Randomized Batch Scheduling with Minimum Configurations for Switches and Routers

Zhen Zhou and Mounir Hamdi
Department of Computer Science
Hong Kong University of Science & Technology
Clear Water Bay, Hong Kong
Email: {cszz, hamdi}@cs.ust.hk

Abstract— **As the technology advances, the speed and sizes of input-queued internet switches increase dramatically. The design of the switch scheduler becomes a primary challenge, because the time interval for making scheduling decisions becomes very small. One method to resolve this problem is to reduce the scheduling frequency for a batch of packets and pipeline the switching tasks. Such method is called *batch scheduling*.**

Since computing and setting up each switch configuration incurs a notable cost, it is preferred to have minimum number of configurations for every batch. We study in this paper approximation schemes for batch scheduling with minimum configurations. We propose three algorithms together with their (expected) approximation ratios. The NAIVE ROUND ROBIN **algorithm runs in $O(N^2)$ time and has the worst possible approximation ratio. We try to improve the approximation ratio by randomization. The** RANDOMIZED ROUND ROBIN **algorithm has an expected approximation ratio of $O(\ln N)$. In the same fashion, we propose the** BARELY RANDOM ROUND ROBIN **algorithm that uses less random bits at a cost of worse approximation ratio. Finally, our simulation results indicate that a fabric speedup of 2 is sufficient for our batch scheduling algorithms to provide quality of service (QoS).**

I. INTRODUCTION

Today's internet relies very much on routing and switching technologies. High-speed core routers use virtual output queues for storing fixed size packets, and a crossbar fabric or an optical fabric for switching packets. At each time slot, a centralized scheduler finds a matching between the inputs and outputs. Then the fabric is configured according to the matching in order to dispatch packets from the inputs to the outputs. Researchers studied extensively efficient algorithms for finding this matching. However, the scheduling algorithms do not scale as fast as router hardware. For instance, with line rates of 40 Gbps (e.g., Cisco's OC768) and 64-byte cells, an algorithm would have to compute a matching every 12.8 ns, while the current highest-capacity commercially available centralized scheduler takes about 50 ns per matching. In optical switches, the overhead comes from not only the computation but also the reconfiguration delays [2], [3], [4]. One possible solution is *batch scheduling*, in which we reduce the rate of finding matchings and pipeline the switching tasks.

In batch scheduling, a switch works in a three-phase cycle. The switch accumulates packets in a batch in the first phase. Then it does the scheduling task in the second phase and switches the packets to output lines in the third phase. In such a manner, pipelining is allowed as shown in Fig. 1. We are guaranteed to have 100% throughput and the worst case switching delay is bounded by the sum of the time spent in the three phases, given that packets that arrive in the first phase are transmitted after the third phase. It is also critical that the third phase never takes longer than the time of the accumulating phase, because otherwise the switch is not stable.

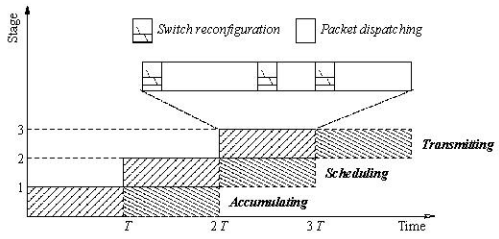

Fig. 1. Three phases of batch scheduling and pipelining.

Since each time it requires a considerable amount of time to compute matchings and reconfigure fabrics (especially in the optical case), it is always preferred to produce minimum number of configurations for each batch. In order to successfully dispatch all packets in a batch, fabric speedup is required both to cover the reconfiguration overhead and to compensate for empty slots left by the scheduling algorithm.

In this paper, we study and propose batch scheduling algorithms that produce minimum number of configurations and try to minimize the scheduling makespan at the same time. We name it MBS (Minimum Batch Scheduling) problem for ease of discussion. This problem is proved to be NP-complete by Gopal and Wong [1]. As 100% throughput and bounded delay have been ensured by pipelining method (with speedup), we may now focus on other design issues for MBS algorithms, such as time complexity and simplicity (due to the hardware limitations). Another major concern is the efficiency of MBS algorithms. Because we are minimizing the number of configurations, we are expected to see a lot of empty time slots for each matching. The smaller the makespan is, the less time slots are wasted, and the more efficient the algorithm is. We study this issue in terms of the approximation ratios, where the algorithms' makespan is compared against the optimal one.

