Achieving Relative Differentiated Services Using Proportional Probabilistic Priority Scheduling on Network Processor

Chee-Wei Tan and Chen-Khong Tham

Department of Electrical and Computer Engineering,
National University of Singapore, Singapore 119260
cheewei@alumni.nus.edu.sg, eletck@nus.edu.sg

Abstract. This paper studies the design and performance of the Probabilistic Priority (PP) packet scheduling algorithm to schedule packets. Unlike an earlier design that uses fractional arithmetic and prohibits large number of classes, we present an integer PP algorithm and show that PP is a special scheme of applying lottery scheduling to bandwidth allocation in a strict priority sense. We then propose a Multi-winner PP (MPP) scheduler using multi-winner lottery scheduling to improve the throughput and response time accuracy and a flexible ticket transfer algorithm to improve the deadline violation probability in probabilistic scheduling. Finally, we investigate the issue of parameter assignment for an MPP scheduler and use our techniques to implement a prototype Assured Forwarding (AF) mechanism in a network processor.

1 Introduction

In the Differentiated Service (DiffServ) architecture, individual flows with similar Quality-of-Service (QoS) requirements are aggregated, and given the same treatment as described by a Per-Hop-Behavior (PHB) in terms of QoS metrics such as average packet delay, packet loss and jitter. The routers do not keep per-flow states and there are no complex resource signaling mechanism involved [1]. The Assured Forwarding (AF) PHB guarantees only that the assured traffic is delivered with a higher probability than the best-effort traffic; in the case of severe network congestion, the assured traffic can still experience severe losses and high delay. This paper analyzes the Probabilistic Priority (PP) scheduling discipline within the framework of relative service differentiation. PP adopts a probabilistic relative service model. At every service round, each class takes a bid. Since higher priority classes have higher probabilities associated with them, in the long run, they will be served more often than lower priority classes. As compared to Strict Priority (SP), this increases fairness among classes and prevent the starvation of lower priority classes. We first show that PP is a cross application of lottery scheduling in a strict priority sense to provide proportional bandwidth sharing among classes. This in turn allows us to benefit from numerous techniques presented in [7,8] to control PP. The lottery and stride

A. Marshall and N. Agoulmine: MMNS 2003, LNCS 2839, pp. 300–312, 2003.

scheduling algorithms are very well-known scheduler for statistical allocation of CPU resources [7,8]. Lottery scheduling randomizes resource allocation among clients whose shares of resources are represented by tickets using policies such as ticket inflation and deflation. An allocation is performed by holding a lottery, and the resource is granted to the client with the winning ticket. Multi-winner lottery scheduling is a variant of lottery scheduling that produces better throughput accuracy for many workloads. Based on this multi-winner concept, we formulate a multi-winner PP algorithm to improve the response-time variability of PP. As lottery scheduling is effectively stateless, a great deal of complexity is removed in comparison to other proportional schedulers. The feasibility of using lottery scheduling in packet forwarding has been analyzed in [2,4,9] but no work has been done to address its weaknesses at the packet level due to its probabilistic nature. The probabilistic relative service model is only suitable for applications that are able to tolerate deadline violations of a few packets. We propose a technique that is analogous to the idea of dynamically-controlled ticket transfer which has been applied to graphics rendering and Monte-Carlo tasks [8] to address this problem.

The rest of the paper is organized as follows. We propose an efficient integer PP algorithm in section 2 and show that PP is indeed a cross application of lottery scheduling. We use the multi-winner concept to generalize PP to improve its throughput accuracy and reduce its response-time variation. We present a technique based on flexible ticket transfer to reduce the deadline violation probability in times of congestion. In Section 3, we investigate parameter assignment and propose a framework to implement Assured Forwarding. Our implementation is described in Section 4. A performance study on a network processor-based router is presented in Section 5. We conclude in Section 6.

2 Probabilistic Priority Scheduler

2.1 Basic PP Integer Algorithm

The work conserving PP Scheduler is based on the SP scheduler with each queue being assigned a probability p_i [4]. By appropriate setting of a parameter $p_i \in [0,1]$, $i = 1, \ldots N-1$ and $p_N = 1$ in a multi-class system, a class is selected with a probability corresponding to equation (1) for service at every cycle. A class parameter of $p_i = 1$ means that the class i definitely gets served when polled if all higher priority classes are empty or not selected during the cycle. Hence PP reduces to SP when $p_i = 1.0$, $i = 1, \ldots N$. Here, we derive an integer algorithm and show that it is indeed a cross application of lottery scheduling in the strict priority sense.

