

A Practical Switch-Memory-Switch Architecture Emulating PIFO OQ

Nan HUA¹, Yang XU², Peng WANG^{1,3}, Depeng JIN¹, Lieguang ZENG¹

¹Department of Electronic Engineering, Tsinghua University, Beijing 100084, P. R. China

²Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China

³Information Engineering University, Zhengzhou 450002, P. R. China

{hn, xy01, wang-p01}@mails.tsinghua.edu.cn, jindp@ee.tsinghua.edu.cn, zenglg@mail.tsinghua.edu.cn

Abstract — Emulating Output Queued (OQ) Switch with sustainable implementation cost and low fixed delay is always preferable in designing high performance routers. The Switch-Memory-Switch (SMS) router, also called Distributed Shared Memory (DSM) Switch, provides a possible way towards practically emulating OQ in backbone switches. However, the architectures and algorithms for SMS switches ever proposed are either unpractical or only supporting First-Come-First-Serve (FCFS) scheduling policy, which cannot support QoS and is unfair for light traffic flow. Our improved SMS architecture and algorithm aim at emulating Push-In-First-Out (PIFO) OQ. We employ a randomly-dispatching first stage and resolve memory access conflicts on the second stage of the switch through a probabilistic matching method, at the cost of fixed delay and sufficiently low cell loss probability (P_{CLP}). The relative fixed delay of our algorithms for an $N \times N$ switch is composed of two parts: N and $(-3/2 \log_2 P_{CLP})$, which result from the pipelined scheduling process and probabilistic method, respectively. Moreover, both the total memory and fabric bandwidth of our architecture implemented on crossbar could be lowered to only $2NR$, where R is line rate, counting read and write separately.

Index Terms — Switch-Memory-Switch, Distributed Shared Memory, Load-balanced, Emulating PIFO OQ

I. INTRODUCTION

As pointed out by numerous works, memory speed has long been the primary bottleneck in designing high-performance core routers. Although an $N \times N$ output-queued (OQ) switches could achieve an ideal performance with the unpractical speedup of N , input-queued (IQ) switches and combined input/output-queued (CIOQ) switches with no or less speedup are employed in practical design, succumbing to the memory bottleneck.

Switches that emulate OQ and overcome the memory bottleneck are always appealing, because OQ switches guarantee 100% throughput and the lowest average cell delay. Emulating OQ with push-in-first-out (PIFO) queues [1][3][4] is even more appealing, which could provide quality-of-service (QoS) guarantee through weighted fair scheduling policies such as Weighted Fair Queueing (WFQ).

However, it is difficult to emulate a PIFO OQ. The classical scheme to emulate PIFO OQ for CIOQ switches runs the switch fabric at the speedup¹ of two and access memory at the speedup of three[3], in which the reduction of bandwidth comes at the expense of complex and unpractical scheduling algorithm. Only recently, it has been proved that crosspoint-buffered CIOQ switch could emulate PIFO OQ with fabric

speedup of two, memory speedup of three, and N cells per crosspoint, $N/2$ fixed relative delay through a relatively less complex distributed scheduling algorithm [4].

Besides CIOQ switches, the Switch-Memory-Switch (SMS) architecture [2][5][6] shown in Fig.1 provides another way towards reducing memory bandwidth while providing performance guarantee. SMS architecture buffers packets in the M two-speedup¹ memories between two interconnects. The SMS router in [2][5][6] only emulates FCFS OQ. It computes the departure time before input cells are distributed into the central memories, and resolves memory access conflict in the first stage. A simple randomized parallel scheduling algorithm RiPSS and a pipelined version PRiPSS are proposed in [5][6], respectively, which could assign packets into $(2 + \epsilon)N$ memories in $O(\log^* N)$ matching rounds with high probability (w.h.p).

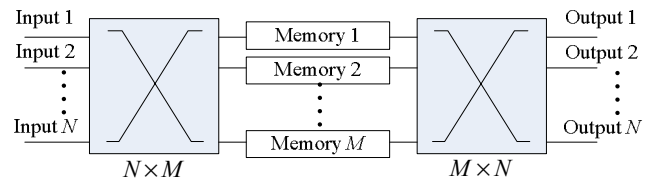


Fig. 1. The architecture of SMS switches

Almost concurrently with [2], S. Iyer *et al.* propose a distributed shared memory (DSM) switch [1], which could emulate PIFO OQ with a totally $4NR$ memory bandwidth, and $5NR$ crossbar bandwidth (complex algorithm) or $8NR$ crossbar bandwidth (simple algorithm), and $2N - 1$ relative delay. The basic idea of [1] is the same as SMS architecture, while in the DSM router the M memories are physically distributed on the N linecards with speedup of memories. The scheduling algorithm for FCFS OQ in [1] is based on bitmaps representing which memories are busy at each slot and need $O(N)$ operations per slot. However, bitmaps for emulating PIFO OQ are more complex than that of FCFS OQ. No practical solution for this problem has been found.

