Switch Simulations Based on Workload Pattern Generation and Smoothed Periodic Input

Qiang Zheng[1], Si-Min He[1], Shu-Tao Sun[2], Yan-Feng Zheng[1], Wen Gao[1]
[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
[2] Communication University of China, Beijing 100024, China
qzheng@jdl.ac.cn, smhe@ict.ac.cn, stsun@jdl.ac.cn, yfzheng@jdl.ac.cn, wgao@ict.ac.cn

***Abstract*—Simulation is crucial to performance evaluation of switches. Currently stochastic simulation is the predominant approach, which has two drawbacks: few workload patterns and long simulation times. In this paper, we propose a novel switch simulation method that is based on two techniques: exhaustive workload pattern generation and smoothed periodic input generation. The exhaustive workload pattern generation can produce a huge number of nonuniform workload patterns, which far exceeds the traditional few ones; conclusions based on such truly extensive simulations are much more convincing. The periodic input or cell arrival pattern outperforms the stochastic counterpart in terms of easy repetition and fast convergence of simulation; in particular, among periodic inputs, the smoothed periodic input is most favorable to switch scheduling, and hence switches performing poor with it probably performs worse under others. Combining these two techniques together can systematically identify lots of stuck states at which some switches show poor performances such as low throughput. Specifically, this method discovers that the throughput of iSLIP, FIRM and DRRM, each with one iteration, may be lower than 60% under certain nonuniform and smoothed periodic traffic pattern.**

Keywords-Performance evaluation, simulation, switches.

I. INTRODUCTION

Performance evaluation is crucial to the research of switch architectures and scheduling algorithms. Theoretical analysis and computational simulation, as two major means of performance evaluation, have their own strengths and weaknesses. Conclusions of the theoretical analysis have good generality. For example, under Bernoulli uniform traffic, an input-queued switch with FIFO (first-input-first-output) queues has a throughput limited to just 58.6% [9]; under any admissible traffic, an input-queued switch with virtual output queues and maximum weight matching schedulers can achieve 100% throughput asymptotically [17] [5] [14]. All of these theoretical analyses use stochastic models and obtain asymptotic conclusions for the average cases, which are not quite relevant to practices. Furthermore, as switch fabrics and scheduling algorithms become increasingly complex, theoretical analysis may become too difficult to draw any conclusion.

In contrast to theoretical analysis, simulation can be performed easily and in a wider range. Any switch architecture, and/or any switch scheduler, is amenable to simulation, and has to be evaluated by simulation. The first step is to generate some traffic patterns, which has two components. One is the workload pattern generation, i.e., generating a rate matrix with each element indicating the normalized rate of flow for an input-output channel. It is uniform if all the elements are equal and nonuniform otherwise. The other is the input pattern or cell arrival pattern generation, i.e., generating flows of cells at each input port with the specified rates. Influenced by the models for theoretical analysis, the workload patterns are usually admissible and identically loaded at each input port, and input patterns can be Bernoulli, exponential, and so on.

While it has been the predominant approach to performance evaluation, stochastic simulation has two weaknesses. The first weakness is that stochastic simulation only adopts few workload patterns. Specifically, let ρ denote the normalized load at each input port and $\lambda_{i,j}$ as the rate of flow arriving at input i and destined for output j, there are four nonuniform patterns used most:

Pattern 1 (*Diagonal*) [13][6][22][1]: $\lambda_{i,i} = 2\rho/3$ and $\lambda_{i,i+1} = \rho/3$ for all i from 0 to $(N-1)$, and $\lambda_{i,j} = 0$ for all other i and j. The operation '+' is subject to modulo N. An example is Fig.1 (a).

Pattern 2 (*Log-diagonal*) [6][22][1]: Arrival rates at the same input differ exponentially; i.e., $\lambda_{i,i+j} = 2\lambda_{i,i+j+1}$, where $0 \le j \le N-2$. An example is Fig. 1 (b).

Pattern 3 (*Lin-diagonal*) [2][1]: Arrival rates at the same input differ linearly; i.e., $\lambda_{i,i+j} - \lambda_{i,i+j+1} = 2\rho/N(N+1)$, where $0 \le j \le N-2$, or $\lambda_{i,i+j} = 2\rho\,(N-j) / (N^2+N)$ where $0 \le j \le N-1$. An example is Fig. 1 (c).