The **main contribution** of this paper is outlined as follows: After formally defining the MBS problem in Section II,

1-4244-1206-4/07/$25.00 ©2007 IEEE

we first exploit its approximation ratio against an optimal algorithm in Section III-A. The upper bound of the approximation ratio is shown to be N, given an $N \times N$ switch. In section III-B, we propose a deterministic algorithm NAIVE ROUND ROBIN that runs in $O(N^2)$ time. It is the origin of two randomized algorithms. The first randomized algorithm RANDOMIZED ROUND ROBIN uses $N \log N$ random bits. It has expected approximation ratio of $\ln N + 1$. The second randomized algorithm BARELY RANDOM ROUND ROBIN has a worse approximation ratio by using only $\log_2 h$ random bits. Because minimum number of configurations is produced, approximation ratios for our MBS algorithms imply a speedup factor (defined in (1)) of switching fabrics. In Section IV, our simulation results show that a speedup of 2 will be sufficient for fabrics in batch scheduling to provide quality of service (QoS) when any of our algorithms is employed.

II. PRELIMINARIES

In an N-input-N-output switch, time is slotted and we assume traffic is *admissible*, so that there is at most one fixed-size packet received from any input port or dispatched to any output port in one time slot. In batch scheduling, packets are first accumulated for T time slots, where T is the fixed *batch length*. We then obtain a *traffic matrix* (i.e., *batch*)

$$D = [d_{i,j}]_{N \times N} \quad d_{i,j} \geq 0,$$

in which any row sum and column sum should not exceed the accumulated port capacity under an admissible traffic. That is,

$$\forall i, j, \quad \sum_i d_{i,j} \leq T \text{ and } \sum_j d_{i,j} \leq T, \quad (T > N).$$

Then batch scheduling algorithms are responsible to find switching configurations between inputs and outputs in order to deliver the accumulated traffic matrix to the output ports. It is not hard to see that the minimum number of configurations needed is N (which also has been formally proved in [1]). Therefore, the goal of the MBS algorithm is to minimize the *makespan* (i.e., the total scheduled time) of transmitting the traffic within exactly N configurations.

Each configuration can be represented as a *(partial) permutation matrix* $P = [p_{i,j}]_{N \times N}$ which is a 0-1 matrix with at most one "1" on each row or column. An "1" on the ith row jth column indicates that input i shall connect with output j in the current switching. An MBS algorithm produces N configurations P_k $(1 \leq k \leq N)$, each lasts for w_k time slots, that *covers* the traffic matrix. We can express our MBS problem as follows.

$$\begin{array}{ll} \text{objective:} & \min \sum_{k=1}^{N} w_k \\ \text{subject to} & D \leq w_k P_k \qquad 1 \leq k \leq N \end{array}$$

Note that it is generally difficult to determine the set $\{P_k\}$ from $\binom{N!}{N}$ candidates. Hence it is not a Linear Programming problem. Once the set $\{P_k\}$ is found, the corresponding weights are merely the largest entries covered in every configuration.

Each time the switch computes and starts up a new switching configuration, it introduces an *overhead* δ. Let N_c denotes the number of configurations for batch scheduling. Assuming that T is larger than $N_c\delta$, in order to transmit all packets in T time slots, speedup is required both to cover the overhead and to compensate for empty slots left by the scheduling algorithm. Since $N_c\delta$ time slots for computing the matching and reconfiguring the switch is substantial, only $T - N_c\delta$ time slots are available for actually transmitting packets. Therefore, the *speedup* required by the schedule is

$$S = \frac{\sum_{k=1}^{N} w_k}{T - N_c\delta}. \tag{1}$$

Our batch scheduling algorithms with minimum configuration make the denominator a fixed value, i.e., $T - N_c\delta = T - N\delta$. Hence, minimizing the speedup is the same as minimizing the makespan.

Related work. Batch scheduling with minimum configurations arises in the context of Satellite Switched Time Division Multiple Access (SS/TDMA) systems. Gopal and Wong proved its NP-completeness and introduced an $O(N^4)$ time algorithm in [1]. Towles and Dally improved the time complexity to $O(N^{3.5})$ by their algorithm MIN in [7]. Both of their algorithms are based on maximal matching subroutines, which forces the overall algorithms to run for at least $O(N^{3.5})$ time [5], [6]. Neither of them provides explicit analysis of the approximation ratios of their algorithms. An algorithm based on divide-and-conquer paradigm was introduced by Zhou, Li and Hamdi [8] that achieves asymptotically the fastest possible time complexity of $O(N^2)$. To the best of the authors' knowledge, no randomized algorithm was introduced for this problem.

