Lottery Scheduling for Flexible and Fine-grained Bandwidth Management in Wireless LANs

Shravan Rayanchu Sharad Saha
Department of Computer Sciences, University of Wisconsin, Madison, WI 53706, USA
{shravan, sharad}@cs.wisc.edu

ABSTRACT

Emerging applications such as VoIP, Multimedia streaming require certain Quality of Service(QoS) guarantees in order to satisfy the requirements of their users. In today's wireless LANs the administrator has little or no control over the amount of bandwidth consumed by different flows or even different traffic classes. Flow level control is beyond the capabilities of the current systems and is desirable in networks which have multiple input flows with different QoS requirements. Towards this end, we propose a probabilistic solution based on lottery scheduling between competing flows for a proportional allocation of the wireless bandwidth. Lottery scheduling when used in conjunction with policies (taken as input) provides a simple and scalable solution for flexible and fine-grained wireless bandwidth management. Our solution works across across different protocols and takes into account the traffic characteristics of the competing flows using the concept of ticket adjustments. We have implemented a prototype lottery scheduler on the madwifi wireless driver and found that it provides a flexible control over the wireless bandwidth, achieving the desired throughput proportions between the flows in the network.

1. INTRODUCTION

Wireless bandwidth is a scarce resource. As users experience the convenience of wireless connectivity and as wireless LANs seek to support advanced multimedia applications such as interactive lecture series, voice over IP (VoIP) etc, Quality of Service (QoS) is becoming increasingly important in 802.11 networks. In today's wireless LANs the administrator has little or no control over the amount of bandwidth consumed by different flows or even different traffic classes. Flow level control is beyond the capabilities of the current systems and is desirable in networks which have multiple input flows with different QoS requirements. Packet scheduling is one approach to address these requirements. However, the scheduling algorithms chosen to multiplex the wireless medium can have a considerable impact on the network throughput and the response time experienced by the applications. Existing 'bandwidth share' schedulers for wireless networks try to provide QoS by dividing the Internet traffic into broad classes with absolute priorities schemes [1]. However this limits them to a coarse control over the traffic i.e it is not possible to assign dynamic policies on a per flow basis. Current applications like media streaming might require flexible and dynamic control over scheduling policies. In the current standards, the priorities are absolute for a given traffic class and there is no concept of dividing the wireless bandwidth into desired proportions. The current schemes are also oblivious to changing packet size in a flow which might have a considerable impact on the achieved throughputs for long running flows.

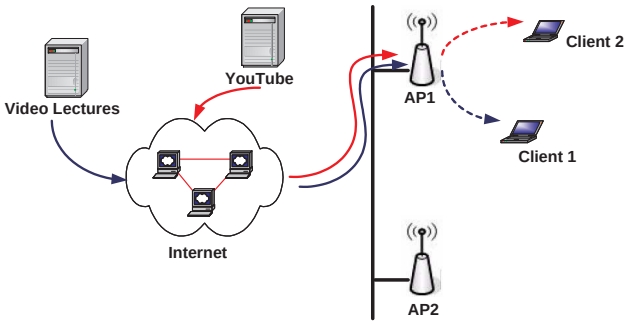

Figure 1: A Motivating Example.

Let us consider a motivating example in order to better understand why such a control over the wireless bandwidth is necessary. Consider the following use case shown in Figure 1 where a client (Client 1) is connected to Access point(AP) in an enterprise wireless LAN. The client starts streaming a live lecture series and is initially satisfied with the quality of video. This is mainly because the client does not have to share the wireless medium. Now consider the situation where another client (Client 2) associates to the same access point and starts streaming a Youtube video. The quality of both the video streams tends to degrade as a result of media contention in the wireless medium. Ideally, most of the proportion of the bandwidth should be allocated to Client 1 and if there is any residual bandwidth, it can

be allocated to Client 2. This is not possible to achieve with existing systems; IEEE 802.11e for example would classify all the video traffic into a single class, thus removing the possibility of isolating the video lecture traffic from the Youtube traffic. Only a solution based on per-flow throughput allocation can possibly satisfy the requirements of the network administrator in this case. More specifically, we need a scheduling algorithm that can provide a fine-grained, flexible, proportional share of the wireless bandwidth of a per-flow basis.

To concurrently meet such varied quality of service needs, many different scheduling policies have been proposed in literature. For wireline networks, these policies are generally referred to as Packet Fair Queuing (PFQ) algorithms [8, 6, 13]. However a direct application of these techniques in a wireless network does not result in fairness guarantees as in the wireline network [11].

In this paper we take a probabilistic approach to achieve proportional fairness for QoS provisioning. Our implementation has two main components - packet scheduler and the flow classifier. To provide a fine grained, proportional share of the wireless medium we implement lottery scheduling at the wireless driver. Lottery scheduling at the driver schedules packets at the granularity of flows and achieves throughputs proportional to the relative shares that they are allocated. The shares (or tickets) can be allocated based on the policies of a given network. Our solution works across multiple clients, different protocols and takes into account the traffic characteristics (e.g., changing packet size) of a flow and adjusts the tickets of the flow accordingly. We also have implemented Stride scheduling [17], which is based on deterministic approach. To administer the flows, we develop a policy classifier, which provides a flexible and dynamic allocation of the wireless bandwidth. It provides ACL-like (Access Control Lists) rules to assign dynamic policies and bandwidth shares to different flows. The classifier also maintains per-flow queues and assigns tickets to these different flows. Lottery scheduling is then done amongst these competing flows to decide the winner which gets to transmit its packet. As the lottery scheduling is based on a very simple approach and does not require any state to be maintained at the software driver, the solution is also scalable.