Although nearly optimal solutions [5][6] to implement SMS emulating FCFS OQ have been proposed, FCFS OQ itself is not good enough, since it unfairly delays light busy traffic and unfairly allocates bandwidth under inadmissible traffic.

This paper focuses on constructing SMS router emulating PIFO OQ, which could support QoS-guaranteed scheduling policies such as WFQ.

Not the same with the approach of [1], we don't attempt to resolve conflicts when receiving packets, but dispatch cells into central memory banks in a random manner. We resolve

¹For convenience of comparison, in this paper we always count read and write operations separately in the definition of memory or fabric speedup.

conflicts in the second stage, with predictable limited delay bound and reordering buffer.

The rest of this paper is organized as follows. Section II presents the baseline model of our SMS architecture and the Delay Queue method. Section III theoretically analyzes the cell loss probability resulted from our architecture. In Section IV, the design of pipeline is discussed and performance evaluation by simulation is presented. In Section V, implementation issues are discussed. In Section VI, we conclude this paper.

In this paper, we consider only the cells, i.e. fixed-length packets, because the Segmentation and Reassembly (SAR) modules should be employed on the input/output part of line-card.

II. BASELINE ARCHITECTURE OF OUR SMS ROUTER

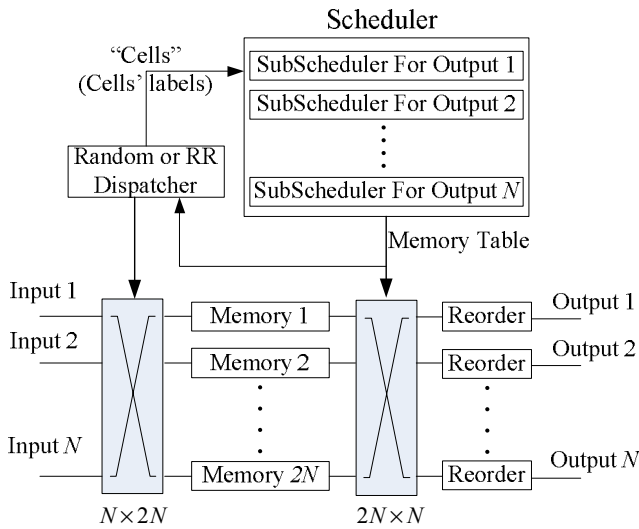


Fig.2. Baseline architecture of our SMS router

The baseline architecture of our $N \times N$ SMS router need $2N$ central memories, each of which need no speedup, i.e. performing **only one read or one write** operation in one time slot. In each slot, our scheduler should first determine which N (at most) of $2N$ memories send cells to the output ports, then the N (at most) newly arrived cells will be randomly dispatched to the other N (at least) memories not used for outputting in that slot. All of these scheduling should ensure that one memory should be selected by no more than one input port or output port.

The random dispatcher is discussed in Sec II.A.

The core idea of this paper lies in scheduler. The scheduler consists of N subschedulers corresponding to the N output ports. These subschedulers cooperate serially in pipeline to determine a transmitting schedule and resolve memory access conflicts. The design of pipeline is discussed in Sec II.B and further discussed in Sec IV. The design of the subscheduler is discussed in Sec II.C. The mis-sequencing caused by our scheduling method need a reordering buffer on each output. The length of the reordering buffer equals to the length of the Delay Queue of the subscheduler described in Sec II.C.

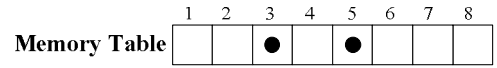
A. The Random Dispatcher --- the First Stage

As stated above, we do not attempt to resolve memory access conflicts of the second stage when receiving packets, different from the approach of previous works [1][2][5][6]. Thus, the first stage of our router is a random dispatcher.