III. APPROXIMATION MBS ALGORITHMS

Let $W_{\mathcal{A}}(D)$ denote the makespan incurred by a deterministic algorithm $\mathcal{A}$ scheduling the batch D. Similarly, let $W_{OPT}(D)$ denote the makespan by the optimal algorithm OPT. The approximation ratio a is defined to be the maximum ratio between the makespan of $\mathcal{A}$ and the makespan of OPT on the same batch for all batches, i.e.,

$$a \leq \frac{W_{\mathcal{A}}(D)}{W_{OPT}(D)}, \quad \forall D.$$

$E[W_{\mathcal{A}'}(D)]$ will substitute for $W_{\mathcal{A}}(D)$ in the above definition, if $\mathcal{A}'$ is a randomized algorithm, and the expectation is taken over all random choices made by $\mathcal{A}'$.

A. Upper Bound of the Approximation Ratios

We prove the following theorem on the upper bound of the approximation ratio of any MBS algorithm.

Theorem 1: The approximation ratio of any deterministic MBS algorithm is no more than N.

Proof: Given any batch $D = [d_{i,j}]_{N \times N}$, suppose any MBS algorithm $\mathcal{A}$ produces weights $w_1, w_2, \ldots, w_N$. So the makespan is $W_{\mathcal{A}}(D) = \sum_{k=1}^{N} w_k$.

Notice that w_k is actually the amount of the largest entry in configuration P_k. Therefore, $w_k = d_{i,j}$ for some i and j.

Let $D' = [w_k]_{N\times N}$ be the batch that contains only the largest entries of each configuration. The optimal off-line algorithm OPT has to at least cover D'. In the best case, OPT's total makespan for D is

$$W_{OPT}(D) \geq W_{OPT}(D') = \max_k\{w_k\}.$$

That is, OPT covers D by only one configuration with weight $\max_k\{w_k\}$ in the best case. The approximation ratio

$$a = \frac{W_{\mathcal{A}}(D)}{W_{OPT}(D)} \leq \frac{\sum_{k=1}^{N} w_k}{\max_k\{w_k\}} \leq N.$$

As this is the best case construction, we conclude that in general the approximation ratio a is at most N. ■

B. Naive Round Robin Algorithm

Before we proceed to the description of the algorithms, we shall define one helpful notation. Let $P = [p_{i,j}]_{N\times N}$ be an arbitrary permutation matrix. We define $P' = P \oplus x$ if and only if $p'_{i,j} = p_{i,(j+x) \bmod N}$ for all $p'_{i,j} \in P'$. In other words, operator $\oplus$ defines the feedback right-shift on all entries of P for x positions. We say two permutation matrices P and P' are *unrelated* if and only if $P' \neq P \oplus x$ for any x. Otherwise they are called *related*. In fact, "unrelated" (or "relate") relation is symmetric. If we cannot arrive to P' by right-shifting the entries in P, neither can we get P by right-shifting the entries in P'.

Now we can introduce a deterministic algorithm called NAIVE ROUND ROBIN (NRR), which is the origin of our randomized MBS algorithms.

Algorithm NRR(D)

Input:

$N \times N$ non-negative integer matrix D.

Output:

A set of permutation matrices $P_1, \ldots, P_N$ and corresponding non-negative integer weights $w_1, \ldots, w_N$.

Procedure:

1) Initialize P_1 to be a diagonal (permutation) matrix.
2) Set $P_k = P_1 \oplus k$, for $1 \leq k < N$.
3) Find the largest entry in D that covered by P_k and set it to be w_k, and output

---

Theorem 2: The approximation ratio of the NAIVE ROUND ROBIN algorithm is N.

