DiaCTC (*N*): An Improved Contention-Tolerant Crossbar Switch

Guannan Qu, Zhiyi Fang, Jianfei Zhang, Lin Chen, Haiqin Qu
Jilin University
China
{gnqu, fangzy}@jlu.edu.cn, jfzhang.jlu@hotmail.com, starle1234@yahoo.cn

ABSTRACT: *We recently proposed an innovative agile crossbar switch architecture called Contention-Tolerant Crossbar Switch, denoted as CTC (N), where N is the number of input/output ports. CTC (N) can tolerate output contentions instead of resolving them by complex hardware, which makes CTC (N) simpler and more scalable than conventional crossbar switches. In this paper, we analyze the main factors that influence the performance of CTC (N) and present an improved contention tolerant switch architecture - Diagonalized Contention-Tolerant Crossbar Switch, denoted as DiaCTC (N). DiaCTC (N) maintains all good features of CTC (N), including fully distributed cell scheduling and low complexity. Simulation results show that, without additional cost, the performance of DiaCTC (N) is significantly better than CTC (N).*

Keywords: Crossbar Switch, Agile Crossbar Switch, Diagonalized Contention-Tolerant Crossbar Switch

Received: 29 October 2013, Revised 28 November 2013, Accepted 4 December 2013

1. Introduction

Crossbar is widely used in high-speed Internet switches and routers for its simplicity and non-blockingness. To simplify scheduling operations, variable size packets are segmented at input ports into fixed-size cells and reassembled at output ports.

According to where packets (cells) are buffered, there are four basic types of crossbar switches, namely output queued (OQ), input queued (IQ), combined input and output queued (CIOQ), and crossbar with crosspoint buffered switches. In an OQ switch, cells arriving at input ports are forwarded to their destination output ports immediately and buffered in output queues. Without delay in input ports and switch fabric, OQ switches are powerful in terms of providing quality of services (QoS). Thus, theoretical studies on QoS guarantee are based on output queued switches [1]. Since an OQ switch requires memory speedup N, where N is the number of input/output ports of the switch, such QoS results are impractical.

The memory of IQ switches operates at the same speed as the external link rate and cells are queued in input ports. To avoid head-of-line (HOL) blocking problem, input buffers are arranged as virtual output queues (VOQs). Since it is hard to ensure QoS on IQ switches, CIOQ switches have been proposed as a trade-off design of OQ and IQ switches. In a CIOQ switch, the memory speed is S times faster than the link rate, where S is in the range of $1 < S < N$, and cells are buffered in both input ports and output ports. It was shown that a variety of quality of services are possible using CIOQ switches with a small constant S.

The performance of an IQ or CIOQ switch depends on scheduling algorithm, which selects contention-free cells and

configures I/Q connections for switching cells in each time slot. For IQ switches, many scheduling algorithms based on maximum matching have been investigated (e.g. [2] [3]). These scheduling algorithms provide optimal performance. Because the time complexity for finding maximum (size or weight) matchings is too high for practical use, heuristic algorithms for finding maximal matchings were considered instead (e.g. [4]-[8]). For a switch with N input ports and N output ports, such schedulers require $2N$ N-to-1 arbiters working in multiple Request-Grant-Accept (RGA) or Request-Grant (RG) iterations (which involve global information exchange) to obtain a maximal matching between inputs and outputs. Though implemented in hardware, these schedulers are considered too slow with too high cost for high-speed networks. The scheduling problem of CIOQ switches has also been considered. It was shown in [9] that, using an impractically complex scheduler, which implements the Stable Marriage Matching (SMM) algorithm [10], a CIOQ crossbar switch with a speedup of two in the switch fabric and memory can emulate an output queued (OQ) switch. This result is only theoretically important, because the SMM problem has time complexity $O(N^2)$.

To reduce scheduling complexity, crossbar switch with crosspoint buffers was proposed, which is also called buffered crossbar switch. Coordinating with input queues, crosspoint buffers decouple scheduling operations into two phases in each time slot. In the first phase, each input port selects a cell to place into a crosspoint buffer in its corresponding row, and in the second phase, each output port selects a crosspoint in its corresponding column to take a cell from. Input (resp. output) ports operate independently and in parallel in the first (resp. second) phase, eliminating a single centralized scheduler. Compared to unbuffered crossbars, the scheduling algorithms of buffered crossbars are much simpler. Considerable amount of work, e.g. [11]-[18], has been done on buffered crossbar with and without internal speedup. However, N^2 crosspoint buffers take a large chip area, which severely restricts the scalability of buffered crossbar switches.

