Implementation of Delay Assurance Service for Voice Applications in Wireless LANs

Pattipaka Subramanyam & Anirudha Sahoo
Kanwal Rekhi School of Information Technology
Indian Institute of Technology Bombay, Mumbai
Email: {subbu, sahoo}@it.iitb.ac.in

Parameswaran Ramanathan
Department of Electrical and Computer Engineering
University of Wisconsin, Madison, Wisconsin, USA
Email: parmesh@ece.wisc.edu

Abstract— **IEEE 802.11 has become de facto standard for wireless LANs. Although it was originally designed for data communication, emergence of Voice over IP (VoIP) has made it attractive for voice applications. But voice applications require delay guarantee. We have implemented Dynamic Class Selection (DCS) mechanism in Neighborhood Proportional Delay Differentiation (NPDD) (proposed in [1]) architecture to provide delay assurance in IEEE 802.11 networks. But we found that DCS is not suitable for voice applications which require tight delay bound. Hence, we propose two Adaptive Class Selection (ACS) mechanisms to provide delay assurances to voice flows. However, delay assurance at all nodes within a LAN cannot be provided with ACS or DCS. So we propose Measurement based Distributed Call Admission Control (MDAC) mechanism to provide delay assurances for voice traffic at all nodes. Our experimental results show that ACS and MDAC perform much better than DCS for voice flows at all loads.**

I. Introduction

The use of wireless devices like Personal Digital Assistants (PDAs), Smart Phones and Laptops based on IEEE 802.11 [2] have become commonplace. Although IEEE 802.11 WLAN was originally designed for providing wireless access for data traffic, the emergence of Voice over IP (VoIP) has made it attractive for wireless voice access through IP network. But VoIP applications require QoS from the network in terms of delay, jitter and packet loss. Point Coordination Function (PCF) in IEEE 802.11 provides real time support in infrastructure mode. But most of the existing IEEE 802.11 solutions do not support PCF. IEEE 802.11 DCF mode does not provide QoS for real time applications. Although IEEE 802.11e can support QoS, this MAC is yet to be standardized. Hence there is a need to provide QoS in IEEE 802.11 based WLANs to support VoIP applications.

Several MAC protocols have been proposed to provide QoS in wireless LANs. These are broadly classified into two categories: centralized and distributed. There are centralized scheduling algorithms [3], [4], [5] where a designated host (e.g. Access Point) coordinates the access to the wireless medium. In distributed protocols, all nodes contend for the medium and can transmit the packet only if it does not hear another transmission. Unfair medium access can occur in distributed protocols under certain circumstances [6], [7]. A decentralized scheme called *Blackburst* was proposed in [8] which minimizes the delay for real time traffic. To access the medium a station sends a black burst by jamming the channel for a period of time. The length of the black burst is determined by the time the station has been waiting to access the medium. Vaidya et al. proposed a QoS scheme in terms of fairness in distributed fashion by allocating bandwidth in proportion to the *weights* of the flows sharing the channel [9]. Hang Su et al. proposed self-adjusting contention window algorithm which modifies the back-off window based on the number of packets transmitted and dropped [10].

The main problem of the above mechanisms is that they would require changes in the MAC firmware. This means that users have to buy new network cards with the modified firmware. This, obviously is an impractical proposition. A more practical solution is to have a QoS scheme which can work on top of the standard IEEE 802.11 MAC layer. Neighborhood Proportional Delay Differentiation (NPDD) is a mechanism that can work on top of any MAC protocol in a wireless device [1]. In NPDD, traffic is put into different classes. Each class is assigned a delay differentiation parameter (DDP). The average delay of packets in different classes is in proportion to their respective DDPs. NPDD scheduler uses Waiting Time Priority (WTP) algorithm [11]. If only NPDD is used, then a flow having a delay bound may not fit into a particular class properly. If the flow is assigned a higher priority, then the actual delay suffered may be much below the delay bound. On the other hand, if it is assigned the next lower class, the delay may be more than the delay bound. Hence Dynamic Class Selection (DCS) is usually employed along with NPDD. DCS runs a periodic decision process to determine if class of a flow needs to be changed. Changing class of a flow makes better utilization of bandwidth while meeting the flow's delay bound.

We implemented the NPDD and DCS mechanism in linux kernel to provide QoS to voice applications. But during our experiment we noticed that DCS could not provide required QoS to high priority flow at a high load condition. The reason is that DCS moves a flow to higher class without considering how the QoS of the higher class will get affected after the move. Hence we devised an *Adaptive Class Selection* (ACS) mechanism that addresses the above drawback of DCS. We have also proposed and implemented a simple measurement based distributed Call Admission Control mechanism to provide QoS assurance at all nodes.