1.1 Roadmap

The rest of the paper is outlined as follows. Section 2 describes the concept of lottery scheduling and how it can be applied for scheduling packets in a wireless LAN. We describe the design and implementation of our scheduling framework and also explain the concept of ticket adjustments to account for traffic characteristics of different flows in the network. In Section 3 we present a detailed evaluation of our implementation and quantify its performance under different scenarios. We present related work in Section 4 and then finally conclude in Section 5.

2. DESIGN AND IMPLEMENTATION

We use a simple approach based on packet scheduling in order to provide a flexible and fine-grained bandwidth management in wireless LANs. Lottery scheduling [16] was introduced in the context of processor scheduling and provides a simple solution based on randomization for proportional share of resources (the CPU) among the contending processes. We adopt this approach for proportional share of bandwidth among contending flows in wireless network.

2.1 Overview of Approach

The wireless network administrator provides a policy file to the system which consists of a set of ($flow$: $ticket$) pairs. We define a *flow* as an entity which can be identified using parameters such as IP addresses, ports, protocols etc. The corresponding *ticket* refers to the proportion of bandwidth to be allocated for that flow. This abstraction allows a flow to be very fine-grained – for e.g. the pair `(SRC_IP 128.105.102.20, SRC_PORT 80, PROTO TCP : 300)` refers to giving a certain proportion (300) to HTTP traffic arising from machine with a specific IP address or it can be coarse-grained – for e.g., the pair `(PORT 5001: 50)` refers to all traffic using port 5001 to be allocated a certain proportion (50). This policy file is input to an access point in the wireless network and whenever a transmission opportunity arises at the access point, the scheduler draws a lottery among the contending flows based on their tickets. The probability of winning a lottery is proportional to the the value of the ticket a flow holds. The winner of the lottery is then provided access to the wireless medium and proceeds with transmitting a packet. We refer the interested reader to [16] where more details about lottery scheduling can be found.

2.2 Why Lottery ?

There were several issues that had to be addressed when developing a solution for bandwidth management in a wireless network. We now discuss what these issues are, and how they were addressed by a solution based on lottery scheduling:

- **Fine-grained allocation:** In our approach to scheduling, we use *flows* as the contenders for the resource (wireless bandwidth). Such an abstraction allows a flow to be fine-grained, for e.g., a flow can refer to the traffic belonging to a specific instance of an application running at a port on a particular IP Address. Thus, instead of coarsely dividing the traffic into small, fixed number of classes

as done IEEE 802.11e, using the general concept of flows is used to achieve fine-grained allocation of bandwidth.

- **Flexibility:** IEEE 802.11e divides the traffic into four broad classes of traffic, the relative priorities of which are fixed. We instead provide a flexible approach to bandwidth management based on *dynamic policies*. The policies are input from a special file which specifies the contending flows and proportions that are to be allocated for a duration of a time. These policies can be changed by the wireless network administrator as and when required depending on the end user requirements.

- **Proportions:** There is no concept of providing proportionally fair allocation of bandwidth in IEEE 802.11e. We assign *tickets* to the contending flows based on the proportions specified in the policy file. Using lottery scheduling based on these tickets helps achieve the required proportion.

- **Traffic characteristics:** It is important to take traffic characteristics such as packet size, bursty nature of traffic etc into account when providing a solution for fair share of bandwidth. The concept of *ticket adjustments* in lottery scheduling provides an elegant method to fit in these parameters in our solution.

- **Scalability:** It is important to note that providing fine-grained allocation might limit the scalability of a system. In particular, since we wanted to provide a per-flow fair share of bandwidth, we have to be careful about the memory and processing overhead involved in maintaining a per-flow state at the access point. This can prove to be a considerable amount of overhead when there are new flows joining the network and some flows leave the network. Lottery scheduling provides a simple and elegant solution which obviates the need for any per-flow state, thus providing a scalable solution.

2.3 Ticket Adjustments

There are cases where adjustments might have to be made to the initially assigned tickets. For example, traffic characteristics might affect the throughput of each flow and might have to be taken into account when providing a mechanism for proportional share of bandwidth. We consider two characteristics – (1) Packet size and (2) Bursty nature of the flows and explore whether they affect the throughputs and if so what adjustments have to made in order to take these into account.

