2) *Admission Decision of RTPS Connection:* If the request is of type RTPS then it should satisfy the necessary condition: *the number of required slots within the npi as per its $minrate$ should be less than or equal to the total number of slots that can actually be accommodated within the npi as per the total bandwidth.* i.e.,

$$\left[(C_i^{RTPS}[npi] * C_i^{RTPS}[minrate]) / slot_size \right] \leq \left[(C_i^{RTPS}[tpj] * BW) / slot_size \right] \quad (3)$$

Once the necessary condition (3) is satisfied, the number of slots needed for making a bandwidth request (called *Request Slot*) should be found within the *tolerated poll jitter* (tpj) in every nominal polling intervals (npi) within $HI[N]$. The required number of data slots should be found in each nominal polling interval of $HI[N]$ as per the $minrate$. This is depicted in Figure 6 and Figure 7. In Figure 6, allocation of the first connection is shown. There is one Request Slot assigned to the connection at tpj and two data slots assigned at npi . Now, when the second connection arrives, it displaces the allocated Request Slots as well as data slots of first connection to the left (Figure 7). This example assumes that the npi and tpj of the two connections are the same. If they are different, then it may lead to non-contiguous allocation very similar to UGS connections discussed earlier. This task of slot allocation is done by $allocate()$ routine in Algorithm 2.

3) *Admission Decision of NRTPS Connection:* Admission criteria of NRTPS are very similar to that of RTPS because they differ only in the parameter values once the NRTPS requests are sorted by *priority*. NRTPS parameters carry larger values of tpj and npi compared to RTPS.

4) *Admission Decision of Best Effort Connection:* If the request arrived is of type Best Effort(BE), the algorithm looks for available free slots and assigns it to the arrived

request. Since BE traffic is not given any guarantees, there is no admission control for BE connections. If a higher priority request arrives, it is serviced first in the next scheduling interval and then only any existing BE connections are considered.

VII. PSEUDOCODE OF QoS-CAC ALGORITHM

In this section, we provide the pseudocode of the call admission control algorithm for UGS and RTPS connection. CAC for NRTPS is very similar to RTPS, hence is not provided here. A new connection request is sent by SS to the BS. The BS runs the appropriate QoS-CAC algorithm to check whether the connection can be admitted or not.

A. CAC for UGS Connection

$Admission_Control_for_UGS()$ handles the admission of UGS connections. It first checks for the necessary condition in Line 1. Then in Line 3, it finds the number of slots required to satisfy QoS requirement of the connection in every ngi period. It then makes sure that the required number of slots are available in every ngi period in the HyperInterval $HI(N)$ (Line 7). Once this is ensured, it then goes on to allocate slots. Allocation of slots is done as explained in Section VI-B.1.

Algorithm 1 Admission_Control_for_UGS ($C_i^{UGS}, HI[N]$)

Require: {/*This algorithm takes new connection request C_i^{UGS} as Input and allocates slots if accepted*/} {/*Check if necessary condition is satisfied*/}

```

1: if  $\left[ (C_i^{UGS}[ngi] * C_i^{UGS}[maxrate]) / slot\_size \right] \leq \left[ (C_i^{UGS}[tgj] * BW) / slot\_size \right]$  then
2:    $no\_of\_ngi = (HI[n] / C_i^{UGS}[ngi])$ ;
3:    $no\_of\_slots = \lceil (C_i^{UGS}[ngi] * C_i^{UGS}[maxrate] / slot\_size) \rceil$ ;
4:    $ngi\_in\_slot\_units = \lfloor (C_i^{UGS}[ngi] * BW) / slot\_size \rfloor$ ;
5:    $initial\_slot = 0$ ;  $slots\_found = 1$ ;
6:    $final\_slot = \lfloor (C_i^{UGS}[tgj] * BW) / slot\_size \rfloor$ ; {/*Check for availability of required number of data slots within tolerated grant jitter in every  $ngi$  within  $HI[N]$ */}
7:   for  $j=1$  to  $no\_of\_ngi$  do
8:      $slots\_found = search(no\_of\_slots, initial\_slot, final\_slot)$ ;
9:     if ! $slots\_found$  then
10:      connection rejected
11:      return;
12:     end if
13:      $initial\_slot = \lfloor (j * C_i^{UGS}[ngi] * BW) / slot\_size \rfloor$ ;
14:      $final\_slot = initial\_slot + ngi\_in\_slot\_units$ ;
15:   end for {/*Required slots are available, now allocate the slots*/}
16:    $initial\_slot = 0$ ;
17:    $final\_slot = \lfloor (C_i^{UGS}[tgj] * BW) / slot\_size \rfloor$ ;
18:   for  $j=1$  to  $no\_of\_ngi$  do
19:      $allocate(no\_of\_slots, initial\_slot, final\_slot, cid)$ ;
20:      $initial\_slot = \lfloor (j * C_i^{UGS}[ngi] * BW) / slot\_size \rfloor$ ;
21:      $final\_slot = initial\_slot + ngi\_in\_slot\_units$ ;
22:   end for
23: end if