II. Proportional Delay Differentiation

In this section we provide a brief overview of two different Proportional Delay Differentiation (PDD) paradigms, since our work is based on it. For more details, readers can refer to [1].

The PDD service model supports N classes relatively ordered in per-hop packet queueing delays at any node k. At node k, the packets with higher priority experience low delay than packets with lower priority. The proportionality between delays of different classes can be tuned by the network designer with a set of class delay differentiation parameters (DDP).

Let $1 = \delta_1 > \delta_2 > ... > \delta_N > 0$ be the DDPs defined by the network designer for N classes. Let d_i^k be the average queueing delay of class i packets at node k. Then the normalized average queueing delay $\tilde{d_i^k}$ for all classes at node k should be equal, where the normalized average queueing delay of flow i at node k is given by

$$\tilde{d_i^k} = d_i^k/\delta_i. \tag{1}$$

At a PDD node the following holds

$$d_i^k/d_j^k = \delta_i/\delta_j \tag{2}$$

for all classes i and j.

Neighborhood PDD (NPDD) is a service model that uses PDD paradigm along with Waiting Time Priority (WTP) [11], [1]. In NPDD, the head-of-line packet of a class i is assigned a waiting time priority $w_i(t)$ and the scheduler always schedules the highest priority head-of-line packet for transmission.

Applications using NPDD service may have their delay much below the delay bound or much above the delay bound. Hence NPDD by itself may not be an efficient mechanism. DCS mechanism addresses this issue by changing the priority of packets dynamically based on the delay suffered by the application. Delay of packets of a flow is measured periodically. If delay of a flow exceeds (falls below) its delay bound for a certain number of successive time periods the flow is promoted (demoted) to the higher (lower) class. For a formal description of DCS algorithm please refer to [1].

III. Adaptive Class Selector

The advantage of DCS is that it can utilize the bandwidth efficiently to meet the delay guarantees of the applications at low load. But the main problem of DCS is that its performance deteriorates at very high load to provide assurances proportionally. At high load, there is a possibility that many flows may not meet their delay bound and therefore may go to higher classes. This will cause the higher class flows to have high delays. Hence, before a flow is promoted to a higher class, it should be made sure the move does not affect other flows so much that they miss their delay bounds. To this end, we have proposed an Adaptive Class Selector (ACS) mechanism in this paper so that the above situation is taken care of. We propose two such algorithms to adaptively change the class of a flow.

A. *Network Model*

The network model considered for this study is a conventional infrastructure based IEEE 802.11 DCF network. The wireless nodes host applications with end-to-end communication flows through AP. Each node has an implementation of ACS architecture shown in Figure 1. For our ACS algorithm, we have implemented the Proportional Average Delay (PAD) Scheduler proposed in [11]. PAD services packets in N classes and realizes *proportional average per hop delays* among them locally at each node. But PAD uses average delay (calculated over all the packets in a class) and calculates the normalized average delay of a class (whereas NPDD applies WTP to head-of-queue packet). The packet at the head of queue of the class having the largest normalized average delay is then scheduled by PAD scheduler. Table I summarizes the notations used in this paper.

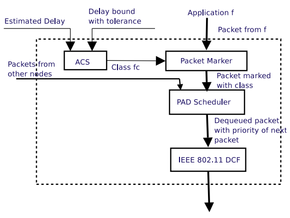

Fig. 1. Architecture of ACS framework

B. *Benefit based ACS*

This version of the ACS is based on what we refer to as *benefit* cost. Each class at a node maintains *benefit* cost which is the cumulative sum of the difference between the delay bound and end-to-end delay suffered by flows in that class. At the k^{th} period for application f, the *benefit* cost of a class $C(k\tau)$ is defined as follows

$$B_{C(k\tau)} = \sum_{f=1}^{M} \hat{D}_f - \sum_{f=1}^{M} D_f(k\tau) \tag{3}$$

The pseudo code for this version of ACS, called Benefit based ACS (B-ACS), is shown in Algorithm 1.

Algorithm *B-ACS* essentially promotes a flow (Step 2) to next class if the new class has a positive benefit and the flow violates its delay bound for K_I consecutive period. If the flow has been enjoying too less delay for K_D consecutive periods, it is demoted to the lower class.

