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, sahoo}@it.iitb.ac.in

Abstract— **Scheduling and Call Admission Control (CAC) in IEEE 802.16 system play a vital role in the performance of the system. The 802.16 standard does not specify any scheduling architecture or CAC. Many proposals assume a bandwidth based CAC which only provides bandwidth guarantee, but cannot fulfill delay and jitter requirements. In this paper, we propose a CAC that ensures QoS guarantee in terms of bandwidth, delay and jitter. We also present a novel method of estimating banwdwidth requirement of variable bit rate application to increase the resource utilization of the system. We present our simulation result to show the effectiveness of bandwidth estimation and better performance over bandwidth based 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. Though some researchers have suggested implicit conventional bandwidth based CAC (BW-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.

[1]This research was partially supported by IRCC, IIT Bombay under grant number 03IR059 and by Ministry of Communications and Information Technology, Government of India, under grant number 1(1)/2004-E-Infra.

II. Related Work

There have been few proposals presented in the literature to support QoS in IEEE 802.16 networks. A joint bandwidth allocation and CAC for real-time and non-real-time polling services is presented in [4]. Packet level performances (e.g., delay) is used as a cost function which is used in an opimization formulation to allocate bandwidth and accept a new call. Chu et al. proposed a QoS scheduling architecture for the MAC protocol in 802.16 networks [5]. It is based on priority scheduling and dynamic bandwidth allocation. They have chosen Grant Per Subscriber Station (GPSS) 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 [6], the authors have proposed an uplink scheduling architecture to support QoS guarantees in terms of delay and bandwidth for both DOCSIS and IEEE 802.16. They have chosen Grant Per Connection (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 [7], 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. Authors in [8] report a scheduling and CAC scheme for real time video applications in fixed 802.16 network. A token bucket based uplink packet scheduling and CAC scheme for 802.16 network in presented in [9]. IEEE 802.16 standard [1] neither specifies any Admission Control nor Scheduling Architecture, and leaves them to the implementors.

III. Our QoS Architecture

IEEE 802.16 standard provides the MAC and physical layer specification. It operates at 10-66 GHz and 2-11 GHz with data rate between 32 and 130 Mbps. It consists of a 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 the SSs to the Internet. 802.16 network can operate in Point-to-Multipoint (PMP) or Mesh mode [1]. 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) [10]. SS uses *Bandwidth* Request mechanisms to specify Uplink bandwidth requirement to the BS. There are two modes for granting bandwidth requested by SS: GPC and GPSS [1]. The architecture and Admission Control mechanism described in this paper assumes PMP, GPC and 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 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). For more details of different modes and services readers are requested to refer to [1].

A. Base Station Architecture

Figure 1 depicts our proposed QoS architecture at the base station, that uses GPC mode for granting bandwidth to SSs. Our main goal in designing the architecture is 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: $UGSQueue > RTPSQueue > NRTPSQueue$). 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 an unique connection identifier (cid) and the Admission control informs the scheduler to allocate bandwidth request slots in the next scheduling interval to that connection (for RTPS and NRTPS connections). SSs look at Uplink Map (UL-MAP) message that 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 (for Non-UGS connections) 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 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 UL-MAP message.

B. Subscriber Station Architecture

Figure 2 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 ($UGSQueue > RTPSQueue > NRTPSQueue > BEQueue$) 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 VI.

IV. Notations and Definitions

The number of connections in a service class admitted into the network is denoted by $N_{service\_class}$. Service class can be one of the four services defined in 802.16 i.e. *service_class* $\in \{UGS, RTPS, NRTPS, BE\}$. For example, N_{UGS} denotes the number of UGS connections in the system. 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 $C_i^{service\_class}$. 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.

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

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

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

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

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

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

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

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. $HI(N)$ takes periodic slot requirements of all the existing connections into account. Thus, it makes sure that required number of slots are available in their respective periodic interval (e.g., npi for RTPS connections).