Real-time random number generator is relatively not preferable. Practically, we recommend a Round-Robin (RR) dispatcher for the first stage. Let RR_{INP} and RR_{MEM} be the RR pointers of the input and the memory which point to the next in each slot. Then counted from the RR_{MEM} th memory, the $(i + RR_{INP}) \bmod N$ th available memory (unused by the second stage) is dispatched to receive cells from the i th input.

B. Scheduler Design--- Pipeline Design and Memory Table

The N subschedulers are pipelined and cooperate with the help of the Memory Table. A **Memory Table** is a bitmap that indicates which memories have been booked by other outputs. Then subschedulers serially fill in and pass the Memory Table to make a transferring schedule for one slot.



(Black point means the corresponding memory has been booked)

Fig.3. A Memory Table of a 4×4 Switch

The **Process Order** $\{S_i\}_{1 \leq i \leq N}$ of a Memory Table denotes the order of the subschedulers processing it, which should be a permutation of numbers from 1 to N and should be determined when generating the blank Memory Table. We wish that a new Process Order be randomly generated for each Memory Table. Specifics of practical pipeline design preserving randomness will be presented in Section IV.

The **Sequence Number** k of a Memory Table denotes how many schedulers have processed it, which means no more than k positions have been occupied.

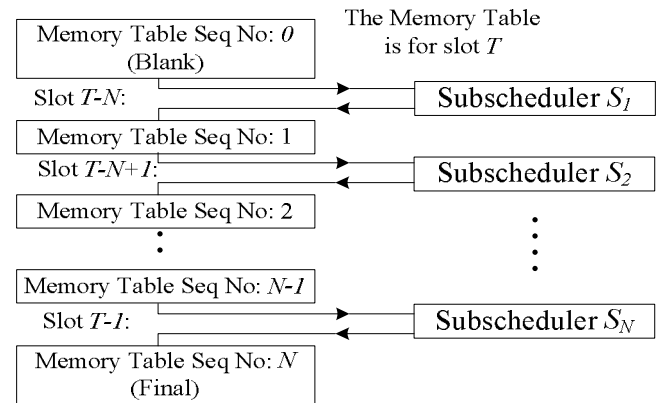


Fig.4. Scheduling Process (Passing and Filling in Memory Table)

The pipelined scheduling process with the Memory Table is exemplified in Fig.4.

C. SubScheduler Design--- PIFO Queue and Delay Queue

The critical issue of SMS router is to resolve memory access confliction and here we suppose the subscheduler stores all cells' scheduling information, i.e. the **Flow Label** (Input-Output-Pair) and the **Memory Number** of each cell,

while the whole content of the cell still stores in the memory. These scheduling information of cells only exist in schedulers, and are called “cell” with quotation marks for convenience. In Section V.C, how to transfer these “cells” within our SMS router is discussed.

In each subscheduler, there are two queues of the “cells”, the **PIFO queue** and the **Delay Queue**.

PIFO Queue [1][3][4] is a generalization of queuing policies including WFQ etc. In the PIFO queue, the relative ordering of the “cells” already in the queue does not change while a new “cell” could be pushed in ahead of or behind of “cells” in the queue. “Cells” could only depart from head of the queue. In Fig.5, we use the ellipsis to denote the “cells” except the first “cell”, because the “cells” in these positions might be pushed backwards by new “cell”. In practical implementation, PIFO queue could be implemented as multiple FIFO queues and this is not the focus of this paper.

The Delay Queue is the core of our algorithm to avoid memory access confliction. The fixed length of the Delay Queue is denoted as DL and some positions of it might be vacant. In each slot, the “cell” sent from the PIFO Queue is put into the last position of the Delay Queue, all remaining “cells” in the Delay Queue are shifted forward, and the “cell” in the first position, if not vacant, is discarded. In each slot, a matching process is executed to search for the first “cell” whose memory has not been booked in the Memory Table.

In Fig.5, the “cell” whose memory number is 1 is selected because the two “cells” preceding it conflict with the Memory Table in Fig.3. There are many positions vacant in the queue because the “cells” in these positions have been selected and popped previously. The “cell” in the first position is discarded, the reason for which should be that it conflicts with all Memory Tables after being put into the queue.

After the searching process, Fig.6 shows the modified Memory Table, which will be processed by the next subscheduler in its Process Order.