Pattern 4 (*Unbalanced*) [12][19][20][18]: Let w denote the unbalanced probability, then $\lambda_{i,j} = \rho\,(w + (1-w)/N)$ if $i = j$ and $\lambda_{i,j} = \rho\,(1-w)/N$ otherwise. When $w = 0$, the traffic is uniform. When $w = 1$, it is completely unbalanced, i.e., the traffic at input i is only destined for output i. An example is Fig. 1 (d).

$$\begin{bmatrix} 0.6 & 0.3 & 0 \\ 0 & 0.6 & 0.3 \\ 0.3 & 0 & 0.6 \end{bmatrix} \quad \begin{bmatrix} 0.4 & 0.2 & 0.1 \\ 0.1 & 0.4 & 0.2 \\ 0.2 & 0.1 & 0.4 \end{bmatrix} \quad \begin{bmatrix} 0.3 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.2 \\ 0.2 & 0.1 & 0.3 \end{bmatrix} \quad \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

(a) (b) (c) (d)

Figure 1. Examples of four nonuniform workload patterns.
(a) ρ = 0.9; (b) ρ = 0.7; (c) ρ = 0.6; (d) ρ = 1.0, w = 0.4.

Current experimental evaluation of switch performances under nonuniform traffic is mainly based on the above four patterns. However, simulation results from so few workload

This work was supported in part by the National Natural Science Foundation of China under Grant 69983008 and the Institute of Computing Technology, Chinese Academy of Sciences under Grant 20056090.

patterns are merely proper to explain the performance under these patterns; they are not convincing enough to predict the performance under general nonuniform traffic. In particular, with so few workload patterns, we can hardly obtain any worst-case performances, such as the worst-case throughput. So far, there are only sporadic reports of stuck states at which switches show quite bad performances; e.g., a 3×3 switch with iSLIP can only achieve a throughput of 66.7% under certain input [3]. To our knowledge, no method for systematically searching the stuck states has been reported till now. In Section IV, we will use our simulation method to systematically seek the potential bad throughput of an input-queued switch operated by iSLIP and other schedulers.

The second weakness of stochastic simulation is that it always needs quite a long time before reaching the steady state. In the stochastic simulation, arriving cells are randomly generated according to the workload pattern. Generally, this simulation has to run as many as hundreds of thousands of time slots before converging to the steady state.

In this paper, we propose a novel switch simulation method, based on two techniques: exhaustive workload pattern generation and smoothed periodic input generation, aiming to remove the two weaknesses of stochastic simulation. The rest of the paper is organized as follows. In Section II, we will elaborate on the workload pattern generation methods, including the exhaustive generation and the random generation. In Section III, we will explain the concept of smoothness and how to generate the smoothed periodic input. In Section IV, we will use some examples to show the application of the workload pattern generation and the smoothed periodic input. Finally, in Section V, we will conclude this paper with future works.

II. WORKLOAD PATTERN GENERATION

A workload pattern for an N by N switch, either uniform or nonuniform, can be expressed as a matrix $\Lambda = (\lambda_{i,j})_{N\times N}$, where $i, j = 0, \ldots, N-1$, and $\lambda_{i,j} \geq 0$ for all i and j. The commonly used workload pattern is admissible (i.e., $\sum_i \lambda_{i,j} \leq 1$ for all j and $\sum_j \lambda_{i,j} \leq 1$ for all i), and the offered loads at each input and at each output are all equal to ρ (i.e., $\sum_j \lambda_{i,j} = \rho$ for all i and $\sum_i \lambda_{i,j} = \rho$ for all j), where $0 < \rho \leq 1$. Therefore, they can be transformed into $\rho\cdot\Lambda'$ where Λ' is a doubly stochastic matrix ((i.e., $\sum_i \lambda'_{i,j} = 1$ for all j and $\sum_j \lambda'_{i,j} = 1$ for all i) and is called a distribution matrix in this paper. For all practical purposes, elements in the matrices are rational numbers. Therefore, a doubly stochastic matrix, or a distribution matrix, is equivalent to an integer matrix $A = (a_{i,j})_{N\times N}$ whose sum of each line (row or column) is equal to a common integer D and $a_{i,j}$ is a nonnegative integer between 0 and D. Now each element of the workload matrix is expressed as $\lambda_{i,j} = \rho\cdot a_{i,j}/D$.

In traditional simulations, ρ varies from 0.1 to 1.0, and four types of nonuniform distribution matrix are used. If more types of distribution matrix can be generated and tested, we can be more confident about the conclusions of performance evaluation. We propose two methods to produce the matrices: the exhaustive generation and the random generation.

A. Exhaustive generation

The exhaustive generation means that given integer parameters N and D, generate all $N\times N$ integer matrices A such that $a_{i,j} \geq 0$, and the sum of each line, row or column, is equal to D. Owing to space limitation, the exhaustive generation algorithm and the proof of its correctness will be published elsewhere soon [8]. Table I lists some numbers of matrices produced by exhaustive generation with some small N and D. It can be anticipated that the number will increase exponentially with N and D.