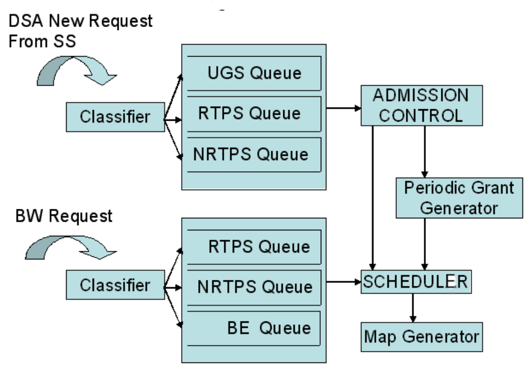

Fig. 1. Base Station Architecture

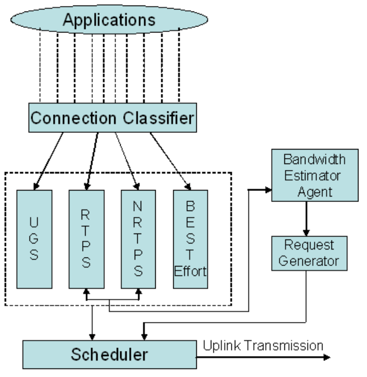

Fig. 2. Subscriber Station Architecture

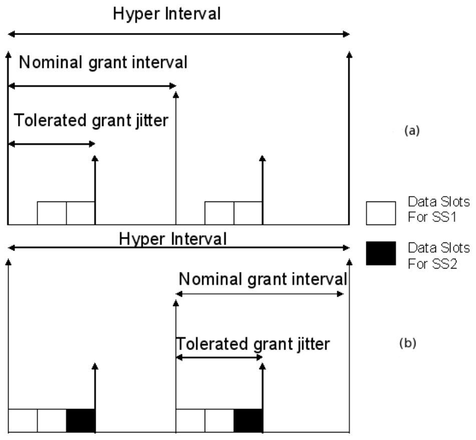

Fig. 3. Slot Allocation of UGS Requests

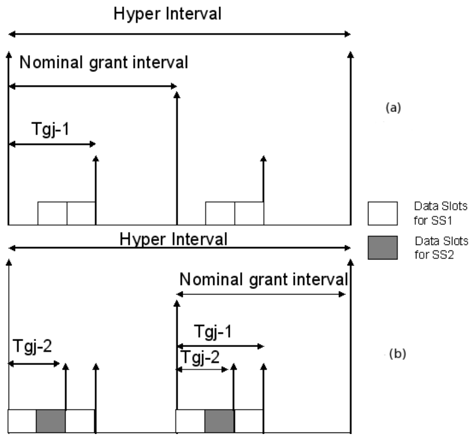

Fig. 4. Non-Contiguous Allocation of Data Slots of UGS Requests

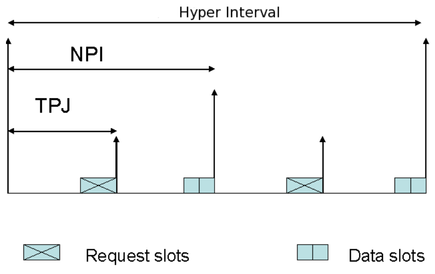

Fig. 5. Slot Allocation of RTPS Request

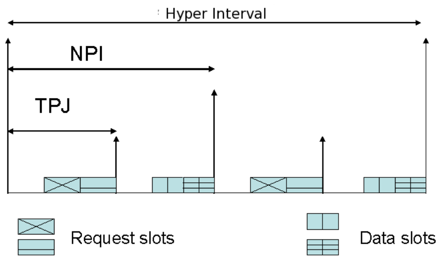

Fig. 6. Slot Allocation of New RTPS Request

V. 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)

Some 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 delay or jitter 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

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

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. The scheduler of the system is very tightly coupled with the QoS-CAC in the sense that the QoS-CAC provides the slot allocation (which the CAC would have done during admission decision) to the scheduler.

C. CAC for UGS Connection

$Admission\_Control\_for\_UGS()$, shown in Algorithm 1, handles the admission of UGS connections. It first checks for the necessary condition in Line 1. The necessary condition is that *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. This condition can be explained as follows. Based on its *maxrate* the connection needs those many number of slots in every *ngi* period. But those many slots should be available within *tgj*. But in an interval of length *tgj* the maximum number of slots avaiable is the BW times *tgj* of the connection (normalized to number of slots). If this number is less than what is required by the connection, then the request can be trivially rejected. 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). Algorithm 1 uses a helper routine $search(no\_of\_slots, initial\_slot, final\_slot)$ to complete this task. This routine searches for no_of_slots in an interval between $[initial\_slot, final\_slot]$. Once this is ensured, it then goes on to allocate slots. Figure 3 shows allocation of data slots to a connection. The connection requires 2 data slots (see Figure 3(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*. By allocating the slots from right to left, displacement of allocated slots of existing connections (to make room for new connection) can happen without any constraints.