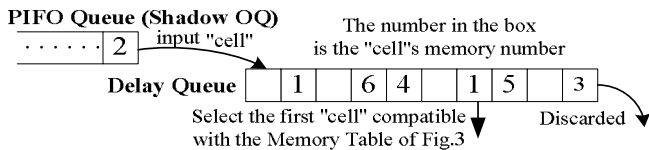


Fig.5. An Example of a matching process performed in Delay Queue

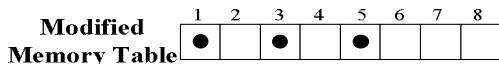


Fig.6. Modified Memory Table which will be passed to the next scheduler

To explain this Delay Queue method more clearly, the matching process of the next slot is exemplified in Fig.7.

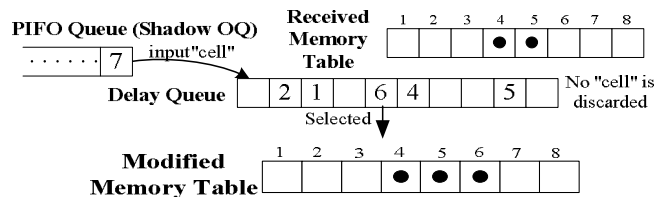


Fig.7. The matching process in the next slot

III. THEORETICAL ANALYSIS OF CELL LOSS PROBABILITY

Our theoretical analysis is based on these three independence assumptions:

Assumption i The distribution of the memory number of the input “cell” is i.i.d. through slots.

Assumption ii The distribution of the Sequence Number of the Memory Table received by one subscheduler is i.i.d. through slots.

Assumption iii If the amount of occupied positions of a Memory Table is given, the distribution of these positions is also i.i.d.

It should be noted that these assumptions are only approximations of reality. The assumption i and ii can not be strictly guaranteed in practical implementation and we could not give a proof of assumption iii. However, this simplified model and the analysis based on it provides us good insights into the performance of our design.

For convenience and the sake of length, here we only analyze an extreme situation that the input traffic load $\rho = 1$, which means each slot one “cell” is put into the Delay Queue. Note that, $\rho = 1$ serves only for bounding the cell loss possibility, and could not be simulated or be found in practice.

We also need these definitions:

Definition of α_j and β_j^i : If a “cell” in the i th position of the Delay Queue sees $j-1$ “cells” preceding it, let α_j be the probability of that none of these $j-1$ “cells” is selected in that slot, let β_j^i be the probability of that the “cell” is not selected in the next all i slots and eventually discarded.

Definition of CL : CL is the amount of “cells” in the Delay Queue. (Counted when the new “cell” is put in but the scheduling process has not been executed.)

DL is the length of the Delay Queue and has been defined in Section II.C.

Obviously, the probability of that the “cell” is selected in that slot is $\alpha_j - \alpha_{j+1}$. The probability of the “cell” in the first position (if exists) being discarded is $\beta_1^1 = \alpha_2$.

For convenience, if $i < j$, $\beta_j^i \triangleq 0$.

Based on the above assumptions and the symmetry of our architecture, we could easily get lemmas as follows:

Lemma 1 The memory number of the input “cell” is uniformly distributed from 1 to $2N$.

Lemma 2 If the received Memory Table has k positions occupied, the probability of any “cell” in the Delay Queue to be incompatible with the Memory Table, is $k/2N$.

Lemma 3 If the subscheduler receives random-sequence-number Memory Table according to Assumption ii,

$$\alpha_j \approx j^{-1} 2^{-j+1} \quad (1)$$

Proof of Lemma 3: If the subscheduler receives a Memory Table with Seq No. k , no more than k positions have been occupied, then,

$$\alpha_j \approx (k/2N)^{j-1} \quad (2)$$

(The equality holds if $\rho = 1$ and there is no “cell” loss.)

According to Assumption ii, consider receiving Memory Tables with random k :

$$\alpha_j \leq \frac{1}{N} \sum_{k=0}^{N-1} \left(\frac{k}{2N}\right)^{j-1} = \frac{1}{2^{j-1} N^{j-1}} \sum_{k=0}^{N-1} k^{j-1} \leq \frac{1}{2^{j-1} N^j} \frac{N^j}{j} = j^{-1} 2^{-j+1}$$

$$\text{Thus, } \beta_1^i = \alpha_2 \approx 1/4 \quad (3)$$

Theorem 1

$$\beta_j^i = (1 - \alpha_j) \beta_{j-1}^{i-1} + \alpha_{j+1} \beta_j^{i-1} \quad (4)$$

Proof: If the “cell” is lost at last, it comes to the $(i-1)$ th position in the next slot with either $j-2$ or $j-1$ “cells” preceding it, depending on whether one of “cells” before it is selected.