Notation	Description
$C(k\tau)$	Class of a flow at time τ and period k.
$D_f(k\tau)$	End-to-end delay of flow f at time τ and period k
$\hat{D}_f$	Delay Bound of flow f
φ_f	Delay Tolerance of flow f
$D_{C(k\tau))}$	Sum of end-to-end delays of flows in class $C(k\tau)$
$DB_{C(k\tau))}$	Sum of delay bounds of flows in class $C(k\tau)$
DB	Sum of delay bounds of all flows at a node
$Delay$	Sum of end-to-end delays of all flows at a node
$M_{C(k\tau)}$	Number of flows in class $C(k\tau)$
$d_{C(k\tau)}$	Queueing delay of class $C(k\tau)$
$B_{C(k\tau)}$	Benefit cost of class $C(k\tau)$
$\lambda_{C(k\tau)}$	Arrival rate of class $C(k\tau)$
q_{ag}	Average (across all classes) aggregate queue size
$q_{C(k\tau)}$	Average queue size at class $C(k\tau)$
$\delta_{C(k\tau)}$	Delay differentiated parameter of class $C(k\tau)$
$GB_{j,k}$	Global Benefit of node j stored at node k

TABLE I
List of Notations Used

Algorithm 1 $B - ACS(C(k\tau), D_f(k\tau), \hat{D}_f, \varphi_f)$

```
 1: if ($D_f(k\tau) > \hat{D}_f$ for $K_I$ consecutive periods) then
 2:   $C((k+1)\tau) = \min(C(k\tau)+1, N)$;
 3:   if $B_{C((k+1)\tau)} > 0$ then
 4:     $B_{C((k+1)\tau)} += DB_f - D_f(k\tau)$;
 5:     $B_{C(k\tau)} -= DB_f - D_f(k\tau)$;
 6:   else
 7:     $C((k+1)\tau) = C(k\tau)$;
 8:   end if
 9: else if (($D_f(k\tau) < \hat{D}_f + \varphi_f$) for $K_D$ consecutive periods) then
10:   $C((k+1)\tau) = \max(C(k\tau)-1, 0)$;
11:   $B_{C((k+1)\tau)} += \hat{D}_f - D_f(k\tau)$;
12:   $B_{C(k\tau)} -= \hat{D}_f - D_f(k\tau)$;
13: else
14:   $C((k+1)\tau) = C(k\tau)$;
15:   $B_{C((k+1)\tau)} += d_f(k\tau) - d_f((k-1)\tau)$;
16: end if
```

C. *Estimation based ACS*

The problem with B-ACS algorithm is that it allows a new flow into class $C(k\tau)+1$ from $C(k\tau)$ even if the *benefit* cost of class $C(k\tau)+1$, $B_{C(k\tau)+1}$, is slightly greater than 0. This may cause the violation of delay assurances of class $C(k\tau)+1$. E-ACS algorithm solves this problem by predicting the expected increase in average queueing delay of class $C(k\tau)+1$ due to the class change of flow f from class $C(k\tau)$ to $C(k\tau)+1$.

In this algorithm, the delay bound requirement for each flow in a class is not checked, but the cumulative delay bound, $DB_{C(k\tau)} = \sum_{f\in C(k\tau)} \hat{D}_f$, of a class $C(k\tau)$ is checked. Since all flows in a class have same queueing delay, the expected cumulative end-to-end delay of class $C(k\tau)$, $D_{C(k\tau)} = \sum_{f\in C(k\tau)} D_f(k\tau)$, is calculated using expected increase in the queueing delay and the number of flows in that class. This can be used to check whether the cumulative delay bound of all the flows in a class can be met or not.

Expected queueing delay of a class can be calculated as follows. Let λ_i be the aggregate arrival rate of class i. Using Little's law[12], [13] and equation (2), the queueing delay of class i is

$$d_i = \frac{\delta_i q_{ag}}{\sum_{i=0}^{N-1} \lambda_i \delta_i} \tag{4}$$

Suppose a flow in class i with rate λ_f changes to class $i+1$ then the effective delay of class i becomes

$$d'_i = \frac{\delta_i q_{ag}}{\sum_{i=0}^{N-1} \lambda_i \delta_i + (\delta_{i+1} - \delta_i)\lambda_f} \tag{5}$$

From equation (4) and (5) the new queueing delay of class i can be expressed as follows

$$d'_i = d_i \frac{\sum_{i=0}^{N-1} \lambda_i \delta_i}{\sum_{i=0}^{N-1} \lambda_i \delta_i + (\delta_{i+1} - \delta_i)\lambda_f}$$

$$d'_i = d_i \frac{\delta_i q_{ag}}{\delta_i q_{ag} + (\delta_{i+1} - \delta_i)\lambda_f d_i} \tag{6}$$