Figure 3 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 3(a)) are shifted towards the left by one slot to make room for the new connection(see Figure 3(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 4. Since tgj of the second connection (see Figure 4(b)) falls on one of the alloted slots of the first connection (Figure 4(a)), one slot of the first connection is shifted to the left to make room for the slot required by the 2nd connection.

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 $\lceil(C_i^{UGS}[ngi] * C_i^{UGS}[maxrate])/slot\_size\rceil \leq \lfloor(C_i^{UGS}[tgj] * BW)/slot\_size\rfloor$ 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
```

D. CAC for RTPS Connection

$Admission\_Control\_for\_RTPS()$, shown in Algorithm 2, is used 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 $HI(N)$. In Line 1, it makes sure that it satisfies 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.*. Since the admission decision is based on the $minrate$ of the connection, the connection is admitted if the system can meet the minimum rate of the connection. Note that if the connection requests for more bandwidth 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). Once the necessary condition 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 $HI[N]$ (Line 6). When this is successful, Algorithm 2 finds required number of data slots in each nominal polling interval of $HI[N]$ as per the $minrate$ (Line 18). This is depicted in Figure 5 and Figure 6. In Figure 5, 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 6). 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 which has been explained in Section V-C.

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 $\lceil(C_i^{RTPS}[npi] * C_i^{RTPS}[minrate])/slot\_size\rceil \leq \lfloor(C_i^{RTPS}[npi] * BW)/slot\_size\rfloor$ 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:     $no\_of\_slots$=$\lceil(C_i^{RTPS}$[npi]*$C_i^{RTPS}$[minrate])/slot_size$\rceil$;
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
```

E. CAC for 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

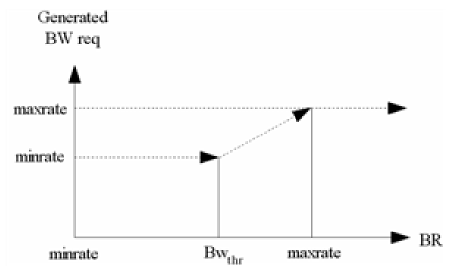

Fig. 7. Working of Bandwidth Estimator

carry larger values of tpj and npi compared to RTPS. Hence, CAC for NRTPS is very similar to RTPS, but is not provided here due to space limitation.

Request for Best Effort(BE) connection is always accepted. However, the scheduler will assign slots only after assigning slots of connections of other types.

VI. 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. We use exponential averaging of queue length to avoid reacting to *instantaneous changes*. 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 7).

- $minrate \leq BR \leq BW_{thr}$: 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 $maxrate$

VII. Simulation Experiments

A. Simulation Parameters

We have developed a simulator using C on a Linux platform to evaluate the performance of our QoS-CAC algorithm. 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 UGS traffic. For this simulation, the codec chosen is always G.711. Figure 8 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. $maxrate$= 256kbps, $minrate$=128kbps, npi=1s and tpj= 0.5s. The Flow-Acceptance ratio vs arrival rate is shown in Figure 9. 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 request 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 10 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 11 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 VI. For this experiment, we have set BW_{thr} midway between $minrate$ and $maxrate$ of a connection. Figure 12 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 10) 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 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.

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

CODEC	*bit rate* (kbps)	*ngi* (ms)	*tgj* (ms)	*active duration*
G.711	64	20	10	180
G.721	32	20	10	180
G.728	16	20	10	180

TABLE II
UGS TRAFFIC PARAMETERS

Service Type	*Max-Rate*	*Min-Rate*	*npi*	*tpj*
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