It is hard to get a closed form solution for (4). We find a nice approximation for (4) through numerical computation.

$$\beta_j^i \approx 1.2 \times 2^{-2(i-j-1)} \quad (5)$$

(error < 2%, when $(3 < j < i - 3, i < 200)$).

Theorem 2 Using CL as the state variable of the Delay Queue System, the successive states constitute a Markov Chain. The transition probabilities are:

$$P_{k,k+1} = P(CL(t+1) = k+1 | CL(t) = k) = \alpha_{k+1} \quad (6)$$

$$P_{k+1,k} = P(CL(t+1) = k | CL(t) = k+1) = \beta_k^{DL} \quad (7)$$

Let π_k denotes the stationary probabilities of CL , then:

$$\pi_{k+1} P_{k+1,k} = \pi_k P_{k,k+1} \quad (8)$$

The cell loss probability (CLP) of the whole system is:

$$P_{CLP}^{system} = \sum_{CL=1}^{DL} \pi_{CL} \beta_{CL-1}^{DL} \quad (9)$$

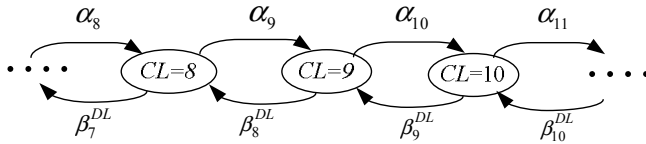


Fig.8. The State Transition of CL

Proof of (6): CL increases only when the k “cells” in the Delay Queue are not matched, because we suppose that in each slot one “cell” is put into the Delay Queue.

Proof of (7): Similarly, CL decreases only when “one” cell is discarded.

(8) and (9) is based on property of the Markov Chain.

Here we will get a concise approximate estimation of the total cell loss probability, use (2), (5) and (8):

$$\pi_{k+1} = \frac{\alpha_{k+1}}{\beta_k^{DL}} \pi_k = \frac{(k+1)^{-1} 2^{-k}}{1.2 \times 2^{-2(DL-k-1)}} \pi_k = \frac{2^{2DL-3k-2}}{(1.2)(k+1)} \pi_k \quad (10)$$

$$\text{Hence, let } k' = \arg \max_k \pi_k, \frac{2^{2DL-3k'-2}}{(1.2)(k'+1)} \approx 1. \quad (11)$$

$$\pi_{k'+j} = \pi_{k'} \prod_{i=1}^j \frac{2^{2DL-3i-3k'-2}}{(1.2)(i+k'+1)} \approx \pi_{k'} \prod_{i=1}^j 2^{-3i} = \pi_{k'} 2^{-\frac{3}{2}j(j+1)} \quad (12)$$

$$\text{Similarly, } \pi_{k'-j} \approx \pi_{k'} 2^{-\frac{3}{2}j(j+1)}. \quad (13)$$

Hence, $\pi_{k' \pm j}$ decays rapidly with j increases, far more rapid than the increase of $\beta_{k' \pm j}^{DL}$, it is reasonable to consider only $\pi_{k'-1}, \pi_{k'}, \pi_{k'+1}$ to compute P_{loss}^{system} , hence,

$$P_{CLP}^{system} \approx \pi_{k'-1} \times 2^{-3} \times 1.2 \times 2^{-2(DL-k')} + \pi_{k'} \times 1.2 \times 2^{-2(DL-k'-1)} + \pi_{k'+1} \times 2^{-3} \times 1.2 \times 2^{-2(DL-k'-2)} \approx 2^{-2DL+2k'+3} \pi_{k'} < 2^{-2DL+2k'+3}$$

An approximate solution to (10) is $k' \approx \frac{2}{3} DL - 2$. (14)

$$\text{Hence, } P_{CLP}^{system} \approx 2^{-2DL+2k'+3} \approx 2^{-2DL+\frac{4}{3}DL-4+3} \approx 2^{-\frac{2}{3}DL-1}. \quad (15)$$

IV. PIPELINE DESIGN AND SIMULATION RESULT

A. Baseline Pipeline Design

In theoretical analysis, we assume that the Sequence Number of Memory Table received by each subscheduler is random, which means the Process Order of a Memory Table is a randomly generated permutation. However, it is not preferable for a subscheduler to process two or more Memory Tables in one slot.