In summary, conventional crossbar switches, including crossbar with crosspoint buffers, require complex hardware to resolve output contentions. We recently proposed a new switch architecture called *contention-tolerant crossbar* switch, denoted by *CTC* (*N*), where *N* is the number of input/output ports [19]. *CTC* (*N*) tolerates output conflicts using a reconfigurable bus in each output column of the fabric. In this way, controllers distributed in input ports are able to operate independently and in parallel. This feature reduces the scheduling complexity and wire complexity, and makes *CTC* (*N*) more scalable than conventional crossbar switches. *CTC* (*N*) opens a new perspective on designing switches. This paper focuses on further discussion on *CTC* (*N*), and presents an improved contention-tolerant switch architecture called *diagonalized contention-tolerant crossbar* switch, denoted as *DiaCTC* (*N*). Simulation results show that, with *staggered polling* (SP) scheduling algorithms [20], *DiaCTC* (*N*) significantly enhances the performance with the same low cost of *CTC* (*N*).

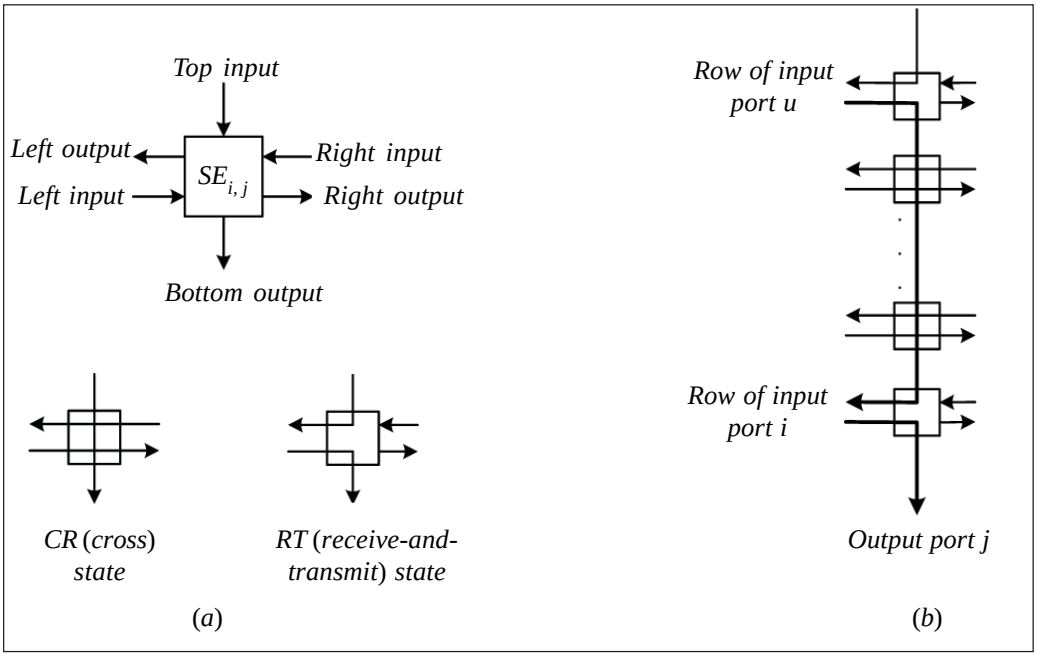


Figure 1. (a) A crosspoint SE and its two states; (b) Each output line of *CTC* (*N*) is a reconfigurable bus

2. Throughput Bottleneck of *CTC* (*N*)

In our previous work, we presented the *CTC* (*N*) architecture. Similar to conventional crossbar, the fabric of *CTC* (*N*) is comprised of N^2 crosspoints (Switching Element, SE) arranged as an $N \times N$ array. Each SE has three inputs, three outputs and two states, as shown in Figure 1 (a). Each input port i is equipped with a scheduler S_i. In one time slot, if input port i ($0 \le i \le N-1$) wants to transmit a cell to an output port j ($0 \le j \le N-1$), S_i sets the state of corresponding $SE_{i,j}$ to receive-and-transmit (RT) state. The remaining SEs in the same row be kept in cross (CR) state. If more than one input ports set their SEs as RT in the same output line (column), the output line is configured as a pipeline, as shown in Figure 1 (b). Cells transmitted from upstream input ports will be intercepted and buffered in downstream input ports. In this way, output contentions are tolerated in *CTC* (*N*). Buffer in each input ports can be arranged as single FIFO queue or Virtual Output Queues according to queueing management policies, which contains cells both from outside of switch and from upstream input ports (if exist).