While it has the best coverage, simulation based on exhaustive workload pattern generation has poor scalability. However, when N and D are small, such exhaustive simulation is feasible. Exhaustive simulation at small size with *good* result might *not* be *sufficient* to guarantee something for larger size, but it is *necessary*. In other words, exhaustive simulation at small size with *bad* result is sufficient to expel the switch from further consideration; at least we can hardly be confident about the performance for larger size.

Anyway, scalability is the bottleneck to exhaustive simulation. One way to alleviate this problem is to reduce isomorphism during generation. In the switch simulation, row interchanges and column interchanges of a workload pattern mean renaming inputs and outputs, and hence are two kinds of isomorphism. Another way, which is scalable, is to randomly sample among the distribution matrices, as explained next.

TABLE I. THE NUMBER OF MATRICES PRODUCED BY EXHAUSTIVE GENERATION.

N \ D	1	2	3	4	5	6	7	8
3	6	21	55	120	231	406	666	1035
4	24	282	2008	10147	40176	132724	381424	981541
5	120	6210	153040	2224955	22069251	164176640	976395820	*

B. Random generation

In order to solve the scalability problem of the exhaustive generation method, we propose a random generation algorithm, which is a simple variant of the exhaustive generation algorithm [8]. Given a positive integer D, we can randomly generate any number of $N\times N$ integer matrices with each line sum equal to D.

The random generation is no longer restricted to small N and D. Actually, any large N and D can be used. What's more, any number of workload patterns can be generated. Therefore such random generation method is both scalable and flexible. In practice, besides testing the commonly used types of workload patterns, we can additionally test random workload patterns as long as resource permits or until we are confident enough about the conclusion. Simply speaking, such random simulation is between the two extremes of the traditional restricted simulation and the newly proposed exhaustive simulation.

The generated workload pattern can be used in conjunction with any input pattern, either the stochastic input or the periodic input, including the smoothed periodic input that will be introduced in the next section.

III. SMOOTHED PERIODIC INPUT

While stochastic traffic patterns are predominant in simulation, periodic traffic patterns are reported sporadically to have special *bad* effects. For example, a periodic traffic pattern may make the throughput of an input-queued switch with FIFO

queues as small as $1/N$ [11]; another periodic traffic pattern makes a 3×3 input-queued switch with virtual output queues and iSLIP scheduler enter a stuck state with a throughput of only 2/3 or 66.7% [3]; still another *pathological* periodic traffic can make a basic load-balanced router deliver a throughput of only $1/N$ [10]. All these throughput results are in sharp contrast to the corresponding theoretical stochastic analysis results. For example, it is proved that a basic load-balanced switch guarantees 100% throughput for any stationary and weakly mixing arrival patterns with admissible mean rate [3].

We prefer periodic input patterns to stochastic ones in simulation for several reasons. First, periodic patterns are much easier to understand than stochastic ones. We wonder if most people in the field understand the essence of stochastic models such as a weakly mixing stochastic process. Second, periodic patterns and the associated simulation results are easy to repeat or reproduce since they have no variance incurred by randomness. Third and most important, periodic input patterns are more likely to cause poor performances of scheduling algorithms, such as low throughput, than the stochastic counterparts, as mentioned above. We speculate that these phenomena are not uncommon; they must be governed by hidden laws.

We know that almost all switches are deterministic since generating randomness in high speed is not feasible [15]. Therefore a deterministic switch can have only a finite number of states such as memory occupancy states and round-robin pointer states. If the switch is not designed properly, the number of switch states might not be able to beat the larger number of input patterns, and the switch is probable to behave poorly under certain adversarial input patterns. The invariant periodic patterns are easy to make the bad states of the switch hold on indefinitely, and hence speed up the emergence of poor performances. In contrast, a stochastic input can hardly maintain any fixed pattern and hence is difficult to unveil the poor performance. In this paper we make some initial attempts.

There are several methods to generate a periodic input. Here, we propose a periodic arrival pattern called the smoothed periodic input, which is based on the algorithm sMUX [7]. Next, we will introduce the concept of smoothness and the algorithm sMUX, and then explain how to generate smoothed periodic traffic for the switch simulation.

A. Smoothness and sMUX

There are n flows of fixed-size cells sharing a link of bandwidth r; each flow f_i has a reserved bandwidth r_i, where $r_i > 0$ and $\sum_i r_i \le r$. This specifies an instance $(r; r_1, r_2, ..., r_n)$, which can be reduced to its normal form $(1; w_1, w_2, \ldots, w_n)$, abbr. $(w_1, w_2, \ldots, w_n)$, in which $w_i = r_i / r > 0$ and $\sum_i w_i \le 1$. For all practical purposes, bandwidth and weight parameters are supposed to be rational numbers.