First, consider a multi-class system of N priority levels with the highest priority level denoted by 1. Let us define the weight of class i to share the server [4] as

$$r_i = p_i \prod_{j=1}^{i-1} (1 - p_j) \tag{1}$$

Without loss of generality, assume that all classes in the system are busy so that the normalized weight of class i among all classes is

$$\hat{r}_{i\in\Omega} = \frac{r_i}{\sum_{j\in\Omega} r_j} \tag{2}$$

where Ω consists of all queues, i.e. $\{1, \ldots, N\}$. After rearranging all r_i such that they share a common denominator of lowest common multiple, we have

$$\hat{r}_{i\in\Omega} = \frac{x_i}{\sum_{j\in\Omega} x_j} \tag{3}$$

where x_j is the numerator of the normalized relative weight $\hat{r}_i$. It is easy to see that this will also be true for all network conditions:

$$\hat{r}_{i\in BQ} = \frac{x_i}{\sum_{j\in BQ} x_j} \quad , BQ \in \Omega \tag{4}$$

where BQ is the set of non-empty queues in Ω. For example, $BQ = \{1, 2\}$ which denotes non-empty queue 1 and 2 can be found in Ω. The total number of possible network conditions, i.e. the permutation of non-empty queues in Ω, is equal to $2^N - 1$ but the most interesting set would be the total number of possible network conditions with more than one non-empty queue which is equal to $M = \sum_{i=1}^{N-1} \sum_{j=1}^{N-i} \binom{N-i}{j} = 2^N - N - 1$. We now have numerator x_i to calculate $\hat{r}_i$ without having to store in advance $\hat{r}_i$ for all possible combinations of empty and non-empty queues with each combination corresponding to a particular instance of Ω. This effectively removes both the need for fractional arithmetic in recalculation of network states whenever p_i changes dynamically and the restriction for a small set of all possible network states. The integer algorithm of PP works *without* the need for *a priori* network state computation. PP is analogous to having sets of different numbers of tickets that are present in a service round with each set corresponding to one of the network conditions in M.

2.2 Multi-winner PP (MPP) Integer Algorithm

Multi-winner lottery scheduling is a generalization of the basic lottery scheduling technique that produces better throughput accuracy and smaller response-time variation [8]. Instead of selecting a winner per round, N_w winners are selected with only the first winner being randomly selected and each winner is guaranteed the use of the resource for one quantum. The set of N_w consecutive quanta allocated by a single multi-winner lottery is referred to as a super-quantum. Due to the probabilistic nature of PP, the highest priority class can exhibit substantial variability over small time scales which can cause its HOL packet to miss its deadline if sufficient numbers of service round are given to its lower priority classes instead. At worst, this may cause buffer overflow and incoming high priority packets to be dropped. This necessitates incorporating a deterministic

mechanism in PP to achieve predictable behavior at small time scales. We use the multi-winner concept to extend the original PP integer algorithm. In this paper, we use a fixed value of $N_w = 20$. The ordering of the winners in MPP is based on a fixed permutation that goes in a round robin fashion, starting from the first winner and followed by its immediate lower priority class. This integer algorithm requires a total of $2^N - 1$ uniform distributions of integer random numbers for N classes. This is analogous to the total number of tickets differing in every service round of lottery scheduling. Waldspurger *et al.* [7] provides a multiplicative linear congruential Park-Miller pseudo random number generator in MIPS assembly language code but we use a generic algorithm *U-map* described in section 4 to scale uniform distributions without using multiplication assembly language instructions. In our algorithm, each super-quantum is reset back to 0 when the network condition changes which would happen very often if the system is highly loaded. This implies that MPP is able to reduce the throughput error and response-time variability. Through extensive simulations under heavy load conditions, we observe that the super-quantum is reset on an average of about 85% of the total time. Hence N_w does not have a significant impact on the reduction rate of throughput error. The advantage of MPP over PP appears to be small for 8 classes but by keeping the number of classes small, we can increase the number of winners to provide stricter throughput guarantees within a class.