```

B. CAC for RTPS Connection

$Admission_Control_for_RTPS()$ shows the algorithm for admitting an RTPS connection. This algorithm is quite similar to UGS except that it also needs to make sure that there are enough number of bandwidth request slots available in every tpj in a HyperInterval $HI(N)$. The admission decision is based on the $minrate$ of the connection. Thus, the connection is admitted if the system can meet the minimum rate of the connection. Thus, it is possible that if the connection requests for more bandwidth

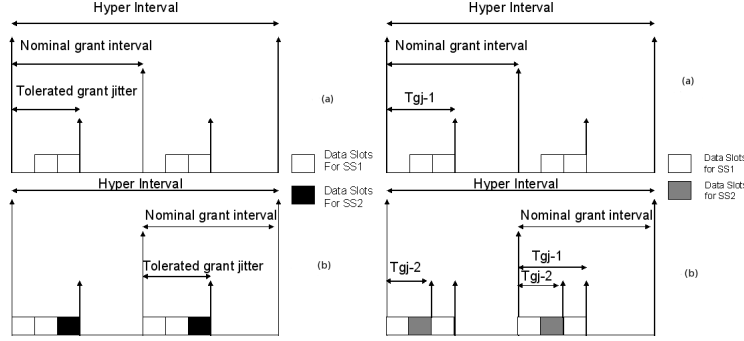


Fig. 4. Slot Allocation of UGS Requests

Fig. 5. Non-Contiguous Allocation of Data Slots of UGS Requests

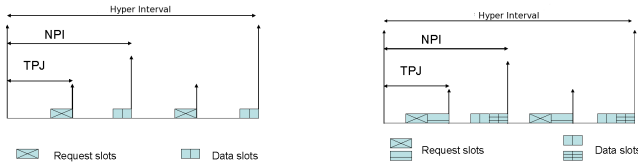


Fig. 6. Slot Allocation of RTPS Request

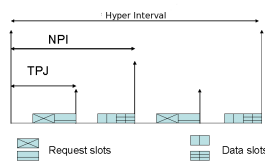


Fig. 7. Slot Allocation of New RTPS Request

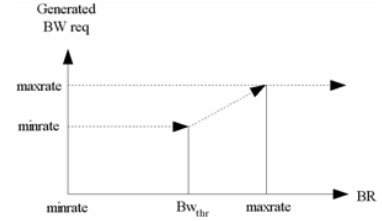


Fig. 8. Working of Bandwidth Estimator

than the $minrate$, the system may not be able to allocate the requested bandwidth, but may only provide $minrate$ (or whatever is available at that time).

VIII. BANDWIDTH ESTIMATOR FOR RTPS AND NRTPS CONNECTIONS

RTPS and NRTPS services have a variable bandwidth requirement (between $minrate$ and $maxrate$). As per the specification, the network needs to guarantee only minimum rate to such connections. If the scheduler allocates only $minrate$, then the system can admit more connections, but packets of admitted connection may encounter large delays, if the connection actually needs more bandwidth. The other option is to always allocate $maxrate$ to the connections. Although this will ensure that packets of the connection have small delays, this scheme will result in wastage of resources when the connection really needs less than the $maxrate$. To address this issue, we propose a simple way of estimating the bandwidth requirement of RTPS and NRTPS connection at the SS. There is a *Bandwidth Estimator Agent* (BEA) at the MAC layer which monitors the queue lengths of each RTPS and NRTPS flows at regular interval and calculates the bandwidth requirement of the flow by measuring the arrival rate of the traffic over the interval. The bandwidth requirement (BR) is calculated as the ratio of change of queue length between current and previous monitoring interval to the monitoring interval. A configurable threshold, called BW_{thr} , is used while requesting for bandwidth. The bandwidth request is made by BEA as per the following rules (also shown in Figure 8).