Algorithm 2 $E - ACS(C(k\tau), D_f(k\tau), \hat{D_f}, \varphi_f)$

```
1:  if ($D_f(k\tau) > \hat{D_f}$ for $K_I$ consecutive periods) then
2:    $C((k+1)\tau) = \min(C(k\tau)+1, N)$;
3:    $d'_{C((k+1)\tau)} = d_{C((k+1)\tau)} \frac{\delta_{C(k\tau)} q_{ag}}{\delta_{C(k+1)\tau} q_{ag} + (\delta_{C(k\tau)+1} - \delta_{C(k\tau)}) \frac{q_{C(k\tau)}}{M_{C(k\tau)}}}$;
4:    $D'_{C((k+1)\tau)} = D_{C((k+1)\tau)} + (d'_{C(k+1)\tau} - d_{C((k+1)\tau)}) * M_{C(k\tau)}$;
5:    $benefit = DB_{C((k+1)\tau)} - D'_{C((k+1)\tau)}$;
6:    if ($benefit >= 0$) then
7:      $tot\_benefit = DB - Delay - (DB_{C(k\tau)} - D_{C(k\tau)}) - (DB_{C((k+1)\tau)} - D'_{C((k+1)\tau)})$;
8:      if ($tot\_benefit < 0$) then
9:        $C((k+1)\tau) = C(k\tau)$;
10:     end if
11:   else
12:     $C((k+1)\tau) = C(k\tau)$;
13:   end if
14: else if ($D_f(k\tau) < \hat{D_f} + \varphi_f$ for $K_D$ consecutive periods) then
15:   $C(\tau+1) = \max(C(\tau)-1, 0)$;
16: else
17:   $C(\tau+1) = C(\tau)$;
18: end if
```

The average backlog due to flow f can be approximately calculated as $\lambda_f d_i = \frac{q_i}{n_i}$, where q_i is the backlog of class i and n_i is the total number of flows in class i.

Hence equation (6) becomes

$$d'_i = d_i \frac{\delta_i q_{ag}}{\delta_i q_{ag} + (\delta_{i+1} - \delta_i)\frac{q_i}{n_i}} \quad (7)$$

Using equation (7) we can predict the approximate increase in the queueing delay of class $i+1$.

$$d'_{i+1} = d_{i+1} \frac{\delta_i q_{ag}}{\delta_i q_{ag} + (\delta_{i+1} - \delta_i)\frac{q_i}{n_i}} \quad (8)$$

The new in end-to-end delay (used in Step 4) is defined as follows

$$D'_{i+1} = D_{i+1} + (d'_{i+1} - d_{i+1}) * n_{i+1} \quad (9)$$

If expected end-to-end delay, D_{i+1}, of class $i+1$ after flow f moves from class i to $i+1$ is greater than the total delay bound, DB_{i+1}, of class $i+1$ then the flow f will not be allowed to change its class. Otherwise, flow f is promoted to class $i+1$. Delay bound of all the classes is checked in Step 7 because the total delay bound (of all the classes) and total end-to-end delay of all the flows at the node is considered.

IV. Connection Admission Control

Both DCS and ACS try to meet delay assurances only at a single node. They are not sufficient for providing delay assurances at all nodes in a IEEE 802.11 LAN, because they do not have congestion information about other nodes. In a shared medium like IEEE 802.11, admitting a flow at one node can affect the performance of another node. Hence new flows should not be admitted to the network (at any node) if the any other node in the network is not able to provide QoS assurance after the flow is admitted.

Violation of QoS assurances in the Wireless LAN may occur due to the following reason. The bit rate of a node suddenly decreases due to some noise in the environment, local mobility of a wireless node, or signal interference with other access points at a wireless node. Hence, QoS assurance is violated if the number of packets sent are less than the total aggregate arrival assured at that node. Modelling this phenomenon is quite complex. Hence, we felt a Measurement based Distributed Admission Control (MDAC) is more appropriate in this scenario. This mechanism captures the current situation in the LAN and takes appropriate action. MDAC captures the fluctuation in output rate in *GlobalBenefit* (benefits of all the nodes in the LAN) cost.