Here we design Process Orders ensuring each subscheduler processes exactly one Memory Table in one slot while preserving randomness to a certain degree, with the help of the Latin Square. A Latin Square of order N is an $N \times N$ matrix composed of numbers from 1 to N such that each number appears once in each row and once in each column. In other words, all rows and all columns are permutations of 1 to N . The following two matrixes are both Latin Squares.

$$Q_1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \\ 2 & 3 & 4 & 1 \end{bmatrix}, Q_2 = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 3 & 2 & 1 & 4 \\ 2 & 4 & 3 & 1 \\ 4 & 1 & 2 & 3 \end{bmatrix}$$

A Toeplitz Latin Square is a matrix defined by $R_{ij} = (j-i+1) \bmod N$, such as the Q_1 above. A Latin Square with randomness could be generated by randomly swapping rows and columns of a Toeplitz Latin Square.

Given a Latin Square $(L_{ij})_{1 \leq i, j \leq N}$, let $a_{ij} = m$, if $L_{mj} = i$. Then we could construct N Process Orders based on Latin Squares. At timeslot $kN+j$, a blank Memory Table for timeslot $(k+1)N+j$ is generated with the process order $A_j = (a_{ij})_{i=j, j+1, \dots, N, 1, \dots, j-1}$. This Process Order means this Memory Table will be processed by the subscheduler $a_{(j+l) \bmod N, j}$ in timeslot $kN+j+l$. In timeslot $kN+j+N-1$, this Memory Table is processed by the last subscheduler in its Process Order and the cells scheduled by it are transferred in timeslot $(k+1)N+j$. The Latin Square ensures in each slot each subscheduler processes exactly one Memory Table.

Hence, our algorithm needs a randomly generated Latin Square to construct a practical pipeline. Once the Latin Square is given, the process orders $(A_j)_{1 \leq j \leq N}$ are fixed. In our

simulation, it is set through randomly exchanging rows and columns of a Toeplitz Latin Square for $2N^2$ times.

B. Simulation Result of Basic Pipeline

We simulate our SMS router through 1×10^7 timeslots, with Weighted Round-Robin (WRR) scheduling policy[4], under both uniform Bernoulli traffic and nonuniform Bursty traffic (Avg. burst length=50). Our results are presented in Fig.8. WRR is also an example for PIFO queueing policy and is good enough for fixed-length cells scheduling.

Fig.9 demonstrates that our algorithm's performance is consistent with and bounded by the analytical results. It is reasonable to expect the cell loss probability decrease more if DL is set slightly more and $DL = 35$ is sufficient for usage.

The traffic pattern is not indicated in Fig.9 because the simulation results under different input traffic patterns are almost the same. It is only the traffic load ρ that affects the result. In this sense, there is no need to test with more traffic patterns. Besides, as expected, the lower input traffic load, the lower cell loss probability.

Fig.9 also shows that the simpler Round-Robin dispatching method in the first stage slightly degrades performance, compared with purely random dispatching.

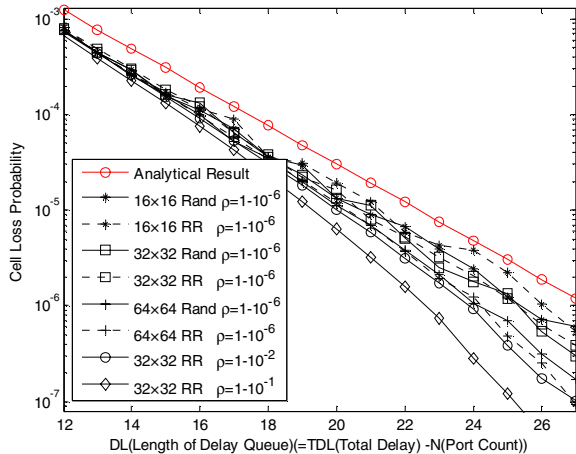


Fig.9 Cell Loss Probability as a function of the fixed delay introduced by Delay Queue

C. Improved Pipeline: Combination with the Delay Queue

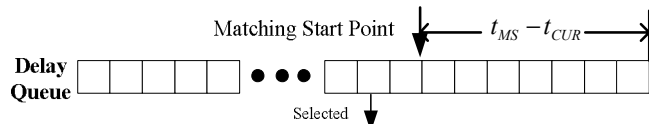


Fig.10 Matching Process with Consideration of Pipelines

There are two procedures causing delay in our system, the pipeline and the Delay Queue. Here we combine these two procedures to reduce total fixed delay as Fig.10 shows. Now the length of the Delay Queue equals the total fixed delay of the system. If receiving a Memory Table for slot t_{MS} in slot t_{CUR} , the search for matching starts at the $t_{MS} - t_{CUR}$ position of the Delay Queue. Thus, the two procedures are combined and the length of the Delay Queue equals the total delay.