- $minrate \leq BR \leq BW_{thr}$: BEA sends a bandwidth request for $minrate$
- $BW_{thr} < BR \leq maxrate$: BEA sends a bandwidth request for BR
- $maxrate < BR$: BEA sends a bandwidth request for $maxrate$

IX. SIMULATION EXPERIMENTS

We have developed a simulator using C on a Linux platform to evaluate the performance of our QoS-CAC algorithm. In this section we present the simulation set up and results of our experiments

A. Simulation Parameters

For our experiments, the system parameters used are as shown in Table I. We divided the entire channel capacity into 2 halves making 16Mbps available to Downlink and 16Mbps towards Uplink. We have used Voice over IP (VoIP) applications as the sources of UGS traffic. The UGS traffic parameters are shown in Table II, which depends on the type of codec used. Table III lists the parameters used for RTPS, NRTPS and Best Effort connections. Arrival of connections (regardless of service type) is modeled with a Poisson distribution. The lifetime of a connection is exponentially distributed with an average lifetime of 180 seconds. We used three codecs for our simulation environment for generating VoIP traffic. A newly generated UGS connection request is assigned a codec randomly out of the three.

B. Experimental Results

1) *Only UGS Connections*: Our first experiment had a dedicated 16Mbps Upstream channel capacity for only

Algorithm 2 Admission_Control_for_RTPS (C_i^{RTPS} , $HI[N]$)

Require: {/*This algorithm takes service request C_i^{RTPS} as Input and allocates slots in the Map if accepted*/} {/*Check if necessary condition is satisfied*/}

```

1: if  $[(C_i^{RTPS}[npi] * C_i^{RTPS}[minrate])/slot\_size] \leq$ 
 $[(C_i^{RTPS}[npi] * BW)/slot\_size]$  then
2:    $no\_of\_npi = (HI[N]/C_i^{RTPS}[npi]);$ 
3:    $npi\_in\_slot\_units = \lfloor C_i^{RTPS}[npi] * BW / slot\_size \rfloor;$ 
4:    $initial\_slot\_bw=0;$ 
5:    $final\_slot\_bw = \lfloor (C_i^{RTPS}[tpj] * BW) / slot\_size \rfloor;$ 
6:   for  $j=1$  to  $no\_of\_npi$  do
7:      $slots\_found = search(bw\_request\_slots, initial\_slot\_bw, final\_slot\_bw);$ 
      {/* $bw\_request\_slots$  is the number of slots required to make a
      bandwidth request*/}
8:     if ! $slots\_found$  then
9:       connection rejected
10:      return;
11:     end if
12:      $initial\_slot\_bw = \lfloor j * (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor$ 
13:      $final\_slot\_bw = initial\_slot\_bw + npi\_in\_slot\_units;$ 
14:   end for {/*Required number of bandwidth request slots are available, now
check if required number of data slots are available*/}
15:    $no\_of\_slots = \lfloor (C_i^{RTPS}[npi] * C_i^{RTPS}[minrate]) / slot\_size \rfloor;$ 
16:    $initial\_slot=0;$ 
17:    $final\_slot = \lfloor (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor;$ 
18:   for  $j=1$  to  $no\_of\_npi$  do
19:      $slots\_found = search(no\_of\_slots, initial\_slot, final\_slot);$ 
20:     if ! $slots\_found$  then
21:       reject the connection
22:       return
23:     end if
24:      $initial\_slot = \lfloor j * (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor$ 
25:      $final\_slot = initial\_slot + npi\_in\_slot\_units;$ 
26:   end for {/*Required number of data slots are present in every  $npi$  in one
 $HI[N]$ , now allocate the bandwidth and data slots*/}
27:    $initial\_slot\_bw=0;$ 
28:    $final\_slot\_bw = \lfloor (C_i^{RTPS}[tpj] * BW) / slot\_size \rfloor;$ 
29:    $initial\_slot=0;$ 
30:    $final\_slot = \lfloor (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor;$ 
31:   for  $j=1$  to  $no\_of\_npi$  do
32:      $allocate(bw\_request\_slots, initial\_slot\_bw, final\_slot\_bw, cid);$ 
33:      $allocate(no\_of\_slots, initial\_slot, final\_slot, cid);$ 
34:      $initial\_slot\_bw = \lfloor j * (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor;$ 
35:      $initial\_slot = \lfloor j * (C_i^{RTPS}[npi] * BW) / slot\_size \rfloor;$ 
36:      $final\_slot\_bw = initial\_slot\_bw + npi\_in\_slot\_units;$ 
37:      $final\_slot = initial\_slot + npi\_in\_slot\_units;$ 
38:   end for
39: end if

```

Parameter	Value
Channel Capacity	32Mbps(QPSK)
Symbol Rate(MBd)	16
Frame Duration	1ms
Physical slots per Frame	4000
Slot size	1 byte

TABLE I
SYSTEM PARAMETERS USED IN SIMULATION

UGS traffic. For this simulation, the codec chosen is always G.711. Figure 9 shows the change in flow acceptance ratio as the average connection arrival rate changes. The flow acceptance ratio is almost 100% until the connection arrival rate is around 38. Thus, if a 802.16 network is to be deployed for a voice only (UGS) traffic, then the network administrator should make sure that new connections arrive at a rate less than 38 connection per second.