Time is slotted, with slot t denoting the real interval $[t, t+1)$, and slot interval $[t_1, t_2)$ denoting the slot set $\{t_1, t_1+1, \ldots, t_2-1\}$. A *schedule* or *scheduler* S for an instance $(w_1, w_2, \ldots, w_n)$ is a function S: $[t_1, t_2) \rightarrow \{\square, \tau_1, \tau_2, \ldots, \tau_n\}$, mapping slots to cells; cell $\square$ stands for the type of empty cells and cell τ_i stands for the type of cells of flow f_i.

The smooth multiplexing problem (SMP) is to generate a *smooth* schedule such that occurrences of each cell τ_i, or equivalently, cells of each flow f_i, are *smoothly* or *evenly* distributed in the whole sequence. Intuitively, in an ideally smooth schedule for an SMP instance $(w_1, w_2, \ldots, w_n)$ any interval of l consecutive slots should cover $(l \cdot w_i)$, or in practice, either $\lfloor l \cdot w_i \rfloor$ or $\lceil l \cdot w_i \rceil$ number of cell τ_i. Such intuitive view of covering, along with integral constraint, shall be taken into account during formalization.

Let $Cover(S, \tau_i, t, l)$, abbr. $Cover_i(t, l)$, denote the number of cell τ_i that are scheduled by scheduler S inside slot interval $[t, t + l)$. By investigating the whole spectrum of $Cover_i(t, l)$ and its worst-case deviation from an ideal distribution on slot intervals starting from arbitrary slot t with arbitrary length l, we obtain a series of measures that gradually become independent of t and l and only dependent on the schedule and the cell.

Given an arbitrary slot interval $L = [t_1, t_2)$, $t_1 < t_2$, which could be finite ($t_2 < +\infty$) or infinite ($t_2 = +\infty$), the minimum and the maximum covers of length l within this interval can be defined as follows:

Minimum cover $cvr(S, \tau_i, l)$
$= min_t\{Cover_i(t, l) \mid [t, t + l) \subseteq [t_1, t_2)\}$, abbr. $cvr_i(l)$;

Maximum cover $CVR(S, \tau_i, l)$
$= max_t\{Cover_i(t, l) \mid [t, t + l) \subseteq [t_1, t_2)\}$, abbr. $CVR_i(l)$.

We stipulate that $cvr_i(0) = CVR_i(0) = 0$. We measure the covering smoothness of the actual distribution of cell τ_i within interval $[t_1, t_2)$ by the following two worst-case covering deviations from the ideal:

Covering sparseness deviation $cvr\text{-}dev_i$
$= max_l\{\lfloor l \cdot w_i \rfloor - cvr_i(l) \mid 0 \le l \le t_2 - t_1\}$;

Covering burstiness deviation $CVR\text{-}dev_i$
$= max_l\{CVR_i(l) - \lceil l \cdot w_i \rceil \mid 0 \le l \le t_2 - t_1\}$.

Note that $cvr\text{-}dev_i \ge 0$ and $CVR\text{-}dev_i \ge 0$ under such definitions. Within interval $[t_1, t_2)$, if $cvr\text{-}dev_i = CVR\text{-}dev_i = 0$, or equivalently, if $\lfloor l \cdot w_i \rfloor \le Cover_i(t, l) \le \lceil l \cdot w_i \rceil$ for any interval $[t, t + l) \subseteq [t_1, t_2)$, the distribution of cell τ_i should be ideal from the covering point of view.

Besides the proportion w_i, each flow f_i is additionally associated with an *initiation time* I_i to mark the earliest time slot the scheduler is ready to schedule flow f_i. This is to reflect the dynamic scheduling of newly admitted flows in practical applications. Starting from time I_i, the j-th ($j = 1, 2, 3\ldots$) cell service provided for flow f_i is *eligible* at time $e_{i,j} = I_i + (j-1)/w_i$ and is expected to finish before *deadline* $d_{i,j} = I_i + j/w_i$. That is, the jth cell of flow f_i is expected to be allocated in the j-th window $[I_i + (j-1)/w_i, I_i + j/w_i)$.

Next, we give the algorithm sMUX and two theorems about sMUX. Due to space limitation, please refer to [7] for the scheduling example of sMUX and the proof of two theorems.