In [19], we theoretically proved that the throughput of *CTC* (*N*) with single FIFO in each input ports and without speedup is bounded by 63%. To improve the performance of *CTC* (*N*), we proposed *staggered polling scheduling algorithm scheme* (SP for short) [20]. In order to ease scheduling operations, buffer in input port i is arranged as N VOQs denoted by $VOQ_{i,j}$. S_i, the scheduler in input port i, maintains two sub-schedulers, i.e. the *primary sub-scheduler* PS_i and the *secondary sub-scheduler* SS_i, as shown in Figure 2. PS_i generates a unique number $c_i(t)$, $0 \le c_i(t) \le N-1$, in time slot t. That is, $c_i(t) = c_i(t)$ if $i = i$. At time t, if $VOQ_{i,ci}(t)$ is not empty, the cell selected by PS_i is the HOL cell of $VOQ_{i,ci}(t)$, denoted by $PS_i(t) = VOQ_{i,ci}(t)$. Otherwise, PS_i returns null, denoted by $PS_i(t) = null$. SS_i maintains a set $L_i(t)$ of non-empty VOQ indices (i.e. $L_i(t) = \{k \mid VOQ_{i,k}$ is not empty in time slot $t\}$), which is updated in every time slot. SS_i chooses one from $L_i(t)$ according to some algorithms as its scheduling result. Random pattern is one of representative secondary scheduling algorithm, i.e. SS_i picks one index of VOQ from $L_i(t)$ randomly. We use $q_i(t)$ to denote the index number of the VOQ chosen by SS_i in time slot t, i.e. $SS_i(t) = VOQ_{i,qi}(t)$. If the $L_i(t)$ is empty, $SS_i(t) = null$.

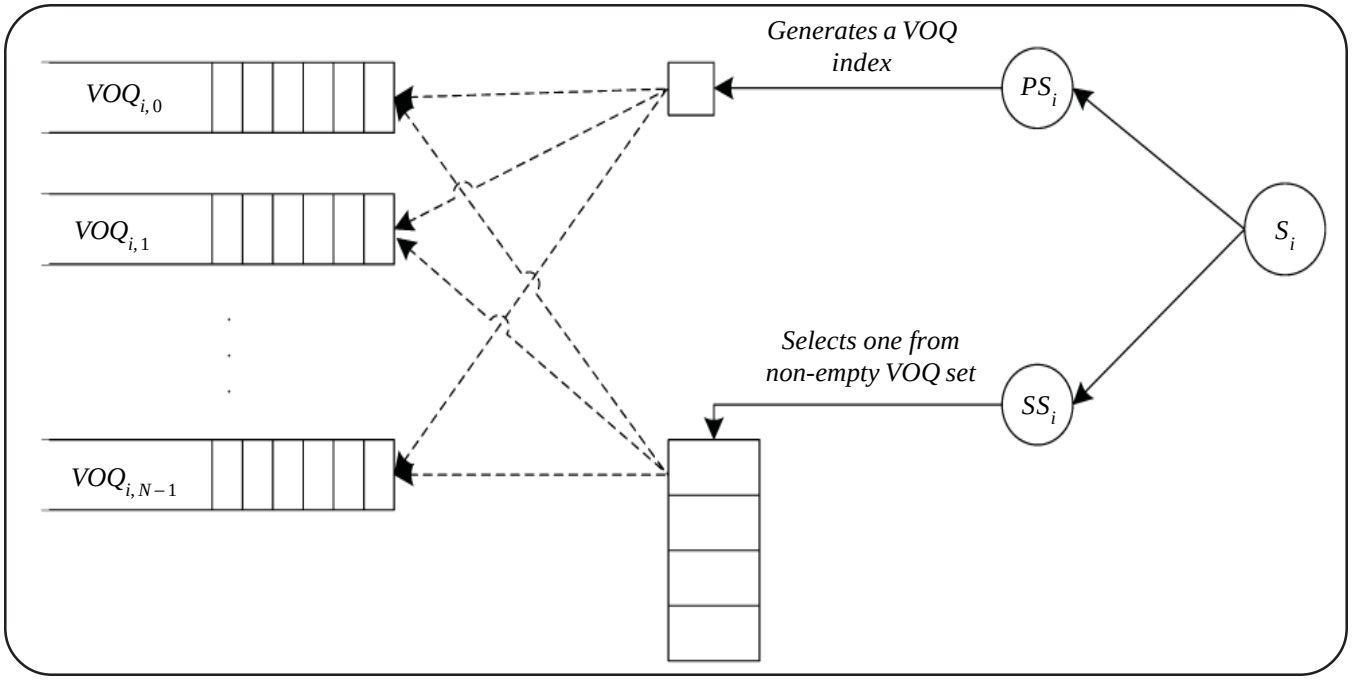


Figure 2. Scheduling process in input port i

The two sub-schedulers operate in parallel. S_i chooses the scheduling result of PS_i first. If $PS_i(t) = null$, $S_i(t) = VOQ_{i,ci}(t)$, which means $VOQ_{i,ci}(t)$ will be serviced at time slot t. If $PS_i(t) = null$, S_i turns to consider the scheduler result of SS_i. If both PS_i and SS_i return null, $S_i(t) = null$, which means no VOQ will be serviced at time slot t.