Packet Size: Let us first consider how the packet size of a flow might affect the throughputs of the flows. Consider the case where there are 2 flows $flow_1$ and $flow_2$ which are initially assigned 100 tickets each. The administrator expects each of the flows to achieve similar throughputs when the wireless medium is under contention. Lottery scheduling ensures that this is indeed the case when the packet sizes used in these flows are the same. However, consider the case where $flow_1$ uses a packet size of 500 bytes whereas $flow_2$ uses a packet size of 1000 bytes. In this case, a scheduling scheme agnostic of packet size would simply provide equal transmission opportunities to both the flows. This would result in effective throughput ratio of $flow_1 : flow_2 = 1 : 2$, this is because $flow_2$ sends twice the amount of data $flow_1$ sends on every transmission opportunity. We use the concept of *ticket inflation* [16] to account for this. That is, we adopt an approach of per-packet ticket inflation where we recalculate the tickets of a flow after every transmission. This adjustment in tickets is based on the size of the transmitted packet as is done as show in Equation 1.

$$new\_ticket = \frac{init\_ticket * base\_size}{pkt\_size} \quad (1)$$

In Equation 1, *init_ticket* refers to the value of the initially assigned tickets (based on the required proportion), *base_size* refers to a constant used for ticket adjustments. In our implementation, we take *base_size* to be the Maximum Transmission Unit (MTU) of the wireless link (1500 bytes). Finally, *pkt_size* is the size of the transmitted packet. Using this equation results in $flow_1$ to be allocated 300 tickets and $flow_2$ to be allocated 150 tickets, which effectively gives $flow_1$ double the number of transmission opportunities than that of $flow_2$. This results in the effective throughput ratio of $flow_1 : flow_2 = 1 : 1$, thus resulting in the desired proportions. However, it is important to note that the overall network throughput would now be lesser compared to the case where lottery scheduling is not used. We explore this tradeoff between bandwidth efficiency and fairness more in Section 3.

Bursty Traffic: Another aspect to consider is whether the bursty nature of a flow might affect the resulting proportion. Consider the case where there are two flows $flow_1$ and $flow_2$ both of which send the traffic at the same rate and are assigned equal proportion. However, $flow_1$ is bursty in nature whereas $flow_2$ sends the traffic with constant inter-packet arrival times. Would this affect the throughput proportions of the flows ? This problem has been previously studied in [8], where the authors propose a mechanism for per-flow share of bandwidths among the flows traversing a router. Specifically, they propose using per-flow queues and use a mechanism based on bit-by-bit round robin for achiev-

ing fair share of bandwidth amongst the flows. They consider the case where some flows might not utilize the bandwidth for certain intervals in between (due to bursty nature of the flows). They use a parameter δ which accounts for the 'amount of history' to be considered and give preference to the flows which under utilize their share during that time. They show that when queue size of the flows is not the limiting factor (i.e when the packets are not being dropped) then the proportional share of the bandwidth is achieved amongst the flows even when the utilization of flows is not considered. That is, throughput of the flows is not affected but the average delay encountered by the flows would be affected. The delay for packets belonging to the bursty flow would be reduced when preference is given to the packets from that flow.

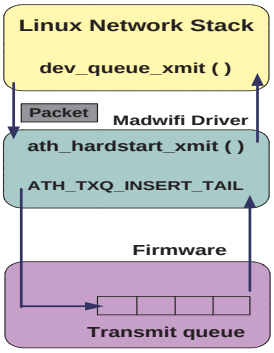

Figure 2: Transmit Path of a Packet in the Current Wireless Stack

A similar approach can be used in our scheduling, where we could keep track of the utilization for each flow and the inflate the tickets accordingly, thus giving more priority to flows which under-utilize the medium:

$$new\_ticket = init\_ticket * util \quad (2)$$

Equation 2 shows how these adjustments can be made. Here *util* refers to the fraction of utilization by a particular flow. We note that this would only affect the average delay encountered by the flows and not the throughput proportion.

2.4 Implementation details

We implemented lottery scheduling on the Linux 2.6.17 kernel running the madwifi wireless driver [2]. The policy file was input using the standard `proc` file system. Figure 2 shows the transmit path in the current implementation of madwifi driver. When a packet (an `skb` buffer) is available for transmission in the Linux networking stack, the function `dev_queue_xmit()` is invoked. This in turn calls the transmit function of the madwifi driver, `ath_hardstart_xmit()`. The driver encapsulates this packet with the IEEE 802.11 MAC header and sets the transmission parameters for this packet (e.g. transmit power, PLY data rate etc) and then handles the packet to the firmware using the function call `ATH_TXQ_INSERT_TAIL` which simply enqueues the packet in the firmware queue (FIFO) and returns. The wait-time for a packet in the firmware queue (i.e. the time spent in the firmware queue before the packet is transmitted in the air) depends on the contention in the wireless medium and whether there are any packets already enqueued in the firmware queue. Once the packet is transmitted in the air, the callback function of the driver (registered with the firmware) is called using `ath_tx_tasklet`, where the necessary parameters to be used for the next transmission are updated.