Algorithm sMUX: *At each time slot t, among those flows that are eligible for scheduling at time t, i.e., their cells to be serviced have eligible times no later than t ($e_{i,j} \le t$, equivalently, $\lceil e_{i,j} \rceil \le t$), allocate slot t to the flow with the earliest upper rounded deadline $\lceil d_{i,j} \rceil$; ties are broken arbitrarily. When no flow is eligible, slot t is left idle.* □

Theorem 1 *In a sMUX schedule for an SMP instance $(w_1, w_2, \ldots, w_n)$ with initiation times $(I_1, I_2, \ldots, I_n)$, the j-th ($j = 1, 2\ldots$) cell service of flow f_i occurs within the slot interval $[\lceil e_{i,j} \rceil, \lceil d_{i,j} \rceil)$, where $e_{i,j} = I_i + (j-1)/w_i$ and $d_{i,j} = I_i + j/w_i$.* □

Theorem 2 *Given a sMUX schedule for an SMP instance* $(w_1, w_2, \ldots, w_n)$ *with initiation times* $(I_1, I_2, \ldots, I_n)$. *Then for any interval length* l *and step size* s, *the sMUX schedule has the following properties*:

(1) $\lfloor l \cdot w_i \rfloor \le Cover(S, \tau_i, I_i, l) \le \lfloor (l-1) \cdot w_i \rfloor + 1$,

(2) $\lfloor (l+1) \cdot w_i \rfloor - 1 \le cvr_i(l) \le CVR_i(l) \le \lceil (l-1) \cdot w_i \rceil + 1$,

(3) $cvr\text{-}dev_i \le 1$, $CVR\text{-}dev_i \le 1$. □

Since certain instances have no solutions with ideal covering properties, such as the instance (1/2, 1/3, 1/6), Theorem 2 implies that algorithm sMUX is almost an optimal scheduler for generating smoothed sequences.

B. Smoothed periodic input generation

We can use each row vector of the workload matrix Λ as the parameters of sMUX to produce the smoothed periodic input at each input port. Fig. 2 is an example, where the input is periodic with a period of 6 time slots.

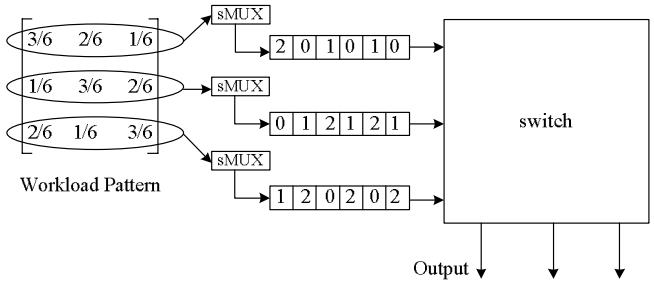

Figure 2. Smoothed periodic input generator.

In principle, the smoothed stream is most favorable for switch scheduling. Thus the smoothed periodic input should be the most schedulable stream among periodic inputs. If an algorithm performs poorly at the smoothed periodic input, it probably performs worse at other periodic inputs. Coupled with the workload pattern generation, the smoothed periodic input can be used for identifying the potential stuck states of scheduling algorithms; we will see some examples of this application in Section IV.

IV. SIMULATIONS

In Section II and Section III, we introduce two components of our simulation method. In this section, we present some new throughput results about several crossbar schedulers with both exhaustive and random workload patterns coupled with smoothed periodic inputs. The throughput is defined as the average rate among outputs during the steady state under full workload ($\rho = 1$) [9].

In our simulations, simulation time for each experiment is at most 200,000 time slots. After first 40,000 slots pass, we start to check the steady state; specifically, every 2,000 slots we calculate the average output rate; when the differences in the consecutive five results are less than 0.1%, we consider that we have already arrived at the steady state and stop the current simulation.

A. Throughput lower than 66.7% for iSLIP

A stuck state was reported that makes a 3×3 input-queued crossbar switch with iSLIP achieve only 66.7% throughput [3]. In order to systematically find the stuck states, we use the exhaustive workload pattern generation together with the smoothed periodic input generation in our simulation.

The scheduling algorithm is 1-iSLIP, and the offered load of each input is set to 1 to measure the throughput. Since the switch size is small ($N = 3$), we set parameter D to 20. The exhaustive generation can produce 26,796 workload patterns.

Our simulation discovers two workload patterns that make the 1-iSLIP achieve only 61.7% throughput, as shown in Fig. 3. Then we use the Bernoulli stochastic input under these two patterns for simulation, and 1-iSLIP achieves 75.8% and 75.4% throughput, higher than those of the smoothed periodic input.