Proof: We prove this ratio by constructing an adversarial traffic matrix D_{adv}. In D_{adv}, $d_{j,2j-1} = T$ for $1 \leq j \leq \frac{N+1}{2}$, $d_{\lceil N/2\rceil+j,2j} = T$ for $1 \leq j \leq N/2$, and all others entries are zero. An example 5×5 matrix is shown below.

$$D_{adv} = \begin{bmatrix} T & 0 & 0 & 0 & 0 \\ 0 & 0 & T & 0 & 0 \\ 0 & 0 & 0 & 0 & T \\ 0 & T & 0 & 0 & 0 \\ 0 & 0 & 0 & T & 0 \end{bmatrix}$$

D_{adv} is clearly admissible as its row sums and column sums are no larger than T. NRR will produce N non-zero configurations for D_{adv}. Each has weight T, while the optimal algorithm will produce only one configuration of weight T. So the approximation ratio of NRR is

$$a_{NRR} \geq \frac{W_{NRR}(D_{adv})}{W_{OPT}(D_{adv})} = NT/T = N.$$

■

Observe that the reason why NRR has a poor approximation ratio is because it uses only N possible permutation matrices. If we denote Φ be the set of all possible permutation matrices, $|\Phi| = N!$. In general, if an algorithm only utilizes a small set ϕ of different permutation matrices ($|\phi| < N!$), one can always find an adversary input that forces the algorithm to be N-approximate. For example, the adversary input could be $T \cdot P$ for any $P \notin \phi$. We actually applied this idea in the proof of NRR's approximation ratio above.

C. Randomized Round Robin Algorithm

After understanding the reason of NRR's shortcoming, we may now try to improve the performance. The idea is randomization. Instead of fixing one initial permutation matrix, we randomly choose one from the set Φ. Not surprisingly, RANDOMIZED ROUND ROBIN (RandRR) is the same as NRR except for the first step.

Algorithm RandRR(D)

1) Randomly choose P_1 from the set Φ of all possible permutation matrices.
2) Set $P_k = P_1 \oplus k$, for $1 \leq k < N$.
3) Find the largest entry in D that covered by P_k and set it to be w_k.
4) Output.

Because RandRR is a randomized algorithm, we shall examine its expected approximation ratio. In our analysis, we assume that RandRR chooses an initial permutation from Φ according to a uniform distribution. Note that it is nothing to do with the input traffic distribution.

Given any $N \times N$ batch b, denote $w_1^O(b), \ldots, w_N^O(b)$ the weights incurred by the optimal algorithm OPT; and $w_1^R(b), \ldots, w_N^R(b)$ the weights incurred by RandRR. Without loss of generality, we assume that the weights are arranged in non-increasing order for both OPT and RandRR. That is, $w_1^O \geq w_2^O \geq \cdots \geq w_N^O$ and $w_1^R \geq w_2^R \geq \cdots \geq w_N^R$. Denote $W^O(b)$ and $W^R(b)$ the makespan of the two algorithms respectively. Let $L(b)$ be a non-increasing list of all N^2 entries in b, for the sake of our analysis. The i-th largest entry in batch b is denoted as $L_i(b)$. In our analysis, we always assume for any input batch. Thus the input b in all notations will be omitted unless otherwise specified.

Given the above setting, we first observe the following proposition.

Proposition 1: $w_i \geq L_{(i-1)N+1}$ for any $1 \leq i \leq N$, regardless the scheduling algorithm.

Proof: Suppose there exists a certain k such that $w_k < L_{(k-1)N+1}$. Since each configuration contains exactly

N entries, $k-1$ configurations with weights $w_1, \ldots, w_{k-1}$ can cover at most $N(k-1)$ entries, There are two cases. If $L_{(k-1)N+1}$ is covered by any of the $k-1$ configurations, the configuration with weight w_k must cover some entry L_x that appears before $L_{(k-1)N+1}$ in the list. Thus $w_k \geq L_x \geq L_{(k-1)N+1}$ leads to a contradiction. Otherwise, $L_{(k-1)N+1}$ must be covered by the configuration with weight w_k. It leads us to a contradiction too. ■

In the most optimistic case where $W = \sum w_i$ is minimized, each configuration i has weight $w_i = L_{(i-1)N+1}$ and contains $L_{(i-1)N+1}$ up to L_{iN}. Therefore, no matter what scheduling algorithm we employ, $w_i \geq L_{(i-1)N+1}$ holds.

The next lemma is going to bound the expected weights of RandRR by the weights of OPT.