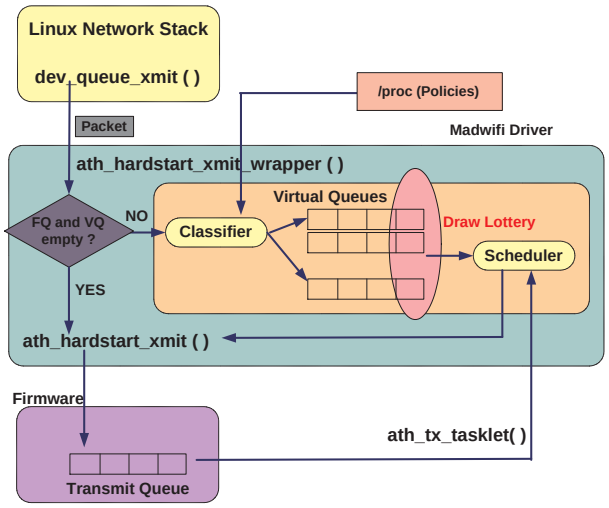

Figure 3: Lottery Scheduling Framework

It is important to note that *currently all the queuing happens in the firmware*, which makes it difficult to implement any form of scheduling at the driver. We therefore needed a mechanism where all the queuing happens in the wireless driver and the firmware essentially acts as a transmit buffer for the packet. Figure 3 illustrates our approach to the problem. We essentially maintain per-flow software queues at the driver. The flows and their proportions are input to the driver based on a policy file using the proc file system. We implement a *classifier* module which takes the policy inputs and is responsible for maintaining the per-flow queues. Whenever a packet is received from the network layer, instead of calling the transmit function of the driver, a wrapper function (`ath_hardstart_xmit_wrapper`) is invoked. The wrapper function then passes the packet to the classifier or immediately calls the transmit function based on the following: (1) If the firmware transmit queue and all the software queues are empty, then the transmit function of the firmware is called and the usual procedure of transmitting a packet follows (2) Otherwise the packet

is sent to the classifier which then enqueues the packet into the appropriate queue based on the input policy. In either case, the function immediately returns before the packet is actually transmitted. Whenever a packet is transmitted by the firmware, the callback function invokes the *scheduler* module. The scheduler then selects a packet from one of the per-flow software queues and calls the transmit function of the firmware. Using this approach allows all the queuing to happen at the software (driver) and the firmware essentially acts as single transmit buffer. Since packets are handed to the firmware only one after the another, there would be delays introduced in the process. We quantify the performance degradation due to this in Section 3.

We have implemented a generic scheduling framework in order to provide flexibility in implementing scheduling decisions. We have implemented (1) Lottery scheduler and (2) Stride scheduler [17] as examples. Stride scheduler basically adopts a *deterministic* approach to achieve a proportional share of the available resource whereas lottery adopts a randomized approach. We compare the performance of both these approaches in Section 3.

3. EVALUATION

In this section, we evaluate the performance of lottery scheduling in presence of multiple clients, different protocols, varying packet sizes, for different traffic types and under lossy wireless links. We also compare the performance of lottery scheduling with that of stride scheduling. Further, we also quantify the overhead due to our implementation of lottery scheduling at the driver level and that due to the maintenance of software queues of packets at the driver.

(a) Single AP - Single Client: We first demonstrate the effectiveness of our approach in a simplistic setting which consists of a single access point (AP) and single client. We set up 2 UDP flows from the AP to client, each using a packet size of 1000 bytes. We use `iperf` to measure the throughput achieved by each of the flows. The rate at which each of the flows send packets is more than the amount that can be sent by the AP, thus resulting in building up of queues at the AP. The tickets are allocated in the ratio of $flow_1 : flow_2 = 1 : 2$. Figure 4 shows the throughputs of both the flows. Lottery scheduling is initially turned off and therefore both the flows achieve equal amount of throughputs. At $t = 10$ seconds, we turn on lottery scheduling (using proc filesystem interface) which results in $flow_2$ achieving double the throughput. Lottery is turned off again at $t = 45$ seconds which results in both the flows sharing the throughputs in equal proportion. Figure 5 shows the ratio of achieved throughputs when lottery is turned on. We observe that using lottery results in a proportion of 1 : 2.018.

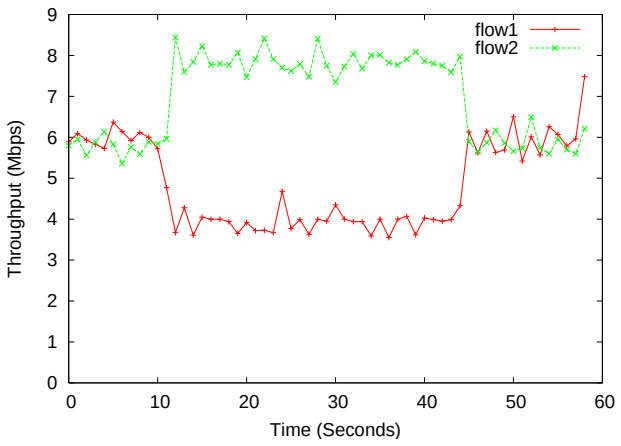

Figure 4: Performance of Lottery Scheduling (Single AP - Single Client)