However, the worst throughput of 4-iSLIP found by this method is 66.7% when $N = 3$ and $D = 20$, just the same as before. But by our method, we can find another workload pattern shown in Fig. 4 that makes a 4×4 switch with 4-iSLIP achieve a throughput of only 65.6%, lower than 66.7%. The same workload pattern results in a throughput of 85.6% under the Bernoulli stochastic input.

$$\begin{bmatrix} 9 & 1 & 10 \\ 6 & 13 & 1 \\ 5 & 6 & 9 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 10 \\ 13 & 7 & 0 \\ 2 & 8 & 10 \end{bmatrix}$$

Figure 3. Workload patterns make a 3×3 switch with 1-iSLIP achieve 61.7% throughput.

$$\begin{bmatrix} 4 & 0 & 3 & 1 \\ 1 & 2 & 0 & 5 \\ 1 & 1 & 4 & 2 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

Figure 4. A workload pattern makes a 4×4 switch with 4-iSLIP achieve 65.6% throughput.

B. New throughput analysis of five schedulers

For a long time, the throughput comparison of iSLIP, RRM [15], FIRM [21] and DRRM [4][12] under nonuniform traffic has been based on few workload patterns and the stochastic input pattern. Now we investigate the throughputs of these algorithms by our new simulation method.

We use a 4×4 crossbar switch and set the parameter D to 7 and 8, which can produce 381,424 and 981,541 matrices respectively. The offered load is 1, and algorithms iterate one and four times. All pointers are initialized to be 0. The throughput results are shown in Tables II to V. We also use the random generation of workload patterns to test larger instances; specifically, $N = 16$, $D = 100$, and 10,000 instances are sampled in each simulation, with results shown in Tables VI and VII.

Note: DRRM1 and DRRM2 in the following tables denote respectively the DRRM algorithm in [4] and [12]. The former is the 2-step counterpart of iSLIP, and the latter is the 2-step counterpart of FIRM. In our expectation, a 3-step scheduler and its 2-step counterpart should behave the same under exhaustive workload patterns.

TABLE II. THROUGHPUT RESULTS WITH ONE ITERATION, $D = 7$.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.571	0.813	0.066	0.052
FIRM	0.625	0.814	0.062	0.048
RRM	0.250	0.742	0.102	0.075
DRRM1	0.536	0.788	0.068	0.052
DRRM2	0.571	0.797	0.062	0.049

TABLE III. THROUGHPUT RESULTS WITH FOUR ITERATIONS, $D = 7$.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.714	0.878	0.053	0.043
FIRM	0.714	0.887	0.050	0.039
RRM	0.714	0.879	0.053	0.042
DRRM1	0.536	0.788	0.068	0.052
DRRM2	0.571	0.797	0.062	0.049

TABLE IV. THROUGHPUT RESULTS WITH ONE ITERATION, $D = 8$.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.562	0.802	0.061	0.048
FIRM	0.594	0.805	0.057	0.044
RRM	0.250	0.731	0.094	0.070
DRRM1	0.563	0.780	0.060	0.046
DRRM2	0.594	0.786	0.056	0.043

TABLE V. THROUGHPUT RESULTS WITH FOUR ITERATIONS, $D = 8$.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.656	0.872	0.049	0.038
FIRM	0.687	0.884	0.045	0.036
RRM	0.656	0.863	0.045	0.036
DRRM1	0.563	0.780	0.060	0.046
DRRM2	0.594	0.786	0.056	0.043

TABLE VI. THROUGHPUT RESULTS WITH ONE ITERATION. RANDOM GENERATION 10,000 TIMES. N= 16, D = 100.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.663	0.726	0.015	0.011
FIRM	0.664	0.725	0.015	0.012
RRM	0.650	0.686	0.014	0.011
DRRM1	0.610	0.683	0.018	0.014
DRRM2	0.605	0.682	0.018	0.014

TABLE VII. THROUGHPUT RESULTS WITH FOUR ITERATIONS. RANDOM GENERATION 10,000 TIMES. N= 16, D = 100.

Algorithm	Worst	Average	Standard Deviation	Mean Absolute Deviation
iSLIP	0.814	0.860	0.010	0.008
FIRM	0.834	0.869	0.009	0.007
RRM	0.820	0.865	0.010	0.008
DRRM1	0.610	0.683	0.018	0.014
DRRM2	0.605	0.682	0.018	0.014

We make the following observations from the tables:

(1) The worst throughputs of these algorithms are lower than 72% even with four iterations, which is not reported before. In particular, iSLIP has a worst throughput of 65.6% even with four iterations, which is worse than the 66.7% result [3], as already mentioned.