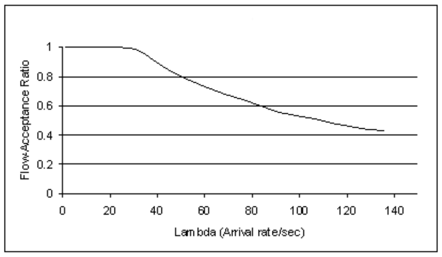

Fig. 8. Flow Acceptance Ratio vs. Connection Arrival Rate for UGS only Traffic

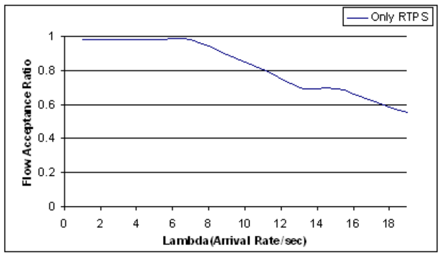

Fig. 9. Flow Acceptance Ratio vs. Connection Arrival Rate for RTPS only Traffic

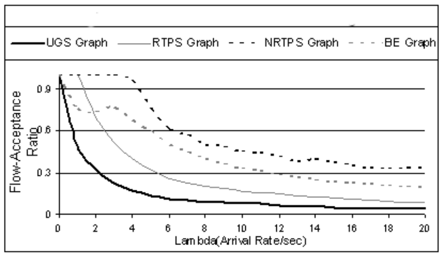

Fig. 10. All Classes of Traffic, RTPS and NRTPS Connections Allocated $maxrate$

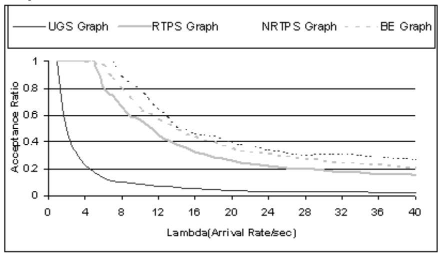

Fig. 11. All Classes of Traffic, RTPS and NRTPS Connections Allocated $minrate$

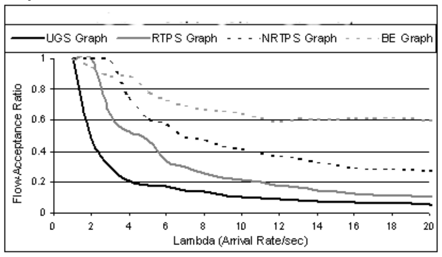

Fig. 12. All Classes of Traffic, RTPS and NRTPS Connections Allocated with Bandwidth Estimator

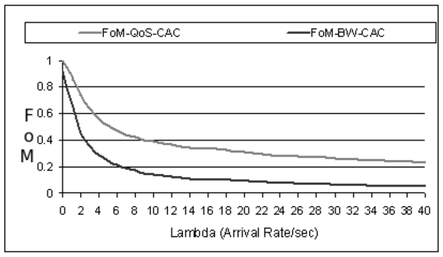

Fig. 13. FoM Comparison

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

$$FoM = \frac{U * (No\_of\_conn\_admitted - No\_of\_conn\_miss\_deadlines)}{Total\_No\_of\_conn\_req}$$

U is the utlization of the system and is defined as 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 13 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.

VIII. CONCLUSION

We have presented a QoS architecture at BS and SS for an IEEE 802.16 network. We have outlined the details of resource (slot) allocation scheme and presented 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 Figure of Merit (FoM) to compare QoS-CAC with BW-CAC and showed that our QoS-CAC performs better than conventional BW-CAC in terms of FoM. Thus, QoS-CAC is more suitable for 802.16 network which need to provide QoS guarantee to connections in terms of delay, jitter and bandwidth.

REFERENCES

[1] "IEEE Standard for Local and Metropolitian Area Networks." IEEE 802.16 Standard, 2004.
[2] K. Wongthavarawat and A. Ganz, "Packet scheduling for Qos support in IEEE 802.16 broadband wireless access systems," *International Journal on Communication Systems*, vol. 16, no. 1, pp. 81–96, 2003.
[3] G. Nair, "IEEE 802.16 Medium Access Control and Service Provisioning," *Intel Technology Journel*, vol. 8, no. 3, 2004.
[4] D. Niyato and E. Hossain, "Joint Bandwidth Allocation and Connection Admission Control for Polling Services in Ieee 802.16 Broadband Wireless Network," IEEE International Conference on Communications (ICC), MAY 2002.
[5] 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.
[6] M. Hawa and D. Petr, "Quality of Service Scheduling in Cable and Broadband Wireless Access Systems," IEEE International Workshop on Quality of Service, May 2002.
[7] J. Chen, W. Jiao, and H. Wang, "A Service Flow Management Strategy for IEEE 802.16 Broadband Wireless Access Systems in TDD Mode," IEEE International Conference on Communications (ICC), MAY 2005.
[8] O. Yang and J. Lu, "New Scheduling and CAC Scheme for Real-Time Video Application in Fixed Wireless Networks," pp. 303–307, IEEE Consumer Communications and Networking Conference, January 2006.
[9] T.-C. Tsai, C.-H. Jiang, and C.-Y. Wang, "CAC and Packet Scheduling using Token Bucket for IEEE 802.16 Networks," *Journal of Communications*, vol. 1, no. 2, pp. 30–37, May 2006.
[10] 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*, vol. 40, no. 6, pp. 98–107, 2002.