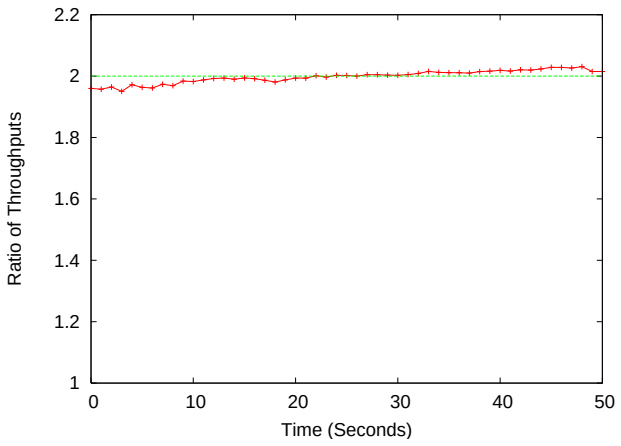

Figure 5: Ratio of Achieved Throughputs

(b) Multiple Clients: We now use a set up with a single AP and two clients. As before we set up UDP flows from the AP to both of the clients. For $client_1$, we set up 2 flows ($flow_1$ and $flow_2$) and for $client_2$ we set up one flow ($flow_3$). The desired proportion between the flows is set to $flow_1 : flow_2 : flow_3 = 1 : 2 : 4$. Figure 6 shows the achieved throughputs when lottery scheduling is turned on at the AP. Initially, only $flow_1$ is active and hence it consumes the entire bandwidth. The second flow $flow_2$ is activated at $t = 20$ seconds and thus it results in the throughput being shared by both flows in ratio of 1 : 2. At $t = 50$ seconds, $flow_3$ for $client_2$ is activated, this results in the throughputs being shared in the desired proportion. The measured throughput proportions for this experiment was 3.98 : 2.01 : 1.

(c) Different Protocols: In this experiment we demonstrate the effectiveness of scheduling in presence

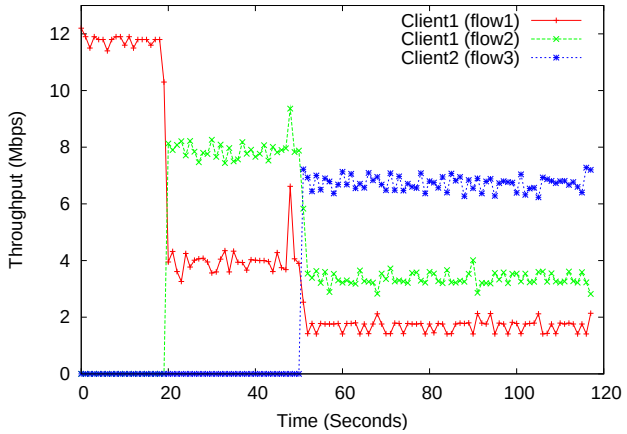

Figure 6: Performance of Lottery Scheduling (Multiple Clients)

of different protocols. We set up 2 flows: $flow_1$ which is a UDP flow from AP to $client_1$ and $flow_2$, a TCP flow from AP to $client_2$. The desired throughputs to be $flow_1 : flow_2 = 1 : 2$. Figure 7 plots the achieved throughputs for this experiment. We observe that initially when lottery scheduling is turned off, $flow_1$ (UDP) takes up most of the bandwidth. This is because, on seeing delays and wireless losses, TCP performs congestion control and backsoff, whereas UDP does not. However, when lottery scheduling is turned on, we see that the desired proportion is achieved. This is mainly because of the maintenance of per-flow queues which isolates the UDP flow and hence does not increase the RTT as perceived by the TCP sender. We observe that the measured throughputs are in the ratio of $1 : 1.93$.

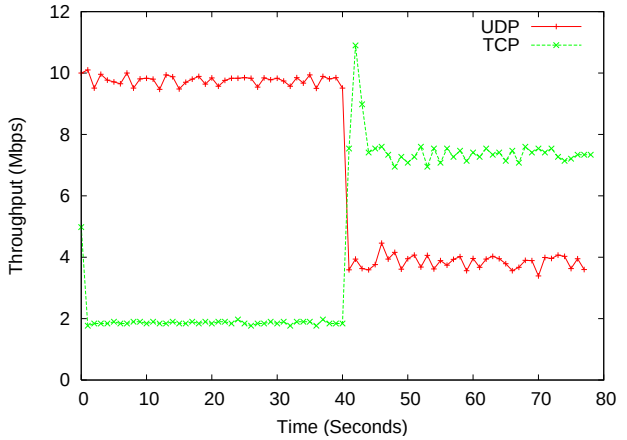

Figure 7: Performance for TCP and UDP

(d) Effect of Packet Size: In this experiment we demonstrate how packet size can affect the throughput proportions of the flows. We set up 2 UDP flows, $flow_1$ from AP to $client_1$ and $flow_2$ from AP to $client_2$. The packet size for $flow_1$ was set to be 600 bytes and the packet size for $flow_2$ was set to be 1200 bytes. The desired proportion given to the lottery scheduler was $flow_1 : flow_2 = 2 : 1$. We plot the measured the throughputs for 3 cases: (a) No scheduling (b) Lottery scheduling without ticket adjustments and (c) Lottery scheduling with ticket adjustments. Figure 8 plots the achieved throughputs in all these cases. Initially when lottery scheduling in turned off, we see that the throughput of $flow_2$ is almost double the throughput of $flow_1$. This is because of the difference in the packet sizes of both these flows. At $t = 40$ seconds, lottery scheduling (without ticket adjustments) is turned on. This results in lottery scheduling giving $flow_1$ double the number of transmission opportunities when compared to that of $flow_2$. However, since this approach is agnostic of the packet sizes, the effective throughput ratios turn out to be $1 : 1$. At $t = 80$ seconds, ticket adjustments (inflation) are turned on which results in recalculation of tickets based on the packet size. This results in lottery giving $flow_1$ four times the number of transmission opportunities of what $flow_2$ receives. This results in the desired throughput ratio of $2 : 1$.