(2) The standard deviation and the mean absolute deviation are based on simulation results of all the generated matrices. They are small for iSLIP, FIRM and DRRM with either one or four iterations; for RRM, the two deviations are large with one iteration, but become small with four iterations. Anyway, such small deviations indicate that the throughputs of these algorithms under nonuniform traffic are quite stable.

(3) Comparing both the worst and the average throughputs under both one and four iterations, FIRM is slightly better than the other four algorithms.

(4) In 1,362,965×5×2 simulations for Tables II-V, the longest simulation time needs less than 20,000 slots after the initial 40,000 slots, much shorter than the stochastic simulation.

(5) The throughput of either DRRM1 or DRRM2 is not improved after the number of iteration is increased. This is due to their pointer updating policies [12]. In general, the 2-step algorithms while symmetric to the 3-step algorithms (DRRM1 vs. iSLIP, DRRM2 vs. FIRM), seem inferior to the latter in terms of throughput, which is beyond our expectation and needs further investigation.

(6) There is an interesting phenomenon: the worst throughput of 4-DRRM1 or 4-DRRM2 when $D = 7$ is smaller than that of $D = 8$. We use M_7 and M_8 to express the set of matrices produced when D is 7 and 8. Since 8 is not a multiple of 7, the set M_7 is not a subset of M_8, though smaller than M_8.

(7) With random sampling for relatively large-scale instances, the results are similar to those of the exhaustive generation for small-scale instances; e.g., throughputs are stable, FIRM is best, 2-step schedulers are relatively poor.

V. CONCLUSIONS AND FUTURE WORK

Simulation is indispensable to performance evaluation of switches since theoretical analysis is quite difficult. However, simulation models are greatly influenced and hence constrained in some aspect by the stochastic models used in theoretical analysis (e.g., the stochastic input patterns), and sometimes are quite ad hoc in some other aspect (e.g., the few workload patterns). Since switch design, and perhaps most equipment designs, are oriented towards the worst-case guarantee or catastrophe prevention, current stochastic simulation is inadequate.

Our first contribution is the use of workload pattern generation to make the simulation results more convincing for non-uniform traffic. Specifically, two scenarios for the use of the idea are proposed: the exhaustive generating of workload patterns is suitable for small-scale simulations (e.g., small N), with an aim to identify potential flaws (e.g., extremely low throughput) in the switch design; the random sampling of the workload patterns is a tradeoff between scalability and accuracy.

Our second contribution is use of the smoothed periodic input pattern to quickly find the potential flaws in the switch design. We notice that the *sporadic* reports of quite bad throughputs of certain switches are all based upon periodic

input patterns, and we propose a *systematic* method for the same task, which uses the exhaustive workload patterns and the smoothed periodic input patterns. Smoothed admissible traffic patterns seem to be most favorable to switches, but they still can find poor throughput that is lower than previous reports.

What we intend to do is to develop a new performance evaluation method that is deterministic, worst-case oriented, and non-asymptotic, hence more relevant to practices. Still much more work to be done, and let's name a few.

First, more periodic patterns should be tested. Given a workload pattern for an input, say (1/2, 1/3, 1/6), besides the smoothed periodic pattern (0 1 0 1 0 2), there can be the clustered periodic pattern (0 0 0 1 1 2), and the smoothed bursty patterns with different burst length, say (0 0 1 1 0 0 1 1 0 0 2 2) with a burst length of 2. What are their different effects on the simulation? Preliminary experiments indicate that the clustered pattern can deliver quite bad throughput, while the smoothed pattern can converge quickly.

Second, the steady state of the simulation could be determined in an accurate way. Under the stochastic model, it is difficult to judge whether the simulation has entered the steady state. When the input is periodic, since the switch has only a finite number of states, each output will converge to a deterministic sequence. Therefore, we can accurately determine the steady state, i.e. compute the period of the sequence.

Third, after we compute the steady state, we can accurately measure the performances. Besides throughput, rate guarantee can also be measured. With smoothed input, any smoothness deviation in the output sequence is due to the switch; hence fairness and jitter performances of the switch (including the scheduler) can be measured accurately.

Fourth, with a complete method available in hand, more switches can be tested under more realistic models. For example, two assumptions are used for theoretical analysis of stability: the admissible traffic is Bernoulli i.i.d. or satisfies a strong law of large numbers; the latter is more realistic. A periodic input is not a Bernoulli i.i.d. input, but satisfies the strong law of large numbers. Hence, any switch that is stable under Bernoulli traffic should be tested by periodic inputs. If the switch is stable in simulations with extensive periodic input patterns, this may be evidence of stability under admissible traffic satisfying the strong law of large numbers. If not, in particular, the switch is surely unstable under admissible traffic satisfying a strong law of large numbers, and hence may be not suitable for practical use.