Fig.11 shows the considerable reduction of the total delay through this improved method.

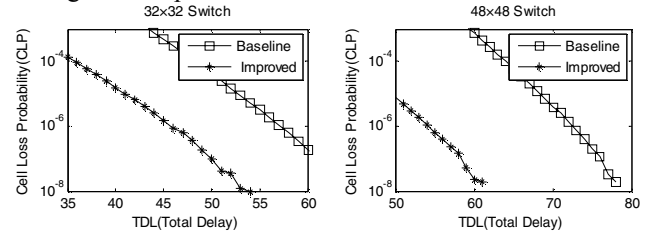


Fig.11 Reduction of the total fixed delay for 32×32 and 48×48 Switches

V. IMPLEMENTATION ISSUES

A. Delay Queue

In hardware implementation, the searching machine of the Delay Queue could be implemented as a priority encoder and a few MUX. Considering the relatively short length of the Delay Queue, the implementation cost of the Delay Queue is sustainable.

B. Memory Access

In the SMS emulating FCFS OQ [2], all cells' transferring slots are determined when arriving. Thus the memory bank need only operate as a FIFO and send cell according to its timestamp.

However, the sending sequence is not determined until the cell is sent, if emulating PIFO OQ. In each slot, each memory bank should not only know whether it is selected to send cells, but also know the cell's address if selected.

To solve this problem, we propose that each memory space is divided into N^2 FIFO queues, similar with the implementation of VOQ in IQ switch. In our Delay Queue method, only the first cell of the FIFO queue might be selected, because the searching engine searches for only the first "cell" compatible with the Memory Table. Thus, given the input-output-pair, it is easy to access the cell for transferring in the specified memory bank.

C. How to transfer "cells" in the switch

All our schemes to resolve memory access confliction and lower memory bandwidth are implemented in the output schedulers. As stated in Section II, the output schedulers should store all the "cells", i.e. the information for scheduling including the input number and the memory number of each cell. Thus, it is also an important implementation issue that how to transfer these "cells" in the switch economically.

First of all, one "cell" is relatively small and contains only $\log(2N) + \log(N^2)$ bits, where $2N$ stands for the number of memories and N^2 stands for the input-output pair. Thus, even simply employing a real shadow switch emulating PIFO OQ for "cells", the throughput of this shadow switch is much less than the main switch.

A better solution is that the output scheduler only caches the first few "cells" of each queue in each memory, which are enough for the fair scheduling algorithms to schedule between flows while considering the delay, while all "cells" are still

stored with the whole content of the cells in memory banks. The renewal of the cache of one “cell” queue in the subscheduler is executed when one “cell” of the queue is popped and the corresponding cell is sent to the output port. With this method, only small caches are needed in the output subschedulers.¹

D. Single-Crossbar-Based SMS router

In practical implementation, similar with the DSM in [1], the central memories banks could be physically distributed in the line-cards as shown in Fig.12. The match graph is a bipartite graph whose vertex’s degree is no more than two. It could be decomposed into two permutations, as shown below. Thus, the crossbar needs to operate at speedup of only two, while the crossbar speedup of four is needed if not considering decomposition of bipartite graph.

From this point of view, our SMS router could be implemented in single-crossbar architecture with the crossbar and the N memories running at speedup of two. Hence the total memory bandwidth and fabric bandwidth² are both $2NR$, where R is the line-speed. Note that, $2NR$ memory bandwidth is the lower limit of any router, among which at least NR is needed for read and at least NR is needed for write.

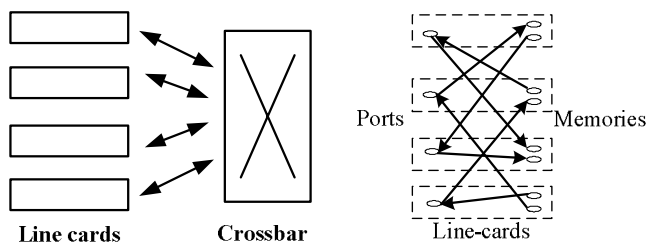


Fig.12. Single-Crossbar-Based SMS router and the Match Graph

Direct decomposition of a two-degree-bipartite is $O(N)$, which is a little complex although each step is very simple. We don’t discuss the direct decomposition algorithm here, because considering the dispatch of new-incoming cells, we propose a combined and simplified decomposition procedure.