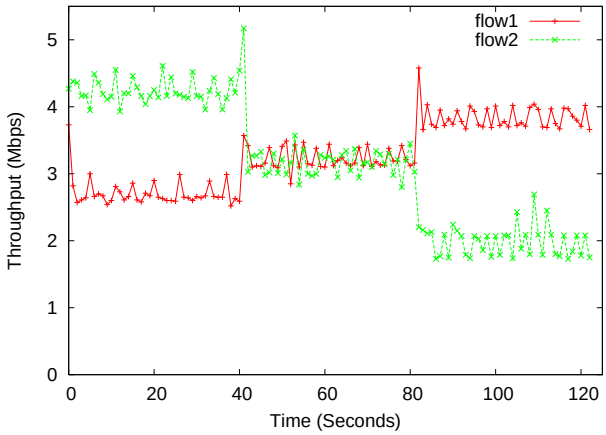

Figure 8: Ticket Inflation: Effect of Packet Size

(e) Efficiency Vs. Fairness: Although, we achieve the required throughput proportion it is important to understand the tradeoff between bandwidth efficiency and fairness. We observe that the aggregate throughput in case (a) is 6.94 Mbps whereas the aggregate throughput for case (c) is 5.83 Mbps. From an efficiency perspective, overall aggregate network throughput increases when $flow_2$ gets more transmission opportunities, because $flow_2$ sends double the amount of data sent by $flow_1$. In case (a) both the flows get equal number of transmission opportunities, however in case (c), $flow_2$ receives much lesser transmission opportuni-

$flow_i$	1	2	3	4	5	6	7	8	9	10
Ratio	1	1.92	2.94	3.98	4.97	6.02	6.95	8.03	8.97	9.93

Table 1: Ratio of throughputs achieved for 10 flows using lottery scheduling. The desired ratio of throughputs was set to be $flow_1 : flow_2 : flow_3 .. : flow_{10} = 1 : 2 : 3 : .. : 10$

ties which results in a decrease in the aggregate network throughput.

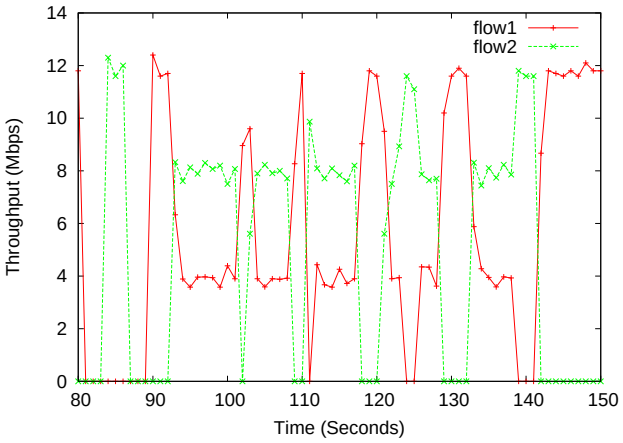

Figure 9: Performance of Lottery Scheduling in presence of Bursty Flows.

(f) Effect on Bursty Traffic: In this experiment, we understand the performance of lottery in presence of bursty traffic. We set up 2 bursty UDP flows from the AP to 2 clients (one flow each). We vary the burst size and the burst inter-arrival times at random. We set the desired proportion to be $flow_1 : flow_2 = 1 : 2$. Figure 9 shows the performance of lottery scheduling in presence of bursty flows. As described in Section 2, ticket adjustments can be used to account for the utilization of flows. However, as noted before, this would only result in change in the average delays experienced by both the flows. The achieved throughputs would not be affected in the presence of backlogged flows where queue size is not the limiting factor (i.e. packets are not dropped). As shown in Figure 9, we see that whenever both the flows contend for the medium (i.e. packets from both the flows are queued up at the AP), lottery scheduling achieves the desired proportion of 1 : 2, whereas when the flows are not contending, then either of the flows is free to take up the entire medium.

(g) TCP and Lossy Links: We now investigate the performance of lottery scheduling when using TCP in presence of lossy wireless links. In this experiment we set up 2 TCP flows from AP to 2 clients. The desired throughput proportion is set to $flow_1 : flow_2 = 2 : 1$. Figure 10 shows the performance of lottery scheduling in presence of lossy wireless links. We observe that