2.3 Flexible Ticket Transfer Algorithm

In the previous section, we described an extension of PP to achieve throughput guarantee. In this section, we aim to reduce the time given up to the lower priority classes by the higher priority classes ("slack" in probabilistic scheduling) by setting a rate of approaching strict prioritization using the relationship between delays of different classes. In particular, we use the following propositions of average delay of class i, $\overline{W_i}$ proven in [6] to affect p_i.
(1) As $p_j \uparrow [0 \rightarrow 1]$[1] for $j < i$, $\overline{W_i}$ is continuously and monotonically increasing.
(2) As $p_i \uparrow [0 \rightarrow 1]$, $\overline{W_i}$ is continuously and monotonically decreasing.
(3) As $p_j \uparrow [0 \rightarrow 1]$ for $j > i$, $\overline{W_i}$ is nearly constant under congested network conditions.
Let us define the initial parameter r_i for class i that satisfies the relationship $r_1 \geq r_2 \cdots \geq r_i \geq \cdots \geq r_N$ for the multi-class system where Class 1 is the highest priority class. Such assignment means that the probability of higher priority class is larger. This algorithm consists of the following two steps. The first step is to reduce the probability of a lower priority class after it has been served by transferring some probability to its immediate higher priority class. Note that the transfer of tickets from the class served to its immediate higher priority class will create a snowball effect that will cause the highest priority class to be eventually served while still using probabilistic scheduling. The second step is to preserve as much as possible the priority allocation that is defined at the start of

[1] Following [6], the notation "$x \uparrow [0 \rightarrow 1]$" means "$x$ increases from 0 to 1".

the algorithm by transferring probability starting from the lowest priority class even though it has not been served to the immediate higher priority class of the class being served if the first step persists. Eventually the class that continuously gets served will lose its bid after the probabilities of all lower priority classes have been depleted.

Table 1. Outline of ticket transfer algorithm

At each service round, suppose classes 1 to L, corresponding to a particular network condition $BQ \in M = 2^N - N - 1$ where N is the total number of classes, are busy,
1. If class i, $1 < i \leq L$, gets served, then $r_i^{'} = max\,(r_i - \triangle_i, r_{i+1})$, and $r_{i-1}^{'} = min\,(r_{i-1} + \triangle_i, 1.0)$, such that $r_i^{'} \geq r_{i+1}$, i.e. transfer $\triangle_i$ of probability being served to the immediate next higher priority level with $r_i \neq 0$.
2. If $r_i = r_{i+1}$, then $r_k^{'} = (r_k - \triangle_k)^+$, $i < k \leq L$ where k is the lowest priority class in BQ that satisfies $r_k \neq 0$, and $r_{i-1}^{'} = min\,(r_{i-1} + \triangle_k, 1.0)$, i.e. transfer $\triangle_k$ of probability being served to the immediate next higher priority class $i-1$.
3. If the highest priority class is served or the network condition BQ changes, $r_i^{'} = r_i$, i.e. reset all class parameters back to their original r_i.

From the algorithm shown in Table 1 and equation (1), we can make the following propositions:
(a) If $p_{i+1} < \frac{p_i}{1-p_i} \leq 1$ and $\triangle_i$ of probability to be served is transferred from class i to class $i-1$, $\hat{p}_i$ decreases, $\hat{p}_{i-1}$ increases, and $\hat{p}_j$, $j \neq i, i-1$ remains constant.
(b) If $p_{i+1} = \frac{p_i}{1-p_i} \leq 1$, and $\triangle_k$ of probability to be served is transferred from class k, $i < k \leq L$ to class $i-1$, $\hat{p_j} \uparrow \left[p_j^{orig} \rightarrow 1\right]$, $j \leq i$ where p_j^{orig} is the original PP parameter of class j.
Proposition (a) states that only the probabilities of the class served and its immediate higher priority class will change while the other classes will maintain the original PP configurations at the initial stages after the algorithm begins while proposition (b) states that higher class priority will approach the configuration of SP, i.e. $\hat{p}_j \rightarrow 1$, $\hat{p}_j \neq 0$, $1 < j \leq i$ if the situation where the highest priority class HOL packet is not served while class i is constantly being served persists. Therefore, from proposition (1) and (2), the average delays of classes with higher priorities than class i will decrease monotonically over time while those classes with lower priorities than class i will increase monotonically over time. We introduce an additional parameter $\triangle_i$ to provide a dynamic feed-forward mechanism based on the current workload or the slack of the corresponding high priority HOL packet. This user-tunable class parameter $\triangle_i$ can be a function of the class's burstiness or the higher priority classes' backlog. It provides a way for the original PP to approach SP in a configurable length of time so that the HOL packet of higher priority classes will not exceed its deadline unnecessarily.