A. MDAC

Nodes run in promiscuous mode to implement MDAC. A node j embeds its QoS assurance information, *GlobalBenefit*, in each packet and sends it over the network. Whenever a packet is received at a node k from a neighboring node j, the *GlobalBenefit* value of node j (GB_j) is retrieved from the packet header and it is updated as a weighted average in the existing neighbor entry at node k as follows,

$$GB^{new}_{j,k} = \alpha * GB^{prev}_{j,k} + (1-\alpha) * GB_j, \quad 0 \le \alpha \le 1 \quad (10)$$

Each node k maintains $GB_{j,k}$. If no corresponding entry is found, new entry is created and *GlobalBenefit* value is stored using the equation (10). If node k does not receive a packet from a neighboring node j for a certain period then the entry $GB_{j,k}$ is removed.

Algorithm 3 $MDAC(C(\tau), \hat{D_f}, \varphi_f, \lambda_f)$

```
1:  /* $l$ is the number of neighbors of node $k$ */
2:  for all $j \leftarrow 1$ to $l$ do
3:    if ($GB_{j,k} < 0$) then
4:      Reject the flow and return
5:    end if
6:  end for
7:  if ($benefit[C(\tau)] > 0$ && $GB_{k,k} > 0$) then
8:    put the flow information in a new flow entry
9:    Accept the flow and return
10: else
11:   Reject the Flow and return
12: end if
```

V. Implementation in Linux Kernel

This section describes the implementation of NPDD mechanism in linux kernel. Due to space limitation we have not provided many internal details of the implementation. For more details readers are referred to [14]. The implementation is independent of kernel version and is implemented as kernel modules [15]. The advantage of the kernel module is that it makes installation easy and avoids kernel recompilation.

A. Protocol Stack

The protocol stack of a node in NPDD Network is shown in Figure 3. The NPDD Network node consists of a *virtual network device* (NPDD device), a *real network device* (802.11DCF) and a *Packet Listener* in the linux kernel. *NPDD device* is a *virtual network device* which captures the outgoing packets to include *NPDD header* and sends packets by changing the packet type to *npdd*. *NPDD header* contains NPDD specific information like class of a flow, *GlobalBenefit* of a node, one way delay of a flow etc. The *Packet Listener* listens for *NPDD* packets. *NPDD device* resides below the network layer whereas the *Packet Listener* resides at network layer. The device driver for *NPDD device* is a kernel module which registers itself as a network device in the kernel. It works above a real wireless device, e.g. wlan0, in the kernel, since it transmits and receives packets through the wireless device.

The *Packet Listener* is also a part of the kernel module which receives packets from network card. Packet reception by a virtual device driver is not possible in linux kernel since there are no hooks in the kernel for virtual device that can be invoked when a packet is received. The only functions provided are the packet reception functions of the network layer code. There are a few alternatives to this including tracing the interrupt that is used by the device and responding to that interrupt. But such mechanisms make the code very much device dependent. Our design is completely independent of the lower level hardware.

B. Architecture of NPDD Device in Linux Kernel

The detailed architecture of the NPDD device driver is shown in Figure 2. In linux kernel, each network device is attached to a queueing discipline to buffer incoming packets. The default queueing discipline is First In First Out (FIFO).

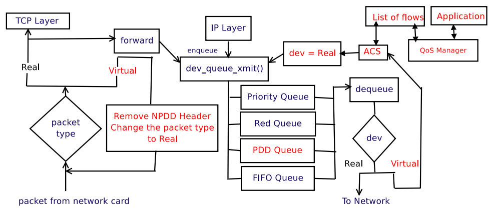

Fig. 2. Architecture of NPDD Device Driver in Linux Kernel

Flow Id	NPDD class	Packet Distribution	Mean Packets	Delay Bound
1	0	Exponential	45	2500ms
2	1	Exponential	45	500ms
3	2	Exponential	45	325ms
4	3	Constant	33	125ms
5	0	Exponential	45	3000ms
6	1	Exponential	45	700ms
7	2	Exponential	45	400ms
8	3	Constant	33	150ms

TABLE II
TRAFFIC SPECIFICATION OF THE FLOWS USED IN OUR EXPERIMENT

A packet destined to the virtual network device (*npdd0*) is enqueued at device layer by the function `dev_queue_xmit()`. Packets from IP layer is enqueued in the queueing discipline attached to *npdd0*. A dequeue daemon is invoked to dequeue a packet whenever the network card is ready to send a packet. If the device is real, then the packet is sent to the network. Otherwise, the packet is forwarded to ACS module. ACS module may change the class of the flow according to the class selection algorithm used (e.g. B-ACS). Now the device associated with the packet is changed to real network device, e.g. wlan0, and `dev_queue_xmit()` function is called. This function enqueues the packet in the queueing discipline attached to the real network device, wlan0. Finally, the packet is dequeued and sent to the network.