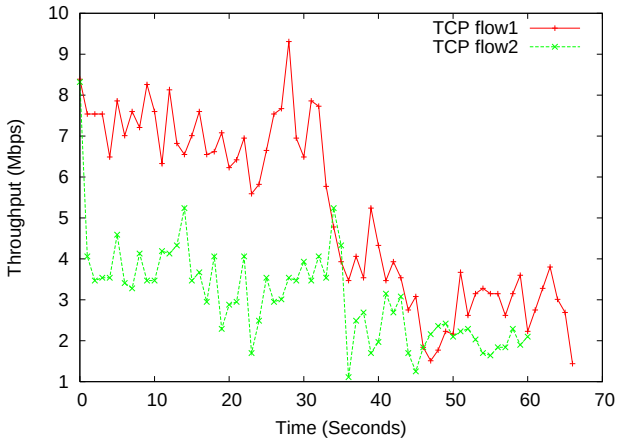

Figure 10: Performance of Lottery when using TCP in Presence of Lossy Links

before ($t = 35$) seconds, lottery performs pretty well achieving the throughputs in desired proportion. However, when the clients are moved farther (thus increasing the losses in the medium) at $t = 35$ seconds, we observe that the desired proportion is no longer maintained. On measuring the queue sizes at these instants, we observed that the packets were no longer being queued up at the software queues. This is because TCP on seeing wireless losses reduces its sending rate under high losses (i.e. after $t = 35$ seconds).

(h) Scalability and Fine-grained allocation: In this experiment we demonstrate the effectiveness of lottery scheduling in achieving a fine-grained allocation of wireless bandwidth. We set up 10 UDP flows from the AP to 2 clients (i.e. 5 flows each). We set desired proportion of throughputs to be $flow_1 : flow_2 : flow_3 .. : flow_{10}$ to be $1 : 2 : 3 .. : 10$. Table 1 shows that the flows indeed share the wireless bandwidth in the desired proportion. This shows that the solution based on lottery scales well and achieves the required fine-grained proportions.

(i) Lottery Vs. Stride: We now measure the performance of lottery scheduling against that of stride scheduling. As before we set up 2 flows with the desired ratio of throughput proportions as 1 : 2. Figure 11 shows the ratio of throughputs achieved for both the scheduling approaches. We observe that stride scheduling, being deterministic in nature, achieves better throughput proportions when compared to that achieved by

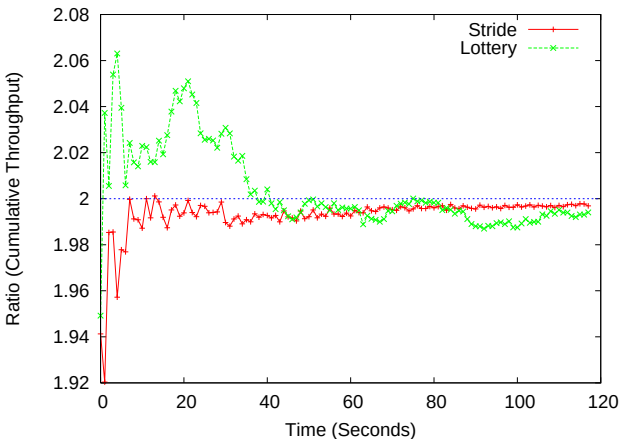

Figure 11: Performance of Lottery Scheduling and Stride Scheduling.

lottery scheduling which gives probabilistic guarantees. However, stride scheduling requires more state to be maintained per flow and involves a considerable amount of processing overhead. This is especially true when new flows are added and old flows leave the network. This constant churn requires stride to update the 'pass' assigned to each flow and result in a considerable amount of overhead when the number of flows are high. Lottery on the other requires almost no state and needs minimal processing. This however comes at the cost of reduced accuracy.

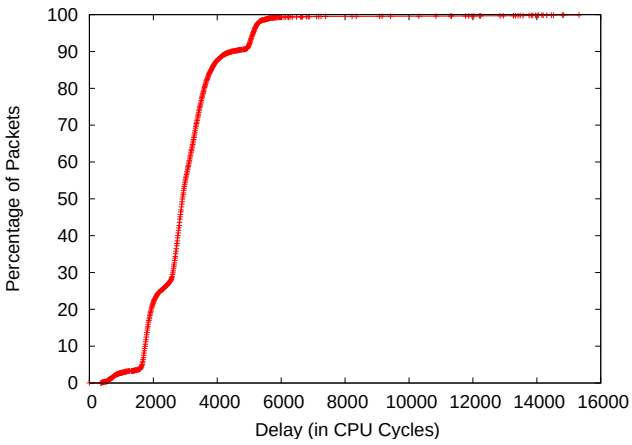

Figure 12: CDF of per-packet delay due to implementation overhead of maintaining software queues.

(j) Implementation Overhead: As explained in Section 2, we maintain per-flow software queues and forward the next packet to the firmware only after the previous packet is transmitted. This would result in delays in transmitting the next packet as it is not readily available in the firmware queue, but is buffered in the software queue and has to be forwarded to the firmware after drawing a lottery. We measure this delay over a set of $50,000$ packets which are transmitted using lottery scheduling. We measure the resulting delay using `rdtscl` in CPU cycles. Figure 12 plots the CDF of this delay in CPU cycles across the set of packets considered in our measurements. We observe that the average delay encountered is 3030 CPU cycles. On a 433 Mhz machine (Soekris net4826 [3]), this results in a delay of 7 microseconds per packet.