2.4 Simulation Studies

In this section, we consider scenarios with high traffic loads and tight deadlines for each class. For each class, we use Long range dependent (LRD) traffic modeled as Pareto On-off processes with shape parameter 1.3 since aggregated traffic in real DiffServ networks is LRD in nature. The mean service time is taken to be the unit of time and the service times of packets in each class follow the same exponential distribution with mean 1.0 units. Results are averaged over 10^6 time unit simulation windows unless otherwise indicated. Throughout this paper, we use λ_i and ρ_i to denote the arrival rate and traffic intensity of class i respectively. In Table 2, the arrival rates for all classes are the same, i.e. $\rho_i = 0.125$ so the system is not overloaded, i.e. $\rho = 1.0$. Each class has the same parameter i.e. $p_i = 0.6$, $i \neq N$. To compare the performance between the various schemes, we use deadline violation probability in Table 2 as a performance metric. The deadlines for class 1 to N, where $N = 8$, are arbitrary selected as 11, 16.5, 22, 27.5, 33, 38.5, 44, and 49.5 time units respectively. The probability transfer quantum is the same for all classes, i.e. $\triangle_i = min\,(0.15, r_i)$.

Table 2. Comparison of (a)deadline violation probabilities (%) and (b)average delay (time units) under full utilization condition

	PP/Lottery	MPP	MPP w/ ticket xfer	SP
Class 1	0.114	0.069	0.036	0.000
Class 2	0.172	0.082	0.068	0.010
Class 3	4.297	1.680	0.646	0.410
Class 4	23.513	17.883	11.451	9.647
Class 5	34.696	27.678	21.410	14.609
Class 6	57.679	45.020	35.453	27.650
Class 7	91.627	88.020	64.952	58.243
Class 8	94.831	90.671	67.316	100.000

	PP/Lottery	MPP	MPP w/ ticket xfer
Class 1	1.350	1.170	1.201
Class 2	1.990	1.460	1.450
Class 3	4.920	3.550	3.471
Class 4	55.290	45.640	37.400
Class 5	198.490	214.620	120.810
Class 6	555.440	402.180	240.080
Class 7	6719.060	3726.320	1973.990
Class 8	22243.940	19847.430	7240.370

Results in Table 2 indicates that ticket transfer algorithm does not have an adverse effect on low priority class though it discriminates against them by allowing high priority classes to be selected as fast as possible. In addition, it

also suggests that this mechanism improves deadline violation probability of low priority classes as opposed to intuition which we investigate next.

We now consider the ticket transfer algorithm used in a 4-class system to evaluate its effectiveness. Each class has parameter $p_1 = 0.5, p_2 = 0.55, p_3 = 0.6$ and $p_4 = 1.0$. Note this parameter assignment provides lower priority classes with higher probabilities of being serviced than in previous simulations. Fig. 1 shows the probabilities of all possible network conditions occurring in the system for SP, PP and MPP with ticket transfer schedulers at both short (10^3 time units) and long timescales (10^6 time units) with respect to packet service times under different loads. Each network condition is binary-coded as follows: bit 0 corresponds to the highest priority class, class 1 hence 0101B implies that only class 1 and 3 are present. Note that the network condition is a function of offered loads and scheduling mechanism. We also compare the Pareto on-off traffic model with the token bucket-constrained traffic model with a bucket depth of 17 time units which exhibits short bursts.

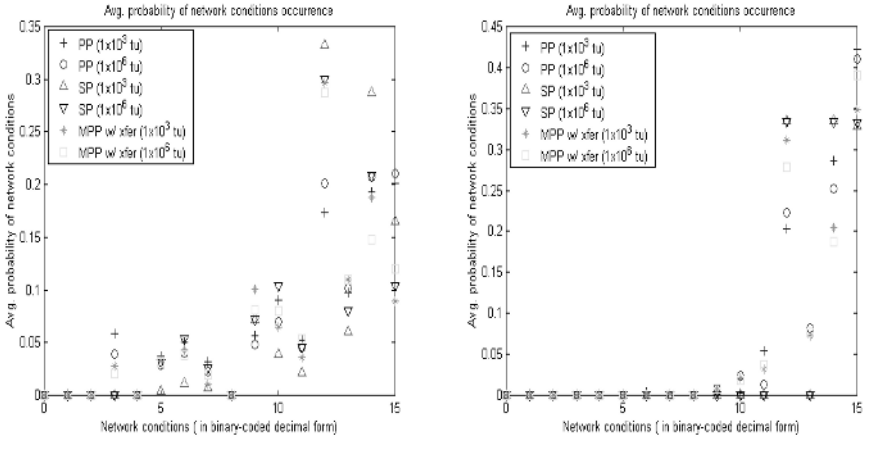

Fig. 1. (a) Comparison of network condition probabilities between PP, SP and MPP with ticket transfer scheme under light load (b) Comparison of network condition probabilities between PP, SP and MPP with ticket transfer scheme under heavy load