In [20], we evaluated the performance of SP with several example scheduling algorithms. Simulation results showed that with

zero knowledge of other input ports, the fully distributed schedulers "*smartly*" cooperated with each other and attained high performance. However, due to the intrinsic feature of *CTC* (*N*), it is hard for the throughput of *CTC*(*N*) to achieve 100%.

In what follows we analyze the main factors that affect the performance of *CTC* (*N*). For easy analysis, let us consider the case with heavy offered load. Since each input port transmits one cells during one time slots, input port *i* can be modeled as a queue, denoted as Q_i, which contains cells from both outside and upstream and with *N* output destinations. *CTC* (*N*) can be modeled as a queueing network as shown in Figure 3, where $a_i^o = \lambda$ is the arrival rate of Q_i from outside, and a_i^u is the arrival rate of Q_i from upstream input ports. r_i is the service rate of Q_i. For switch without internal speedup and under heavy traffic, we have $r_i = 1$.

For Q_i, the aggregate arrival rate a_i is:

$$a_i = a_i^o + a_i^u = \lambda + a_i^u \tag{1}$$

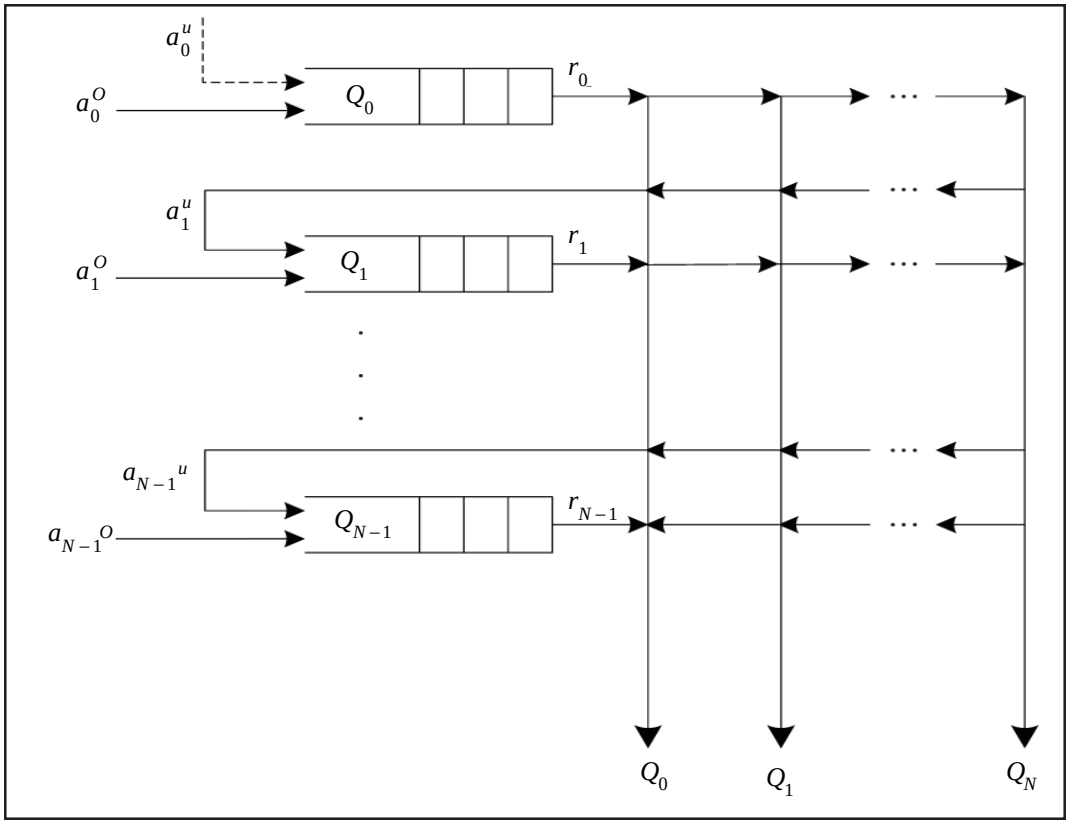


Figure 3. Queueing model of *CTC* (*N*) with single FIFO queue in each input port

When $a_i > r_i$, Q_i is overloaded and the throughput of *CTC* (*N*) fails to achieve 100%.

Let $a_{i,j}^u$ be the arrival rate of Q_i from upstream input ports with output port *j* as its destination. We have

$$u_i = \begin{cases} 0 & \text{if } i = 0 \\ \sum_{j=0}^{N-1} a_{i,j}^u & \text{if } 0 < i \le N-1 \end{cases} \tag{2}$$

From the analysis in [21], we know that

$$a_{i,j}^u > a_{k,j}^u \tag{3}$$

for $i > k$ and $0 \le j \le N-1$. It implies,

$$a_i^u > d_k^u \tag{4}$$

for $i > k$. Clearly, the aggregate upstream traffic increases as input port i increments. In the worst case with $\lambda = 1$, from Equation (1) and (2), we have:

$$a_i = \begin{cases} 1 & \text{if } i = 0 \\ 1 + \sum_{j=0}^{N-1} a_{i,j}^{u} & \text{if } 0 < i \le N-1 \end{cases}$$

Obviously, even Q_1 is overloaded for $a_1 > r_1$. With the same service rate and multiple times heavier arrivals, downstream input ports suffer from more severe overload.