Summary: Our implementation of lottery scheduling works across multiple clients, different protocols and is scalable. Ticket adjustments help take into account different traffic characteristics of flows under consideration. Scheduling approach does not give desired proportions when TCP is used in presence of lossy links because of TCP congestion control mechanisms. We observed that stride scheduling achieves a better performance over lottery scheduling in terms of accuracy, but requires more state and processing at the access point. Overhead of maintaining software queues in our implementation results in a delay of about 7 microseconds per packet.

4. RELATED WORK

Packet scheduling algorithms have gained importance since the weighted fair queuing (WFQ) algorithm was proposed in [8]. Several modifications have been proposed to improve the computational complexity and performance of this algorithm. Self clocked fair queuing (SCFQ) [9] algorithm and WF^2Q [6] for example have been significant improvements. However though these algorithms have extensively been researched in the wireline domain, wireless fair scheduling is an uncharted territory.

Recently, there has been work which focuses towards providing performance guarantees in the presence of channel contention in wireless [7, 10]. The underlying idea is to combine the best features of CSMA which is a contention based scheme and contention free schemes like TDMA. There has been work on scheduling protocols that take priorities into account, when performing medium access control in wireless [5, 14, 15, 12]. For instance, Aad and Castellucia [5] present service differentiation mechanisms for wireless networks. Their mechanisms allow a host to pick a backoff interval as a function of its priority, larger backoff intervals used for lower priority. However it involves changing the wireless MAC. We tend to achieve the same results using a probabilistic approach without changing the MAC. In [4], the authors propose a MAC protocol for wireless networks to support prioritized scheduling along with a weighted fair sharing of the bandwidth amongst the

users belonging to the same priority levels. There has also been some interesting work for scheduling on a real time traffic in a wireless LAN [15]. This work, however assumes that a flow transmits packets with a constant rate. Such assumption cannot be made when performing proportional fair scheduling.

IEEE 802.11e is a proposed enhancement to the 802.11a and 802.11b wireless LAN (WLAN) specifications. It offers quality of service(QoS) features, including the prioritization of data, voice, and video transmissions.802.11e also enhances the 802.11 Media Access Control layer (MAC layer) with a coordinated time division multiple access (TDMA) construct, and adds error-correcting mechanisms for delay-sensitive applications such as voice and video. Classification of the whole traffic into four levels however provides a very coarse level control.

5. CONCLUSIONS

We have shown that Lottery Scheduling can be used to achieve a flexible and fine-grained bandwidth allocation in a wireless LAN. The approach is simple, stateless and is scalable. It works across multiple clients and different protocols. The concept of ticket adjustments help take into account different traffic characteristics of flows under consideration (packet sizes, bursty nature etc) and help achieve the desired throughput proportions. Better performance in terms of accuracy can be achieved using stride scheduling, however, it requires more state and processing at the access point. Further, we observed that when used with TCP in presence of lossy links, the desired proportion may be difficult to achieve because of the TCP congestion control mechanisms. Overall, it is interesting to note that lottery scheduling although being very simple achieves proportional-share bandwidth allocations in a wireless network with minimal overhead at the access points.

6. REFERENCES

[1] IEEE 802.11 wireless lan medium access control (mac) and physical layer (phy) specifications: Amendment 8: Medium access control (mac) quality of service enhancements. `http://standards.ieee.org/getieee802/download/802.11e-2005.pdf`.

[2] Madwifi wireless driver. `http://www.madwifi.org`.

[3] Soekris engineering. `http://www.soekris.com/net4826.htm`.

[4] P. B. A. Dugar, N. Vaidya. Priority and fair scheduling in a wireless lan. In *nternational. Conference. on Mobile. Computing. and Networking*, 2000.

[5] I. Aad and C. Castellucia. Differentiation mechanism for ieee 802.11. In *IEEE INFOCOM*, 2001.

[6] J. C. R. Bennett and H. Zhang. Wf2 q: Worst-case fair weighted fair queueing. In *INFOCOM*, 1996.

[7] k. C. C. Chang, J. Chang and M. You. Gauranteed quality-of-service wireless access to atm. In *preprint*, 1996.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM*, 1989.

[9] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *IEEE INFOCOM*, 1994.

[10] A. Muir and J. Garcia-Luna-Aceves. Supporting real-time multimedia traffic in a wireless lan. In *Proc. SPIE*, 1997.

[11] P. Ramanathan and P. Agarwal. Adapting packet fair queueing algorithms to wireless networks. In *International Conference on Mobile Computing and Networking*, 1998.

[12] K. Ramanritham and W. Zhao. Virtual time csma protocols for hard real-time communications. In *IEEE trans. Software eng.*, 1987.

[13] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quaity of service guarantees via service curves. In *ICCCN*, 1995.

[14] S. Sharrock and D. Du. Efficient csma/cd based protocols for multiple priority classes. In *IEEE Trans. Computers*, 1989.

[15] J. Sobrinho and A. KrishnaKumar. Real-time traffic over the ieee 802.11 medium access control layer. In *Bell Labs Technical J.*, 1996.

[16] C. A. Waldspurger and W. E. Wiehl. Lottery scheduling: Flexible proportional-share resource management. In *OSDI*, 1994.

[17] C. A. Waldspurger and W. E. Wiehl. Stride scheduling: Deterministic proportional-share resource management. In *MIT/LCS/TM-528*, 1995.