Note that, in contrast to intuition, the deadline violation probability of the lowest priority class is improved significantly when the ticket transfer algorithm is used because higher priority classes are assured to get transmitted within short timescale and this implies that the probability of network conditions containing these high priority classes occurring within a longer time frame will be smaller than that in comparison to normal PP scheduling. From Fig. 1 and Fig. 2, we can make the following observations:

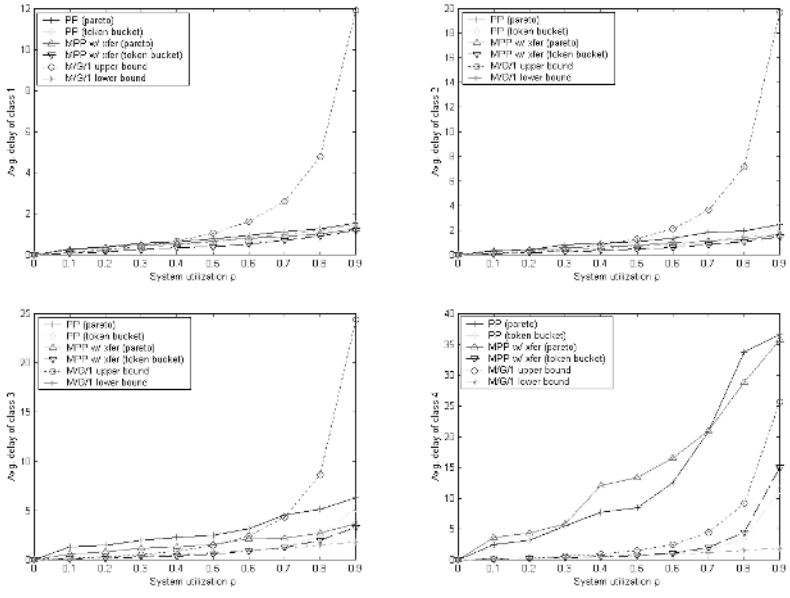

Fig. 2. Average queueing delay under different traffic loads using Pareto on-off and token bucket filter constrained traffic

– We found that MPP with ticket transfer can always achieve smaller average delay and deadline violation probability than PP and MPP scheme for most classes. Its deadline violation probability of the lowest priority class can be better than SP.

– Generally the delay of token bucket-constrained traffic lies in between the $M/G/1$ delay bounds derived in [6]. But the heavy-tailedness of Pareto on-off, for eg. with a shape parameter of 1.3, and burst rate 0.25 can cause the delay to exceed the $M/G/1$ delay bound.

– The ticket transfer algorithm has an evident impact on reducing the mean delay of all classes except the lowest priority class. This is due to: (a) the probability of the network condition 12 (1100B) that contains only the two lowest priority classes becomes higher, and (b) the probability of the network condition 15 (1111B) that contains all classes becomes smaller, and in both cases, they approach that of SP. Both (a) and (b) increase the probability of the lower classes being serviced. Since the algorithm differentiates that higher priority classes are served as fast as possible when network conditions containing them appear, the mean delays of higher priority classes will therefore be much smaller than PP.

3 Achieving Assured Forwarding Using MPP

We consider 8 QoS classes and we configure a MPP scheduler to have 2 segregation groups AF_1 and AF_2. Each group has the last parameter $p_4^{AF_1} = p_4^{AF_2} = 1$. In each group, the AF classes are assigned parameters $p_1^{AF_i} < p_2^{AF_i} < \cdots < p_4^{AF_i}$, $i = 1, 2$. The following theorem ensures that this parameter assignment guarantees AF classes to obtain better statistical relative delay service differentiation than its immediate lower priority class. The group segregation property states that in PP, the service discipline among segregation groups is exactly the Strict Priority discipline hence the first AF group is guaranteed to have better service than the second group in terms of delay [4]. By means of segregation, this framework (a) provides more isolation among high priority classes that demand low delay and deadline violation probability, and low priority classes that require at least best effort service, and (b) reduces the number of classes within a group as this means a smaller number of network conditions within each group therefore we can configure more number of winners within each super-quantum, i.e. smaller spacing between consecutive winners to improve the response time variability in multi-winner scheduling. Since each group is based on MPP scheduling, there is fairness in the resource allocation within each group by means of fair distribution to excess capacity [4]. The ticket transfer algorithm is used in the first segregation group to provide improved deadline violation probability and average delay. Since we do not consider admission control, we expect some form of policing to limit the burst size and amount of bandwidth admitted to each class to prevent starvation if a non-conforming flow enters the node.