SP scheduling algorithm scheme was designed for reducing upstream arrivals by diminishing interceptions. With staggered polling pattern, the primary sub-schedulers select cells to form a conflict-free I/O matching. In order to maximize the utilization of input ports, the second sub-schedulers select cells arbitrarily, which may cause conflicts and interceptions. Simulation results in [20] showed that SP algorithms successfully enhance the throughput. However, the structure of *CTC* (*N*) dictates that Equations (3) and (4) still hold with SP algorithms. Unbalanced upstream arrivals lead to overloading traffic for downstream input ports. It explains the phenomenons in [20] that the throughput began to go down when offered load $\lambda = 0.5$, where the input port $N - 1$ who had heaviest upstream traffic started to be overloaded.

3. Diagonalized Contention-tolerant Crossbar Switch Architecture

In order to improve the throughput of *CTC* (*N*), we introduce an improved *CTC* (*N*) architecture called *Diagonalized Contention-Tolerant Crossbar Switch*, denoted as *DiaCTC* (*N*). *DiaCTC* (*N*) is exactly the same in all aspects of *CTC* (*N*), except the connections in each SE column.

In *CTC* (*N*), SEs in each output column form a unidirectional alignment. $SE_{i,j}$ is an upstream node of $SE_{i,j}$ in output column j, where $0 \le i < i \le N-1$ and $0 \le j \le N-1$. Therefore, input port 0 is the top input for any output destination and it only has traffic from outside. A cell transmitted out from input port 0 could be intercepted by N –1 downstream input ports. Input port $N-1$ is the bottom input for any output destination. Cells buffered in input port $N-1$ are from outside and $N-1$ possible upstream input ports by N possible output columns. While in *DiaCTC* (*N*), consider output column j, $0 \le j \le N-1$. SEs in output column j are classified into the following three classes:

• *Head SE:* $SE_{i,j}$ is the head *SE* of output column j when $i = j$. The associated *Ii* is the top input for output destination O_j. Cell transmitted from I_i to O_j possibly is intercepted by *Id*, $d = i + k \bmod N$ and $1 \le k \le N-1$.

• *Tail SE:* $SE_{i,j}$ is the tail *SE* when $i = j - 1 \bmod N$. The associated I_i is the bottom input for output destination O_j. Cells transmitted from I_i to O_j arrives at O_j without being intercepted.

• *Internal SE:* $SE_{i,j}$ is an Internal *SE* when $i = j + k \bmod N$, $1 \le k \le N-2$. I_i has upstream inputs and downstream inputs for output destination O_i. A cell transmitted from I_i to O_j could be intercepted by its downstream input I_d, i.e. $d = i + k \bmod N$ and $1 \le k \le (j-1-i) \bmod N$.

Let C_i be the aggregation of cells which might be buffered in I_i. $c_{s,d}$ is the cell which originally arrived at I_s from outside with O_d as its destination. $c_{s,d} \in C_i$ and it satisfies following condition:

$$s = \begin{cases} i & \text{if } d = i \\ (i-k) \bmod N, 0 \le k \le (i-d) \bmod N & \text{otherwise} \end{cases}$$

Figure 4 (b) shows a *DiaCTC* (4), and its counterpart *CTC* (4) is shown in Figure 4 (a) with SEs serving as Heads and Tails being labeled. In *CTC* (*N*), all SEs in the top row are Head SEs and SEs in the bottom row are Tail SEs, while in *DiaCTC* (*N*) there is exactly one Head SE and exactly one Tail SE in each row.

Compared with *CTC* (*N*), *DiaCTC* (*N*) balances the aggregate upstream traffics over all input ports without additional hardware cost, and Equations (3) and (4) don't hold. In the next section, we show that better performance will be achieved by this simple,

but meaningful, modification of *CTC* (*N*).