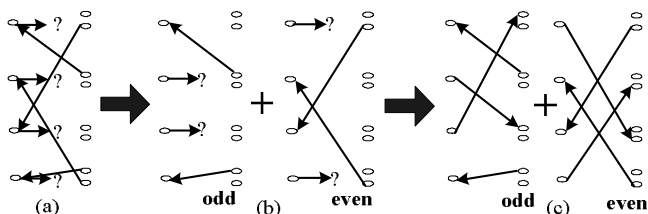


Fig.13. Decomposition of the Match Graph and Dispatch Of New Cells

As described before and exemplified by Fig.13(a), the memories outputting cells are determined by the scheduler and the dispatcher should determine the memory banks storing the incoming cells. The output of cells from memories could be easily decomposed into two steps, depending on the parity of the memories which should be read out for the output, as Fig.13(b) shows. Then the ports and memories unselected for output in each step are matched for dispatching cells into the memories with some Round-Robin or other random mechanism, as Fig.13(c) shows.³

VI. CONCLUSION

In this paper, we achieve the goal of emulating PIFO OQ in an SMS router of $2N$ no-speedup memories, at the cost of accidental cell loss probability (P_{CLP}), fixed cell delay

$$(TDL = N + DL, DL < -\frac{3}{2} \log_2 P_{CLP} - 1), \text{ and additional}$$

scheme for transferring cells’ scheduling information (“cells”).

Our SMS architecture could be implemented in a single-crossbar scheme with the fabric and N memories all running at speedup of two, counting the read and write separately. Hence, the total memory bandwidth and the total fabric bandwidth² of the single-crossbar implementation scheme are both $2NR$, where R is the line speed.

To achieve the goal, we use N subschedulers and an N -level pipeline. Subschedulers in the pipeline serially fill in the Memory Table to resolve memory access conflicts. The Delay Queue of each subscheduler selects the first “cell” compatible with the Memory Table in each slot. This method ensures cells scheduled by the PIFO queue being transferred within relative fixed delay bound with high probability, independent of input traffic pattern. The approximate bound of cell loss probability is mathematically analyzed and validated through simulation. We also present a matching procedure combining the Delay Queue and the pipeline to reduce the fixed delay.

Compared with previous works [1][2][5][6], our works not only provides practical support to the PIFO scheduling policy—a generalization of fair queueing algorithms, but also greatly reduces the requirement for the memory bandwidth and fabric bandwidth, achieving the minimum memory bandwidth bound for any router.

Though far from perfection, we believe our works are encouraging. In future, we will be interested in reducing the fixed delay with some parallel request-grant process, and also in improving and analyzing the specifics of our schemes.

REFERENCE

- [1] S. Iyer, R. Zhang, and N. McKeown. “Routers with a Single Stage of Buffering.” In ACM SIGCOMM 2002
- [2] A. Prakash, S. Sharif, and A. Aziz. “An $O(\log^2 n)$ algorithm for output queueing.” In IEEE INFOCOM 2002.
- [3] Shang-Tse Chuang, Goel, A., McKeown, N., Prabhakar, B. “Matching output queueing with a combined input/output-queued switch”. In IEEE JSAC 1999
- [4] Shang-Tse Chuang, Sundar Iyer, Nick McKeown. “Practical Algorithms for Performance Guarantees in Buffered Crossbars.” In IEEE INFOCOM 2005
- [5] A. Aziz, A. Prakash, and V. Ramachandran. “A near optimal scheduler for switch-memory-switch routers.” In ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, CA, June 2003.
- [6] A. Aziz, A. Prakash, and V. Ramachandran. “Randomized Parallel Schedulers for Switch-Memory-Switch Routers: Analysis and Numerical Studies”. In IEEE INFOCOM 2004.

¹For the sake of length, the specifics of this scheme is only briefly described here.

²For convenience of comparison, in this paper we count read and write operations separately, when calculating the memory bandwidth, similar with [1].

³For the sake of length, the decomposition algorithm is only briefly described here.