Theorem 1. *An assignment of average probability parameter for each class where* $0 < p_1 < p_2 \cdots < p_N = 1$ *satisfies the priority hierarchy for relative service differentiation.*

Proof. Define r_i as in equation (1) and q_i as the average queue length of class i. At steady state, we want higher priority classes to have shorter backlogs. Hence using the relationship that $\frac{r_i}{r_1} \propto \frac{q_i}{q_1}$, we can use Little's theorem [5] to show that $p_i = \frac{\lambda_i}{\prod_{k=1}^{i-1}(1-p_k)\sum_{j=1}^{N}\lambda_j}$, $i = 1, \ldots, N$. Since $p_{i-1} = \frac{\lambda_{i-1}}{\prod_{k=1}^{i-2}(1-p_k)\sum_{j=1}^{N}\lambda_j}$ therefore $\frac{p_{i-1}}{p_i} = \frac{\lambda_{i-1}}{\lambda_i}(1 - p_{i-1})$ which can be further simplified to $p_{i-1}\lambda_i\left(\frac{1}{p_i} + \frac{\lambda_{i-1}}{\lambda_i}\right) = \lambda_{i-1}$. Since the highest priority class gets served with the highest probability, its average departure rate must be the greatest among all classes, i.e. $\lambda_1 > \lambda_2 \cdots > \lambda_N > 0$. Thus we have $p_{i-1}\left(\frac{1}{p_i} + \frac{\lambda_{i-1}}{\lambda_i}\right) > 1$. Rearranging the term leads to $\frac{p_i p_{i-1}}{p_i - p_{i-1}} > \frac{\lambda_i}{\lambda_{i-1}}$ hence $p_{i-1} < p_i$. Thus the theorem is implied. Since this assignment is independent of the number of classes in the system, $p_1 > \frac{1}{2}$.

4 Efficient Implementation of MPP in IXP1200

We implemented our proportional bandwidth guaranteed probabilistic priority multi-class framework proposed in the previous section on Intel IXP1200 network

processor [3][2]. To convert the class parameter p_i to tickets in lottery scheduling, all the assigned parameters p_i within a group are normalized to their least common multiple. For a large number of classes, we use Euclid's Greatest Common Divisor algorithm to speed up computation in the StrongARM core before supplying the scheduler's parameters in a numerator vector string to the microengines. For an 8-class system at an egress port, all the class parameters are stored in only two SRAM memory words with each parameter p_i occupying 8 bits thus the smallest probability being addressable is $\frac{1}{256}$ which offers relatively high computational granularity. In comparison, earlier implementation [6] will require over 100 Bytes of parameters' storage and larger memory access overheads. Clearly, our approach reduces memory access overhead drastically and accommodates more classes in a multi-port setting.

4.1 Fast Algorithm for Scaling Uniform Distribution

The StrongARM is elected to run a periodic task of generating uniform pseudo-random numbers in the SRAM. When the microengines require a random number for computation, they simply do a table lookup. This table has to be updated often by StrongARM to prevent a microengine from reading the same entry twice. However, we note that too high a refreshing frequency will lead to a higher latency for a microengine's SRAM read operation to this shared table due to increased contention between StrongARM and microengines. Numerous techniques exist for scaling uniform random numbers. An exact scaling method would convert the random number from an integer to a floating-point number between 0 and 1, multiply it by X, and then convert the result back to the nearest integer [8]. Alternatively, 32-bit random numbers in a particular uniform distribution, $Uniform[0, X]$ can be obtained by dividing any random 32-bit wide number in the range 0 to $2^{32} - 1$ by X and keeping the remainder under the assumption that $X \ll 2^{32} - 1$ [8]. Due to the significant computation overhead of integer division (measured as 378 cycles and independent of the value size of X), this method is not scalable without a pseudo random number generation co-processor. From the observation that each bit in any 32-bit uniformly distributed random number has an equal chance of being a "1" or "0", we use a simple generic bit-wise algorithm to map this uniform random number into another equally uniform random number, effectively scaling $Uniform\left[0, 2^{32} - 1\right]$ to $Uniform\left[0, X\right]$. This algorithm shown in Table 3 first performs the AND operation, and then re-claims those bits lost in the AND operation, ignoring bits which are outside the desired range. It is noteworthy that the instruction cycle count for this algorithm depends on the value size of X, i.e. we can trade-off computational granularity with speed. For X less than 255, this algorithm takes 41 instruction cycle counts. In the worst case, mapping a full 32-bit value of X requires a maximum of 173 instruction cycle counts but the gain is already an exponential increase in computational granularity to approximately $2^{32} - 1$.