Lemma 1: $E[w_i^R] \leq \frac{1}{i}\sum_{k=1}^{i} w_k^O$ for any $1 \leq i \leq N$, where $E[w_i^R]$ is the expected value of the ith largest weight incurred by RandRR.

Proof: By Proposition 1, we have $w_i^R \geq L_{(i-1)N+1}$. That means w_i^R must appear within the range $[L_1, L_{(i-1)N+1}]$ in list L. As we assume the uniform distribution of its occurrence, it can be estimated to be the arithmetic mean of the range substracting the previously determined weights. That is,

$$\begin{aligned} E[w_i^R] &= \frac{1}{iN-i-N+2}\left(\sum_{k=1}^{(i-1)N+1} L_k - \sum_{k=1}^{i-1} w_k^R\right) \\ &\leq \frac{1}{(i-1)N+1}\sum_{k=1}^{(i-1)N+1} L_k \\ &\leq \frac{1}{(i-1)N+1}(NL_1 + NL_{N+1} + \cdots \\ &\quad + NL_{(i-1)N} + L_{(i-1)N+1}) \\ &\leq \frac{1}{i}(L_1 + L_{N+1} + \cdots + L_{(i-1)N+1}). \end{aligned} \tag{2}$$

Since by Proposition 1, $w_k^O \geq L_{(k-1)N+1}$, we conclude that (2) can be further derived into

$$E[w_i^R] \leq \frac{1}{i}\sum_{k=1}^{i} w_k^O. \tag{3}$$

■

We then prove the expected approximation ratio of RandRR in the following theorem.

Theorem 3: The expected approximation ratio of RandRR is $\ln N + 1$.

Proof: By the definition of the makespan and the linearity of the expectation, $E[W^R] = \sum_{i=1}^{N} E[w_i^R]$. This can be further expressed by Lemma 1 so that

$$\begin{aligned} E[W^R] &\leq \sum_{i=1}^{N}\left(\frac{1}{i}\sum_{k=1}^{i} w_k^O\right) \\ &= \sum_{i=1}^{N}(H_N - H_{i-1})w_i^O \leq \sum_{i=1}^{N} H_N w_i^O, \end{aligned}$$

where H_i is the Harmonic number. Therefore,

$$E[W^R] \leq H_N \sum_{i=1}^{N} w_i^O = H_N W^O.$$

Because RandRR produces exactly N configurations, its expected scheduling cost with respect to any input batch b is $E[C_{RandRR}(b)] = E[W^R(b)] + N\delta$. Compare it with the optimal scheduling cost $C_{OPT}(b) = W^O + N\delta$, we obtain the expected approximation ratio for RandRR.

$$a_{expected} = \frac{E[C_{RandRR}(b)]}{C_{OPT}(b)} \leq \frac{H_N W^O + N\delta}{W^O + N\delta} \leq H_N.$$

Because $\ln N \leq H_N \leq \ln N + 1$, it follows that the expected approximation ratio of RandRR is $\ln N + 1$. ■

By randomization, we have achieved a big step in approximation ratio from N to expected $\ln N + 1$. Besides, the algorithm is fast. The time complexity of RandRR is $O(N^2)$. This is because for each configuration P_k, we spend $O(N)$ for shifting from P_{k-1} to P_k, and $O(N)$ time for determining its weight. In addition, $O(N^2)$ is the best possible asymptotic time complexity for the MBS algorithms, because any input batch has size $\Theta(N^2)$.

D. Barely Random Round Robin

The major drawback of RandRR is that it requires an external source of randomness, as large as $N!$; because it will choose a permutation matrix uniformly at random from the set Φ at each run. This may cause some difficulty in implementation. This difficulty can also be measured in terms of *random bits*. The larger the sample space is, the more random bits will be used to control the random draw. RandRR uses $O(N \log N)$ random bits for $N!$ samples, since $\log_2 N! = O(N \log N)$.

As a matter of fact, we can trade off between the number of the random bits and the approximation ratio. A *barely random algorithm* is an algorithm that uses limited (often constant) number of random bits. To modify our RANDOMIZED ROUND ROBIN to be a barely random algorithm, we first fix a small set H of the permutation matrices. Note that the permutation matrices in H are pairwise *unrelated*. Otherwise, we could keep one of the *related* permutations and obtain a smaller set H. The BARELY RANDOM ROUND ROBIN (BRRR) works as follows.