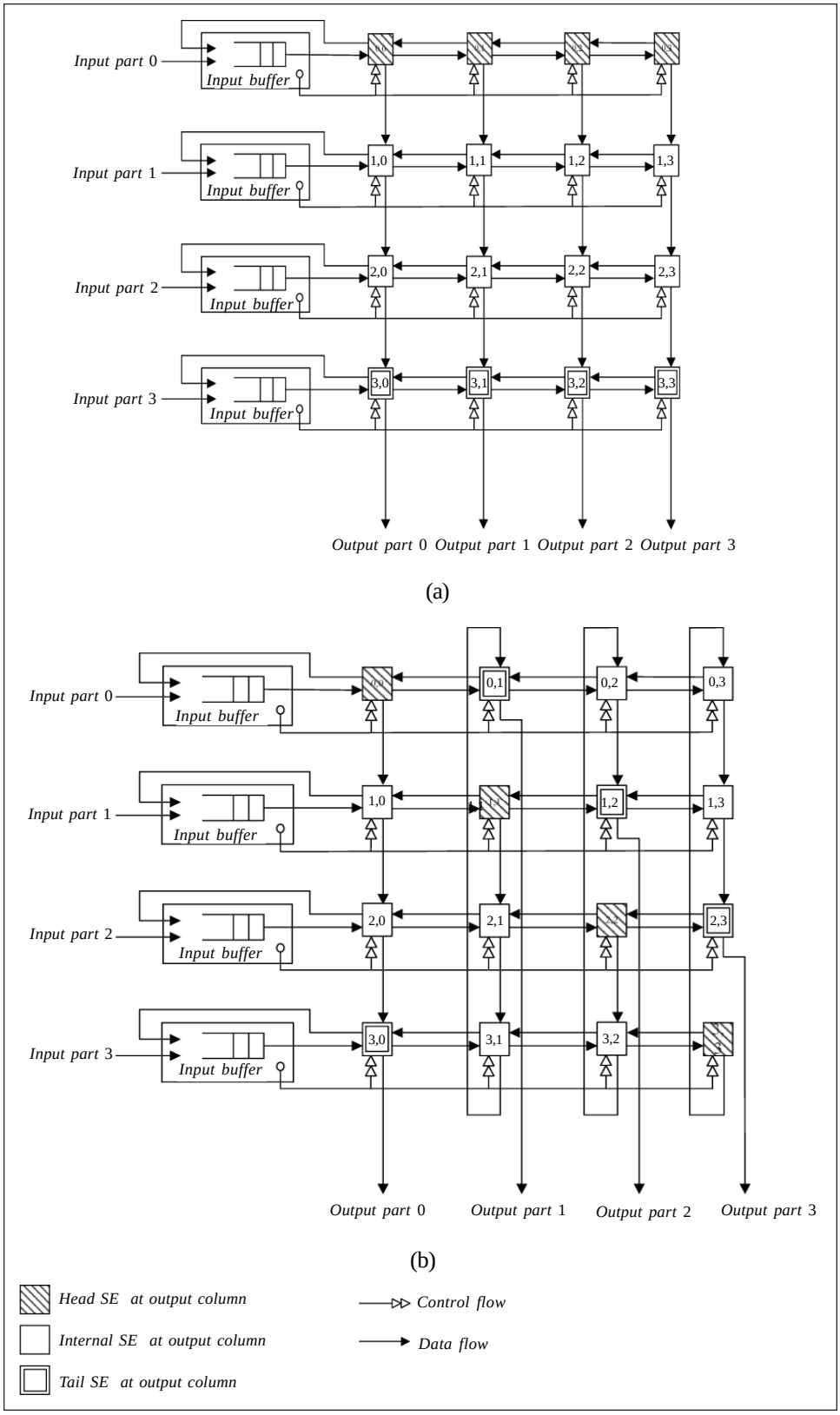


Figure 4. (a) *CTC* (4) architecture; (b) *DiaCTC* (4) architecture

4. Performance Evaluation

The performance of *DiaCTC* (32) and *CTC* (32) with Staggered Polling (SP) algorithm scheme are compared in terms of mean cell delay under uniform traffic and nonuniform traffic by simulations. Random pattern is chosen as an example secondary scheduling algorithm in SP algorithm scheme. We also consider the well-known iSLIP method of one iteration for 32 × 32 conventional crossbar switch for the reason that only one iteration may be performed in each time slot in cell switching at the line speed.