[2] We use Intel network processor IXM1200 c-PCI hardware based on IXP1240 chipset.

Table 3. *U-Map* scaling algorithm

```
    int result         =    0;
            int comparator    =     denominator & random_number;
if(comparator == 0) comparator = denominator;
while(comparator)
            {
                result      | =    (comparator & random_number);
                comparator  >>=   1;
            }
            if(result > denominator)    result =    denominator ^ result;
            return  result;
```

5 Performance Study and Results

In this section, we evaluate the performance of our implementation. In our experiments, we use token bucket metering to characterize the service and allocate a pre-calculated buffer space for each class. We present here the results in terms of mean delay and deadline violation probability. The topology of the experimental test-bed is shown in Fig. 3. All network links are full-duplex and have a capacity of 100 Mbps. We classify the traffic generated as Assured Forwarding (AF) and Best-Effort (BE). We implement 8 QoS classes with DiffServ Codepoints (DSCP) classification using our framework with two segregation groups. Each priority class in AF has marking 0x2e, 0x0a, 0x12, 0x1a, 0x22, 0x0c, 0x14, and 0 respectively. Class 1 and 2 traffic is sent from Sender 1 with the rest of the traffic in Class 3 to 8 from Sender 2. All flows are independent Poisson processes with exponentially distributed packet lengths and have the same mean sending rate and mean packet size. In order to simulate congestion, we use one IXP1200 (IXP Router 2) to generate high volume of traffic at the Gigabit output which is in turn forwarded to the Fast Ethernet output port on the other IXP1200 (IXP Router 1) which runs the MPP scheduler algorithm. Additional cross-traffic is also generated in the background to vary the congestion load pattern. All traffic terminates at Receiver.

The parameters for the framework are as shown in Fig. 3.The deadline violation probabilities of class 1 and class 2 are shown in Fig. 4(a). As expected, the deadline violation probabilities of class 1 and class 2 of MPP with ticket transfer scheme lie in between that of normal PP and SP. At low load, the deadline violation probability is very close to that of SP. Fig. 4(b) shows the average delay ratio between classes of the MPP with ticket transfer scheme measured within an interval of 1 hour. As the traffic load increases, the delay differences between classes become wider. Thus, with appropriating setting of the class parameter as described in section 4, higher priority classes get better delay differentiation at medium to high load. Due to space limitation, we do not present the packet loss statistics but we observe that the packet loss for each class in our experiments is strictly increasing as the priorities get lower. Note in Fig. 4(b) that the delay

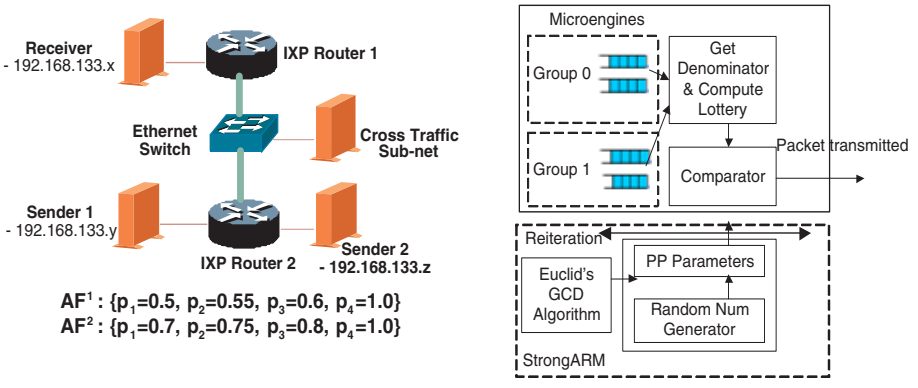

Fig. 3. (a)Relative DiffServ Test-bed and Assured Forwarding framework configuration (b) Block diagram of implementation on IXP1200 network processor

spacing between the last class in AF_1 and the first class in AF_2 is quite small. However, MPP scheduler observes the strict priority rule between segregation groups hence we can expect packet loss and deadline violation probability of the first class in AF_2 to be higher.