When a packet is received from the network by the real network device driver, it does all the processing of removing hardware headers and hands it over to the appropriate network layer by calling the function `netif_rx()`. The *packet type* is used to identify which network layer protocol the packet should be handed over to. If the packet type is virtual then NPDD header will be removed and will be forwarded to corresponding network layer protocol.

C. QoS Manager

QoS Manager is the module with which application programs communicate to ask for new connections. QoS manager runs admission control mechanism, MDAC, to reject or accept a new connection. If MDAC returns *accept* then the connection is admitted and the flow information is added to the list of existing flows. Once admitted, the application can start sending packets. QoS Manger was implemented in network device driver. A flow can communicate with the virtual device driver kernel module for flow addition, deletion and status information through the device generic function `do_ioctl()`. A flow sends its flow information through `do_ioctl()` to linux device module. Then the device module invokes *QoS Manager* to add, delete or get status information of a flow. When an application is finished, it calls `do_ioctl()` to delete flow information and corresponding resources.

VI. EXPERIMENTAL TESTBED & RESULTS

All the wireless nodes used for our experiments are Pentium 4 based PCs running Linux Debian 3.1 kernel version 2.4.27-3-386. All the wireless nodes are equipped with D-Link 520 wireless cards based on IEEE 802.11b DCF [16]. The AP is a D-Link 1000AP with a bit rate of 11Mbps. Distributed Internet Traffic Generator(D-ITG) is used for generation of traffic [17]. D-ITG has support to generate VoIP calls with various codecs, with Voice Activity Detection (VAD) and compressed RTP.

A. Topology and Parameters Used

In this experiment, we consider an IEEE 802.11b based WLAN with 3 wireless nodes located within the radio range

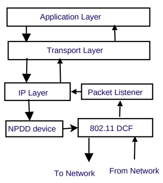

Fig. 3. Protocol Stack at NPDD node

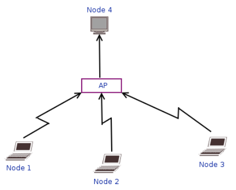

Fig. 4. Topology Used in Experimental Testbed

of an AP. The topology of the wireless network for the experiments is shown in Figure 4. Node 4 is a wired node and remaining all nodes are wireless systems operating in IEEE 802.11 DCF mode. Node 4 is the destination node for the traffic generated by D-ITG traffic generator from all the three wireless nodes. All wireless nodes are within each other's radio range. We have used delay utility as the performance metric for comparison. Delay utility is defined as the ratio of number of packets meeting its delay bounds to the total number of packets sent.

Two mechanisms, B-ACS and E-ACS were evaluated in the experiments and their performance is compared with *baseline* and DCS scheme. The *baseline* scheme represents a best effort service with a single FIFO and 802.11b DCF at all wireless devices. *DCS* scheme consists of 4 prioritized classes with 802.11 DCF at each node and *PDD* scheduler is used to service the packets. The proposed mechanisms, *B-ACS* and *E-ACS*, consists of 4 classes with 802.11 DCF at each wireless device. The parameters for the evaluation of schemes are shown in Table IV. In our experiments the delay tolerance is set to 25% of the Delay Bound of application.

The traffic pattern is modeled as follows. Each node initiates 8 UDP flows between itself and the node 4 via the AP. The packet arrival patterns and the delay bound requirement for each flow at each node is described in Table II. UDP flows are exponentially distributed with mean inter departure time(IDT) of 22.22ms (45 packets/sec) for class 0, 1 and 2 with a packet size of 512 bytes. For class 3, two voice flows with G.729.3 codec were generated. Each codec type sends 33 packets/sec.

Figure 5 shows the delay utility of three nodes achieved by the UDP flows. The delay bounds on the X-axis corresponds to the delay bound of flows listed in Table II. It is clear that *baseline* scheme is not suitable for meeting delay requirement when delay bound is low. *DCS* scheme has poor delay utility for flows with lower delays. For example, for flows with delay bounds 125ms and 150ms DCS has worse delay utility than *B-ACS* and *E-ACS*. The reason for this is that the class 3 flows (which have these low delay bounds) (refer to Table II) would be demoted to smaller class by DCS when their benefit is high. But once they go to lower class, their delay increases significantly (due to the proportional parameter chosen), so packets would start missing delay bounds. Then the flows are again promoted back to higher class. When delay bound is little loose (e.g. 500ms, 700ms), even if the flow is demoted, due to loose bound, fewer packets miss deadline. Hence DCS performs better than B-ACS and E-ACS in these ranges of delay bound. Since *B-ACS* and *E-ACS* algorithms check for benefit values before promoting or demoting flows, they give higher delay utility for class 3 flows (flows with delay bounds of 125ms, 150 ms). *E-ACS* provides higher utility than *B-ACS* since it takes into account benefit of all the classes.