2) *Only RTPS Connections:* In this experiment, we used a dedicated 16Mbps Upstream channel capacity for only RTPS traffic. The same parameters are used for every RTPS connection which are as follows. $maxrate=$

CODEC	bit rate	npi(ms)	tpj(ms)	active-duration(s)
G.711	64kbps	20	10	180
G.721	32kbps	20	10	180
G.728	16kbps	20	10	180

TABLE II
UGS TRAFFIC PARAMETERS

256kbps, $minrate=128kbps$, $npi=1s$ and $tpj= 0.5s$. The Flow-Acceptance ratio vs arrival rate is shown in Figure 10. The Acceptance Ratio starts dropping much more quickly than the UGS-only setup as arrival rate increases. This is because of the fact that RTPS flows have requesting rates much higher than that of UGS traffic. Tight delay requirements along with higher requesting rates of RTPS connections adversely affect admission of a new connection.

3) *All Classes of Traffic:* In the next experiment we allowed all types of traffic. It used the parameters listed in Table III to generate RTPS, NRTPS and BE traffic and Table II to generate UGS traffic. Arrival rate (λ) (of each class) are increased from 1 to 20 arrivals/sec. As mentioned before, lifetime of each connection is exponentially distributed with a mean of 180sec. Since BE traffic does not go through CAC, Flow-Acceptance of BE traffic is actually the ratio of number of slots assigned to BE class to the total number of slots required to satisfy all the BE traffic. Figure 11 shows the Flow-Acceptance ratio of each class of traffic as the arrival rate increases, when each RTPS and NRTPS connection is allocated its $maxrate$. Similarly, Figure 12 shows the Flow-Acceptance ratio of each class when each RTPS and NRTPS connection is allocated its $minrate$. Allocating $minrate$ to RTPS and NRTPS connections leaves more slots for other connections compared to the case when $maxrate$ is allocated. Hence, the Flow-Acceptance ratio improves across all the classes when $minrate$ based allocation is done to RTPS and NRTPS connections. However, RTPS and NRTPS traffic do not require constant bandwidth, but need varying bandwidth between its minimum and maximum rate. We used the bandwidth estimator to change the bandwidth requirement dynamically as described in Section VIII. For this experiment, we have set BW_{thr} midway between $minrate$ and $maxrate$ of a connection. Figure 13 shows the Flow-Acceptance ratio of each class of traffic when the bandwidth estimator is used. When this is compared with the case when allocation is done based on $maxrate$ (Figure 11) it can be noticed that Flow-Acceptance ratio improved for all classes of traffic except for NRTPS. NRTPS is similar to RTPS connection except that the parameters have larger values (e.g., the $minrate$ and $maxrate$ are larger than RTPS connection). Thus, slots (bandwidth) which were saved because of bandwidth estimation could not be used to admit more NRTPS connections because of their large bandwidth requirement. Other types of connections, because of their small parameter values, were able to take up the saved slots and improve their acceptance ratio.

Since BW-CAC admits connections based solely on availability of bandwidth (slots), it will typically have higher utilization, but will incur deadline misses. Our QoS-CAC, on the other hand, will have lower utilization, but will