REFERENCE

[1] A. Baranowska, G. Danilewicz, W. Kabacinski, J. Kleban, D. Parniewicz, and P. Dabrowski, "Performance evaluation of the multiple output queueing switch under different traffic patterns," *Proc. IEEE GLOBECOM '05*, Nov./Dec. 2005, pp. 609-613.

[2] A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "A framework for differential frame-based matching algorithms in input-queued switches," *Proc. IEEE INFOCOM '04,* Mar. 2004, PP. 1147-1157.

[3] C.S. Chang, D.S. Lee, and C.M. Lien, "Load balanced Birkhoff-von Neumann switches, Part I: One-stage buffering," *Computer Communications*, vol. 25, no. 6, Apr. 2002, pp. 611-622.

[4] H.J. Chao, "Saturn: A terabit packet switch using dual round-robin," *IEEE Comm. Magazine*, vol. 38, no. 12, Dec. 2000, pp. 78-84.

[5] J.G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," *Proc. IEEE INFOCOM '00*, Mar. 2000, pp. 556-64.

[6] P. Giaccone, Devavrat Shah, and Balaji Prabhakar, "An implementable parallel scheduler for input-queued switches," *Proc. IEEE Hot Interconnects '01*, Aug. 2001, pp. 9-14.

[7] Simin He, Shutao Sun, Wei Zhao, Wen Gao, and Yanfeng Zheng, "Smooth Switching Problem in Buffered Crossbar Switches," *Proc. ACM Sigmetrics '05*, Jun. 2005, pp. 386-387.

[8] Si-Min He and Shu-Tao Sun, "Generating doubly stochastic matrices," tech. rep., 2005.

[9] M. J. Karol, M. G. Hluchyj, and S.P. Morgan, "Input Versus Output Queueing on a Space-Division Packet Switch," *IEEE Trans. Comm.*, vol. 35, no.12, Dec. 1987, pp. 1347-1356.

[10] Isaac Keslassy, "The Load-Balanced Router," Ph.D. dissertation, Univ. Stanford, 2004.

[11] S.-Y.R. Li, "Theory of periodic contention and its application to packet switching," *Proc. IEEE INFOCOM '88*, Mar. 1988, pp. 320-325.

[12] Y. Li, S. Panwar, and H.J. Chao, "On the performance of a dual round-robin switch," *Proc. IEEE INFOCOM '01*, Apr. 2001, pp. 1688-1697.

[13] M. Ajmone Marsan, A. Bianco, E. Filippi, P. Giaccone, E. Leonardi, and F. Neri, "On the Behavior of Input Queuing Switch Architectures," *European Trans. Telecommunications*, vol. 10, no. 2, Mar./Apr. 1999, pp. 111-124.

[14] M. Ajmone Marsan, P. Giaccone, E. Leonardi, and F. Neri, "Local Scheduling Policies in Networks of Packet Switches with Input Queues," *Proc. IEEE INFOCOM '03*, Mar./Apr. 2003, pp. 1395-1405

[15] N. McKeown, "Scheduling Cells in an Input-Queued Switches," Ph.D. dissertation, Univ. California at Berkeley, 1995.

[16] N.W. McKeown and T.E. Anderson, "A Quantitative Comparison of Scheduling Algorithms for Input-Queued Switches," *Computer Networks and ISDN Systems*, vol. 30, no. 24, Dec. 1998, pp. 2309-2326.

[17] N.W. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Trans. Comm.*, vol. 47, no. 8, August 1999, pp. 1260-1267.

[18] L. Mhamdi, and M. Hamdi, "Practical scheduling algorithms for high-performance packet switches," *Proc. IEEE ICC '03,* May 2003, pp. 1659-1663.

[19] R. Rojas-Cessa, E. Oki, Zhigang Jing, and H.J. Chao, "CIXB-1: combined input-one-cell-crosspoint buffered switch," *Proc. IEEE HPSR '01*, May 2001, pp. 324-329.

[20] R. Rojas-Cessa, E. Oki, and H.J. Chao, "CIXOB-k: combined input-crosspoint-output buffered packet switch," *Proc. IEEE GLOBECOM '01*, Nov. 2001, pp. 2654-2660.

[21] D.N. Serpanos and P.I. Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues," *Proc. IEEE INFOCOM '00*, Mar. 2000, pp. 548-555.

[22] Devavrat Shah, P. Giaccone, and Balaji Prabhakar, "An efficient randomized algorithm for input-queued switch scheduling," *Proc. IEEE Hot Interconnects '01*, Aug. 2001, pp. 3-8.