B. Admission Control

In this Section we provide experimental results of MDAC call admission algorithm. We have used the same topology as mentioned before. At each node 70% of the bandwidth was occupied with class 0, class 1 and class 2 traffic. Remaining bandwidth is used up by the voice calls. MDAC is tested along with *DCS*, *B-ACS* and *E-ACS* algorithms. The arrival of voice calls follow Poisson distribution and life time of voice calls is exponentially distributed. The mean lifetime of each call is 180sec. The mean arrival rate of calls is changed from 0.75 arrivals/minute to 1.5 arrivals/minute.

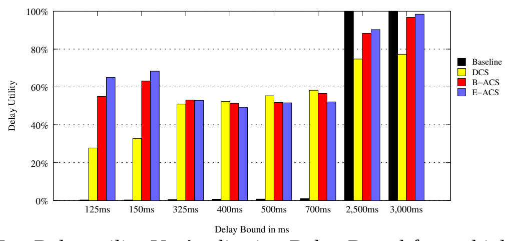

Fig. 5. Delay utility Vs Application Delay Bound for multiple nodes

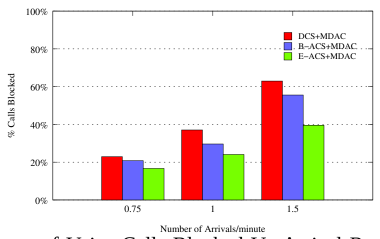

Fig. 6. Percentage of Voice Calls Blocked Vs Arrival Rate of Voice Calls

Number of Voice Flows/ ALDB	2 (21.56% load)		3 (32.35% load)		4 (43.14% load)		6 (64.71% load		8 (86.27% load)	
	DB 125ms	DB 150ms	DB 125ms	DB 150ms	DB 125ms	DB 150ms	DB 125ms	DB 150ms	DB 125ms	DB 150ms
150ms	72.78	76.65	28.00	44.36	13.09	28.94	9.68	11.10	4.16	5.82
200ms	98.74	95.60	81.81	78.78	69.27	67.07	26.67	26.84	11.11	10.45
250ms	99.87	99.47	97.17	95.09	91.68	89.53	54.32	54.73	19.26	18.59
300ms	100.0	99.67	99.27	97.25	96.60	94.57	64.35	64.70	22.15	21.59
350ms	100.0	99.87	100.0	98.25	98.37	96.93	74.56	74.90	26.74	25.98
400ms	100.0	99.97	100.0	98.90	99.33	98.32	79.79	79.85	32.53	31.42

TABLE III
DELAY UTILITY VARIATION WITH NUMBER OF VOICE FLOWS WITH BOUND ON END-TO-END DELAYS

Scheme	Baseline	DCS,B-ACS, E-ACS
Delay Tolerance	N/A	$.25x$
NPDD Classes	N/A	4
DDP δ_i, i $\in$ 1,2,3,4	N/A	$[1, \frac{2}{5}, \frac{4}{25}, \frac{8}{125}]$
DCS sensitivity parameters(K_I, K_D)	N/A	(1,1)
DCS period(seconds)	N/A	1
Packet size	512	512
Per-class max queue size	2000	500
PHY Specification	802.11b	802.11b
MAC Specification	DCF	DCF

TABLE IV
PARAMETERS OF EVALUATED SCHEMES

The percentage of calls blocked for all the mechanisms with MDAC is shown in Figure 6. Number of calls blocked will be more in case of *DCS*. The fluctuation of class is more in case of DCS, since it does not check the benefit of the target class. But in case of B-ACS, the benefit value of the target class is checked before a class change is made. But a class change in a proportional differentiation based network affects all the classes. Since E-ACS checks the *GlobalBenefit*, it indirectly checks the condition of all the classes (as opposed to only target class). Hence E-ACS performs the best.

From the results presented so far it is clear that E-ACS is best suited for voice calls which have low delay requirement. In our next experiment, we vary the load due to voice calls (by increasing the number of voice calls) and observe the delay utility of E-ACS algorithm. In this experiment, we have set two kinds of delay bounds. One is the application level delay bound (ALDB), which is the delay bound to be met by the application. The other is the network level delay bound which is used while executing E-ACS algorithm. The delay utilities for various flows with delay bounds can be found in Table III. Now given these results, an administrator can estimate the delay bound of voice applications as follows. For a given application level delay bound he should decide how much should be the network level delay (which will be used by the E-ACS algorithm), for which he may have to figure out various delays at the host. The network level delay bounds used for voice flows in the experiments are 125ms and 150ms. The table gives the choice of choosing the network level delay bound (a parameter for E-ACS algorithm) based on desired delay utility. For example if the system will have 3 voice calls (32.35% load) and application level delay of 300ms is required and a delay utility of 100% is desired, then the administrator has to go for 125ms as network level delay.