4.1 Uniform Traffic

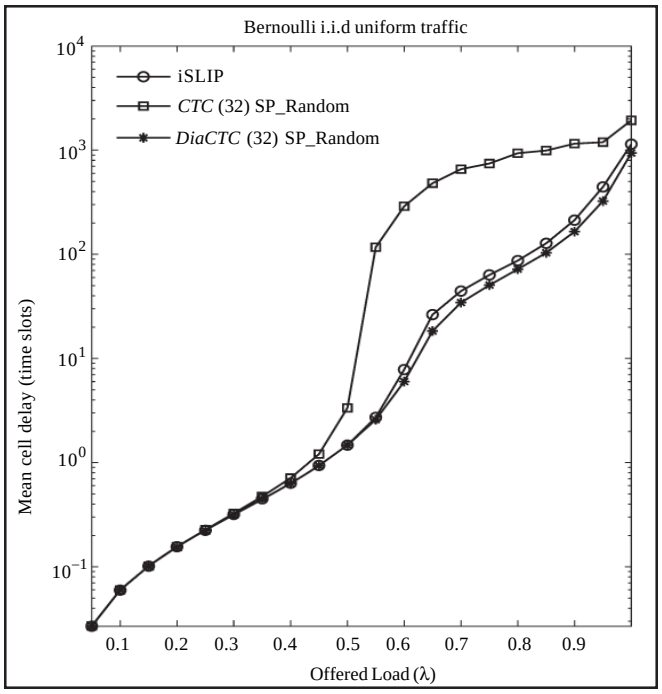


Figure 5. Mean cell delay under Bernoulli i.i.d. uniform traffic

For uniform traffic, the traffic distributed over all output destinations uniformly with Bernoulli arrivals and Bursty arrivals. Under Bernoulli uniform traffic, *DiaCTC* (32) SP Random performs the same performance with iSLP when offered load $\lambda \leq 0.55$, and has smaller mean cell delay than iSLIP when $0.55 \leq \lambda \leq 1$, as shown in Figure 5. Obviously, compared to the performance of *CTC* (32) SP Random, *DiaCTC* (32) SP Random has remarkable improvement.

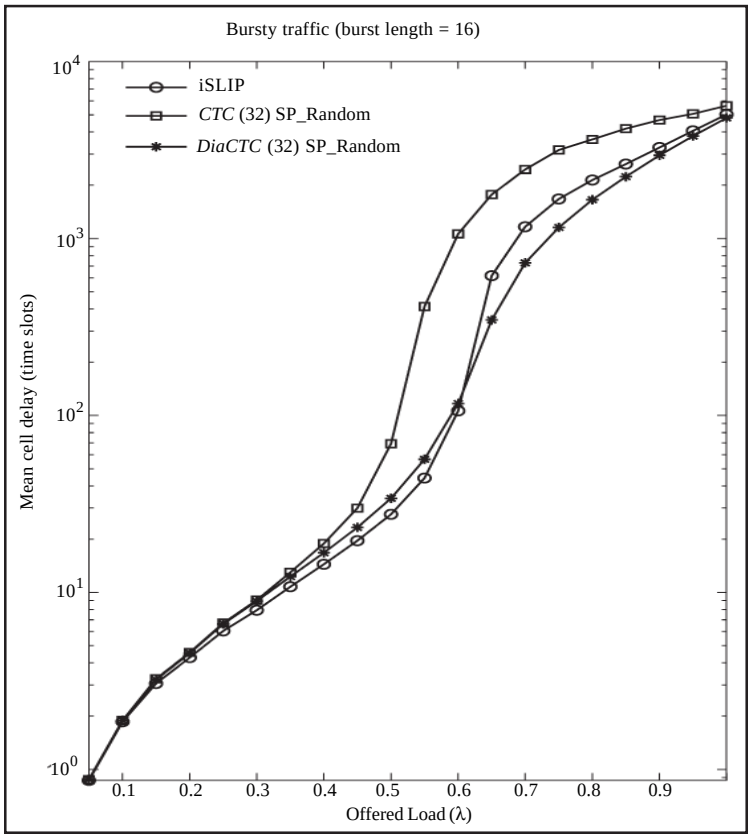
Figure 6 illustrates the performance under bursty arrivals with burst length are 16, 32 and 64. The performance of *DiaCTC* (*N*) SP Random, *CTC* (*N*) SP Random and iSLIP decrease slightly with increasing burst length at the same offered load λ, and the performance decline of iSLIP is more evident than the other two. It implies that *CTC* (*N*) and *DiaCTC* (*N*) switches are affected less than iSLIP by burst length. The iSLIP outperforms *DiaCTC* (32) SP Random and *CTC* (32) SP Random in these three graphs when $\lambda \leq 0.6$, however, *DiaCTC* (32) SP Random shows the best performance when $0.6 < \lambda \leq 1$. *CTC* (32) SP Random has the similar performance with *DiaCTC* (32) SP Random and iSLIP at $\lambda = 1$ with minor difference.