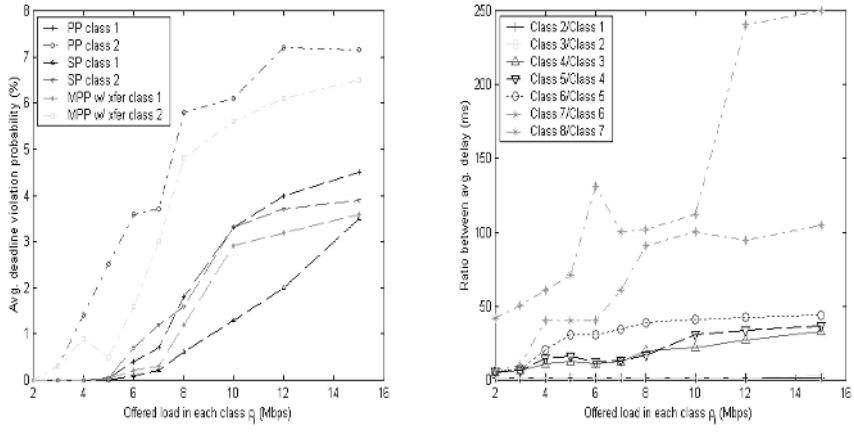

Fig. 4. (a)Deadline violation probabilities (b)Delay ratios between classes

In order to compare the impact of packet sizes on the performance of the MPP with ticket transfer scheme with PP scheme under congested conditions, we repeated the same experiments with different packet size distributions. The observed experimental results were largely similar to those obtained above but

we note that for large packet size close to MTU, the benefit of the ticket transfer algorithm is not so obvious because, at high load, the time for a single packet transmission becomes longer thereby increasing the probability of the network condition where all classes' HOL packets are present as is in the case of SP. Nevertheless, the queuing delays in this case are still not as high as those for SP or the PP scheme. In summary, the MPP with ticket transfer scheme is good when the primary goal is to provide relative delay differentiation as in SP while ensuring that deadlines of higher priority classes are not unnecessarily violated, and also meeting specific timing requirements, for eg. small delay bounds for high priority classes as in absolute QoS.

6 Conclusion

This paper showed that PP is a special scheme of lottery scheduling in the strict priority sense and the PP algorithm was generalized to a Multi-winner PP algorithm which ensures that high priority classes get served within deterministic time quanta. A ticket transfer algorithm was proposed to overcome the problem of the highest priority class from missing its deadline at small time scales. Simulations showed that MPP with ticket transfer surpasses the original PP and SP using bursty traffic class aggregation. Our algorithm provides lower deadline violation probability and mean delay to most classes than original PP. Finally, we presented the performance of our algorithms on high-speed routers.

References

1. S. Blake, D. Black, M. Carlson, E. Davis, Z. Wang and W. Weiss, An architecture for differentiated services, IETF RFC 2475, Dec 1998.
2. J. Eggleston and S. Jamin, Differentiated services with lottery scheduling, *Proc. Int'l Workshop on Quality of Service (IWQoS'01)*, Jun 2001.
3. Intel IXP 1200 Network Processor: Microcode Programmer's Reference Manual Revision 11, Part No. 278304-011 March 2002.
4. Y. Jiang, C.K. Tham and C.C. Ko, A probabilistic priority scheduling discipline for multi-service networks, *Computer Communications 25 (13) 2002 pp. 1243–1254*, 2002.
5. L. Kleinrock, Queueing Systems: Vol.2, Computer applications, John Wiley & Sons, 1976.
6. C.-K. Tham, Q. Yao and Y. Jiang, Achieving differentiated services through multi-class probabilistic priority scheduling, *Computer Networks, 40, pp. 577–593*, 2002.
7. C. Waldspurger and W. Weihl, Lottery scheduling: Flexible proportional-share resource management, *Proc. the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Nov 1994.
8. C. Waldspurger, Lottery and stride scheduling: Flexible proportional-share resource management, Ph.D. Thesis, Massachusetts Institute of Technology, Sep 1995.
9. M. Zhang, R. Wang, L. Peterson and A. Krishnamurthy, Probabilistic packet scheduling: Achieving proportional share bandwidth allocation for TCP flows, *Proc. IEEE INFOCOM 2002*, Jun 2002.