Algorithm BRRR(D)

1) Randomly choose P_1 from the set H.
2) Set $P_k = P_1 \oplus k$, for $1 \leq k < N$.
3) Find the largest entry in D that covered by P_k and set it to be w_k.
4) Output.

Again the difference lies in the first step of the procedure only. Therefore, the time complexity of BRRR is $O(N^2)$ too. Let $h = |H|$ be the size of the sample space of BRRR. The number of random bits used by BRRR is $\log_2 h$, which is

much less than that of RandRR. However, the performance is also downgraded. This is because Lemma 1 may not hold any more. When fewer permutations are used, $E[w_i^R]$ can not be estimated by the arithmetic mean. Instead, some adversarial input may force w_i^R take large values as it can not average the worst case by full randomization. It is not hard to see that when $h = 1$, BRRR is actually NRR; when $h = N!$, BRRR is actually RandRR. Intuitively BRRR performs better than NRR but worse than RandRR (because of less random bits) in terms of the approximation ratio. Therefore, we conjecture that the approximation ratio of BRRR is a function of h that lies in the range of $[\ln N + 1, N]$.

IV. Experiments and Remarks

The purpose of this simulation is to evaluate the actual performance of our MBS algorithms in terms of their approximation ratios. Notice that because we do not have any idea of the reconfiguration delay δ, the ratio of the makespans W_{ALG}/W_{OPT} are measured instead of the actual approximation ratio. In fact, W_{ALG}/W_{OPT} is a loose upper bound for the approximation ratio. Though it is not very accurate, it does provide some useful information.

In our first experiment, we fix the size of the switch to be $N = 128$. The input traffic matrices have random entries. The input traffic is defined as follows: Packets arrive at the inputs according to the independent and identical distributed (i.i.d.) Bernoulli processes. The traffic is admissible with respect to the batch length T. To test for different value of T, we made it also random and $T \gg N$. We depict the statistics of 10^4 runs of the three algorithms in Fig. 2. The average of these 10^4 sample executions is about 1.70, with standard deviation around 0.05, for all three algorithms. All the outputs vary within the range of 1.60 to 1.85, and form approximately a Gaussian distribution.

In Fig. 2, we observe that the performance of the three algorithms are almost identical. This is because on a fixed input, any randomized algorithm behaves exactly as a deterministic algorithm after making its random decision. On some inputs, the randomized algorithm will perform worse than the deterministic one, and better on the other inputs. The performance will adjust itself on a large number of random inputs. The figure merely portraits our expectation.

Our second experiment tests how BRRR performs on various sizes of the set H of initial permutations. We fixed the size of the switch $N = 128$, and $h = |H|$ varies from 1 to 128. The average approximation ratio with respect to each h is shown in Fig. 2. We find that BRRR is not sensitive to the change of h, when the inputs are random. It is due to the same reason described in the previous paragraph.

Finally, we tested our algorithms with respect to different switch sizes. Each algorithm will execute on 10^3 uniformly random inputs for each switch size. Although the sizes of the switch are not necessarily exponents of 2, we choose it this way to observe the trend more easily. The statistical properties are summarized in Table I. As the size increases, the performance of the three algorithms becomes stable. The average tends to be the same and the standard deviation decreases. We notice that W_{ALG}/W_{OPT} of the three algorithms increase proportional to $\ln N$ but are much less than $\ln N$. This is because the larger the switch size, the more empty slots it would produce. When the input traffic is also uniform, Lemma 1 gives excellent estimation of the expected weights of the three algorithms. Thus the arguments in the proof of Theorem 3 also describe the trend we observed in Table I. We may also observe that W_{ALG}/W_{OPT} do not exceed 2, which implies that a speedup of 2 is sufficient in general.