We also have experimental results with regards to delay and jitter in the network. Effectiveness of our MDAC and ACS was demonstrated by running one of the widely used VoIP application *gnomemeeting* on our testbed. We are not able to present those results due to space constraints. Those results can be found in [14].

VII. CONCLUSIONS & FUTURE WORK

We have implemented DCS mechanism for providing QoS to VoIP applications. But we found that DCS only works fine at low load. At high load DCS fails to provide QoS because DCS promotes a flow without considering how the change would affect the QoS of the higher class. We have proposed two Adaptive Class Selector(ACS) mechanism in order to provide QoS assurances at high load. Experimental results show that ACS mechanisms work better than DCS to provide QoS assurances for higher classes. For low delay applications E-ACS is the most suitable mechanism in terms of delay utility. To assure QoS at all nodes in a WLAN, a measurement based Call Admission Control called MDAC is proposed. Experimental results shows that MDAC is able to accept more connections with ACS mechanism than DCS mechanism.

We intend to extend our work to provide delay assurance in a multi-hop wireless network. The admission control algorithm we have presented cannot handle hidden terminal problem. We would like to address it in the current implementation. We also would like to implement our MDAC in an Access Point instead of mobile client. That would require a linux based Access Point (so that we can put our changes in the kernel). This would also take care of hidden terminal problem.

REFERENCES

[1] Kuang-Ching Wang Parameswaran Ramanathan. A Cross-layer Approach for Concurrent Delay and Throughput Assurances in Multihop Wireless Hotspots. In *WMASH*, 2003.

[2] IEEE. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications , IEEE Standard 802.11, June 1999.

[3] C. S. Chang and K. C. Chen. Service curve proportional sharing algorithm for service-guaranteed multi-access in integrated-service wireless networks. In *Proceedings of IEEE VTC*, volume 2, pages 1288 –1293, 1999.

[4] C. S. Chang and K. C. Chen. A unified wireless LAN architecture for real-time and non-real-time communication services. In *IEEE/ACM Transactions on Networking*, pages 44 – 59, February 2000.

[5] M. Veeraraghavan, N. Cocker, and T. Moors. Support of voice services in IEEE 802.11 wireless LANs. In *Proceedings of IEEE INFOCOM*, volume 1, pages 488 – 497, April 2001.

[6] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc network. In *IEEE Communications Magazine*, volume 39, pages 130 – 137, June 2001.

[7] Jinyang Li, Charles Blake, Douglas S., J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad-hoc wireless networks. In *Proceedings of ACM Mobicom*, 2001.

[8] A.S. Krishnakumar J.L. Sobrinho. Quality of Service in ad hoc carrier sense mutilple access networks. In *IEEE Journal on Selected Areas in Communications*, volume 17(8), pages 1353–1368, 199.

[9] Nitin H. Vaidya, Paramvir Bahl, and Seema Gupta. Distributed fair scheduling in a wireless LAN. In *Mobile Computing and Networking*, pages 167–178, 2000.

[10] Hang Su and Peiliang Qiu. IEEE 802.11 Distributed Coordination Function : Performance Analysis and Protocol Enhancement. In *AINA*, volume 2, pages 335 – 338, 2004.

[11] Parameswaran Ramanathan Constantinos Dovrolis, Dimitrios Stiliadis. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. In *IEEE Transactions on Networking*, volume 10:1, New York, NY, February 2002.

[12] G. Bloch, S. Greiner, H. Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley and Sons, 1999.

[13] L. Kleinrock. *Queueing systems, Volume II*. John Wiley and Sons, 1976.

[14] S. Pattipaka. *Providing Delay Assurances in Infrastructure based Wireless LANs*. M.Tech. Thesis, Indian Institute of Technology, Bombay, 2005.

[15] Peter Jay Salzman and Ori Pomerantz. *The Linux Kernel Module Programming Guide*. Peter Jay Salzman, 2001.

[16] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 ghz band, IEEE Standard 802.11b, Sept. 1999.

[17] Distributed Internet Traffic Generator. `http://www.grid.unina.it/software/ITG/`.