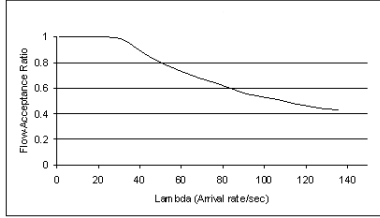


Fig. 9. Flow Acceptance Ratio vs. Connection Arrival Rate for UGS only Traffic

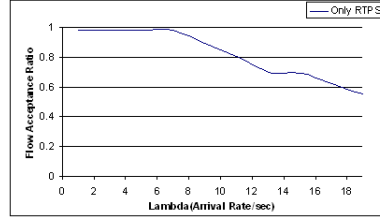


Fig. 10. Flow Acceptance Ratio vs. Connection Arrival Rate for RTPS only Traffic

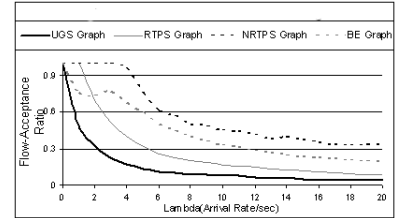


Fig. 11. All Classes of Traffic, RTPS and NRTPS Connections Allocated *maxrate*

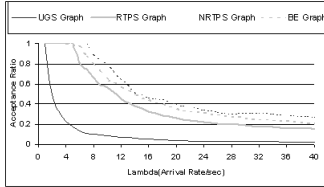


Fig. 12. All Classes of Traffic, RTPS and NRTPS Connections Allocated *minrate*

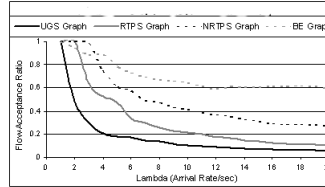


Fig. 13. All Classes of Traffic, RTPS and NRTPS Connections Allocated with Bandwidth Estimator

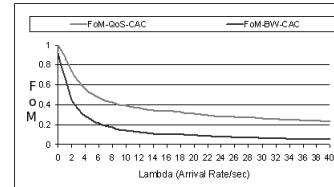


Fig. 14. FoM Comparison

Service Type	Max-Rate	Min-Rate	n_{pi}	t_{pj}
RTPS	128kbps	64kbps	0.5s	0.5s
RTPS	256kbps	128kbps	0.5s	0.5s
RTPS	512kbps	256kbps	0.5s	0.5s
NRTPS	128kbps	64kbps	1s	1s
NRTPS	256kbps	128kbps	1s	1s
NRTPS	512kbps	256kbps	1s	1s
BE	32kbps	-	-	-
BE	64kbps	-	-	-
BE	128kbps	-	-	-

TABLE III
RTPS, NRTPS, BE PARAMETERS

have no deadline misses (because it admits connections only when deadline of the connection can be met). Thus, BW-CAC will typically have higher flow acceptance ratio than QoS-CAC. But since there is a tradeoff between utilization achieved and deadlines missed, comparing the flow acceptance ratio of the two is not fair. Hence we define a composite performance index called *Figure of Merit (FoM)* as defined below

$$FoM = \frac{Utilization * (No.of_conn_admitted - No.of_conn_miss_deadlines)}{(Total_No.of_conn_req)}$$

Utilization is the ratio of number of slots assigned to the connection to the total number of slots available. Note that for QoS-CAC, the number of connections missing deadline is zero. Hence the *FoM* for QoS-CAC is essentially utilization multiplied by Flow-Acceptance ratio. But for BW-CAC, there will be connections missing deadline. This is factored into *FoM* as a penalty to the raw acceptance ratio by subtracting the number of connections missing their deadlines from the admitted connections. Figure 14 is a plot between arrival rate and *FoM* for BW-CAC and QoS-CAC. It is clear from the plot that QoS-CAC has a much better

FoM than BW-CAC. Hence, QoS-CAC is better suited for real time communication than BW-CAC.

X. CONCLUSION

We have presented a QoS architecture at BS and SS for an IEEE 802.16 network. The paper then outlines the details of resource (slot) allocation scheme and presents the pseudocode of our QoS-CAC algorithm. We also proposed a simple but effective method of estimating bandwidth of RTPS and NRTPS connections which enhances the performance of the system in terms of Flow-Acceptance ratio. We presented performance of our CAC in different scenarios. We introduced a composite performance index called *FoM* to compare QoS-CAC with BW-CAC and showed that our QoS-CAC performs better than conventional BW-CAC in terms of *FoM*.

REFERENCES

- [1] "IEEE Standard for Local and Metropolitan Area Networks." IEEE 802.16 Standard, 2002.
- [2] K. Wongthavarawat and A. Ganz, "Packet scheduling for QoS support in IEEE 802.16 broadband wireless access systems," Intel Journal on Communication Systems, 2003.
- [3] G. Nair, "IEEE 802.16 Medium Access Control and Service Provisioning," Intel Technology Journal, August 2004.
- [4] G. Chu, D. Wang, and S. Mei, "A QoS Architecture for the MAC protocol of IEEE 802.16 BWA System," IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions, June 2002.
- [5] M. Hawa and D. Petr, "Quality of Service Scheduling in Cable and Broadband Wireless Access Systems," Tenth International Workshop on Quality of Service, May 2002.
- [6] Jianfeng Chen, W. Jiao, and H. Wang, "A Service Flow Management Strategy for IEEE 802.16 Broadband Wireless Access Systems in TDD Mode," IEEE, 2005.
- [7] C. Eklund, R. B. Marks, and K. L. Stanwood, "IEEE Standard 802.16: A Technical Overview of the WirelessMAN Air Interface for Broadband Wireless Access," IEEE Communications Magazine, 2002.