TABLE I

Algorithms' statistical performance of W_{ALG}/W_{OPT} on various switch sizes (over 10^3 uniformly random inputs)

	Average			Standard Deviation		
N	NRR	RandRR	BRRR	NRR	RandRR	BRRR
2	1	1	1	0	0	0
4	1.1904	1.1938	1.1924	0.1454	0.1501	0.1476
8	1.3442	1.3458	1.3429	0.1276	0.1267	0.1255
16	1.4599	1.4599	1.4604	0.0940	0.0939	0.0942
32	1.5381	1.5381	1.5381	0.0694	0.0694	0.0694
64	1.6663	1.6663	1.6663	0.0489	0.0489	0.0489
128	1.6916	1.6916	1.6916	0.0362	0.0362	0.0362
256	1.7547	1.7547	1.7547	0.0252	0.0252	0.0252
512	1.8091	1.8091	1.8091	0.0183	0.0183	0.0183

TABLE II

Summary of MBS Algorithms

	Time	*(Expected) A.R.*	*# Random Bits*
NRR	$O(N^2)$	N	Deterministic
RandRR	$O(N^2)$	$\ln N + 1$	$O(N \log N)$
BRRR	$O(N^2)$	$[\ln N + 1, N]$	$\log_2 h$

V. Conclusion

We conclude this paper by summarizing the results in Table II. In addition to the theoretical performance bounds, we also provide the simulation results. These results actually suggest that the approximation ratio study underestimates the algorithms' performance under the randomly generated traffic.

Here in this research, we only study the case when the number of configurations $N_c = N$. It is interesting to study the tradeoff between the number of configurations and the performance. It will also be important to understand whether preemption would lead to better approximation ratios in theory and in practice.

References

[1] I. S. Gopal and C. K. Wong. Minimizing the number of switchings in an SS/TDMA system. *IEEE Transactions on Communications*, vol. 33, pp. 497 – 501, 1985.

[2] P. B. Chu, S. S. Lee and S. Park. MEMS: The Path to Large Optical Crossconnects *IEEE Communications Magazine*, vol. 40, Issue 3, pp. 80-87, Mar. 2002.

[3] J. E. Fouquet, S. Venkatesh, M. Troll, D. Chen, H. F. Wong, and P. W. Barth. A compact, scalable cross-connect switch using total internal reflection due to thermally-generated bubbles. In *Proceedings of Lasers and Electro-Optics Society Annual Meeting*, pp. 169 – 170, 1998.

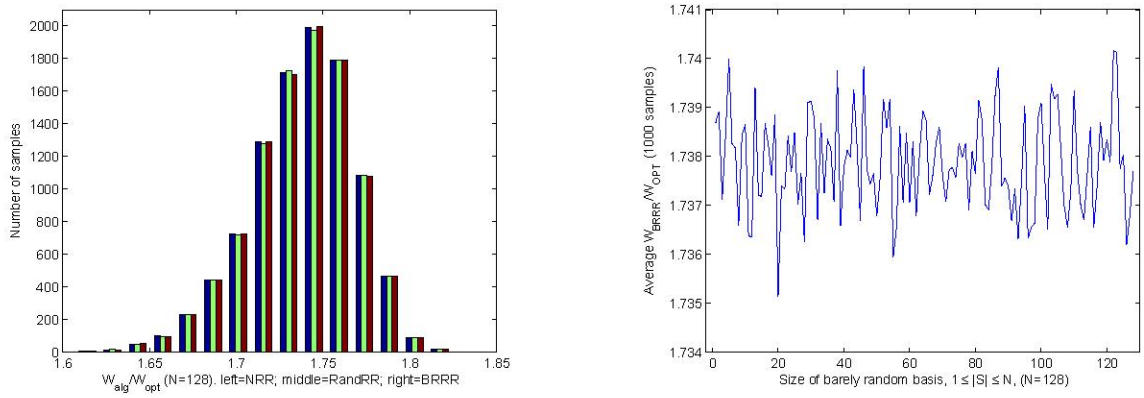

Fig. 2. Statistical distribution of W_{ALG}/W_{OPT} (left) and Average W_{BRRR}/W_{OPT} v.s. $|H|$ (right) on a 128×128 switch.

[4] O. B. Spahn, C. Sullivan, J. Burkhart, C. Tigges, and E. Garcia. GaAs-based microelectromechanical waveguide switches. In *Proceeding of IEEE/LEOS International Conference on Optical MEMS*, pp. 41 – 42, 2000.

[5] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, vol. 2(4), pp. 225 – 231, 1973

[6] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 434 – 443, 1990.

[7] B. Towles and W. J. Dally. Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Transactions on Networking*, vol. 11, pp. 835 – 847, 2003.

[8] Z. Zhou, X. Li, and M. Hamdi. Fast scheduling for optical packet switches with minimum configurations. In Proceedings of *IEEE Workshop on High Performance Switching and Routing (HPSR05)*. Hong Kong, May 2005.