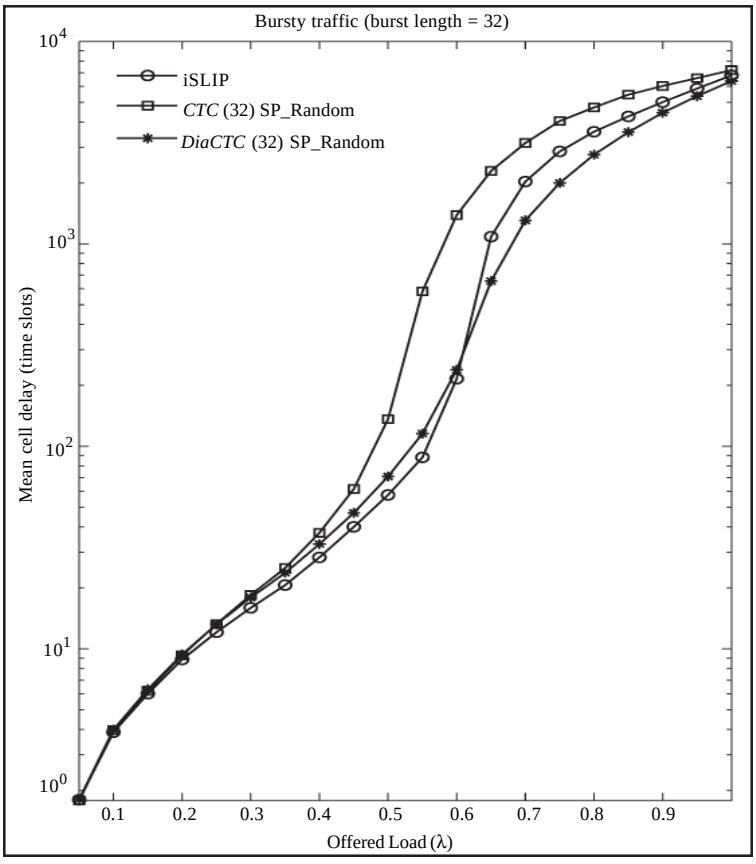
4.2 Non-uniform traffic

We chose three schemes from several nonuniform traffic models: Asymmetric [22], Chang's [23], and Diagonal [24]. Let $\lambda_{i,j}$ be the offered load arriving at input port *i* and forwarding to output port *j*. Asymmetric traffic model is defined as

$$\lambda_{i,(i+j)} \bmod N = \lambda a_j$$

where $a_0 = 0$, $a_1 = (r-1)/(r^N - 1)$, $a_j = a_1 r^{j-1} \forall\, j \neq 0$, and $\lambda_{i,j}/\lambda_{(i+1)} \bmod N$, $j = r$, $\forall i \neq j$, $(i+1) \bmod N = j$, $r = \lambda_{min}/\lambda_{max} = aN{-}1/a1 = r - 1/(N-2)$. Chang's traffic model is defined as


Bursty traffic (burst length = 16)
iSLIP
CTC (32) SP_Random
DiaCTC (32) SP_Random
Mean cell delay (time slots)
Offered Load (λ)


Bursty traffic (burst length = 32)
iSLIP
CTC (32) SP_Random
DiaCTC (32) SP_Random
Mean cell delay (time slots)
Offered Load (λ)

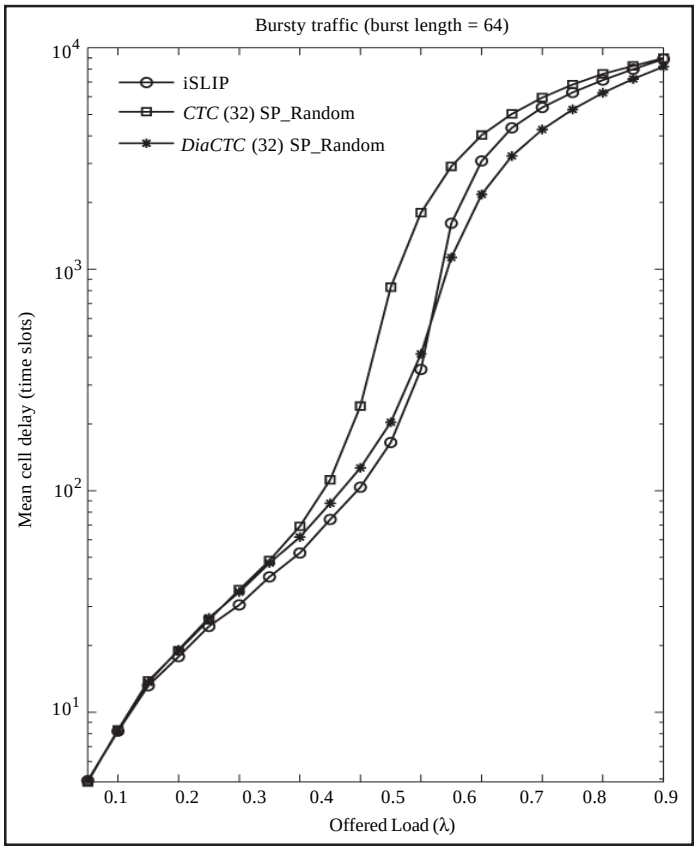


Figure 6. Mean cell delay under bursty traffic with different burst length

$$\lambda_{i,j} = \begin{cases} 0 & \text{if } i = j; \\ \dfrac{\lambda}{N-1} & \text{if } j = i+1; \end{cases}$$

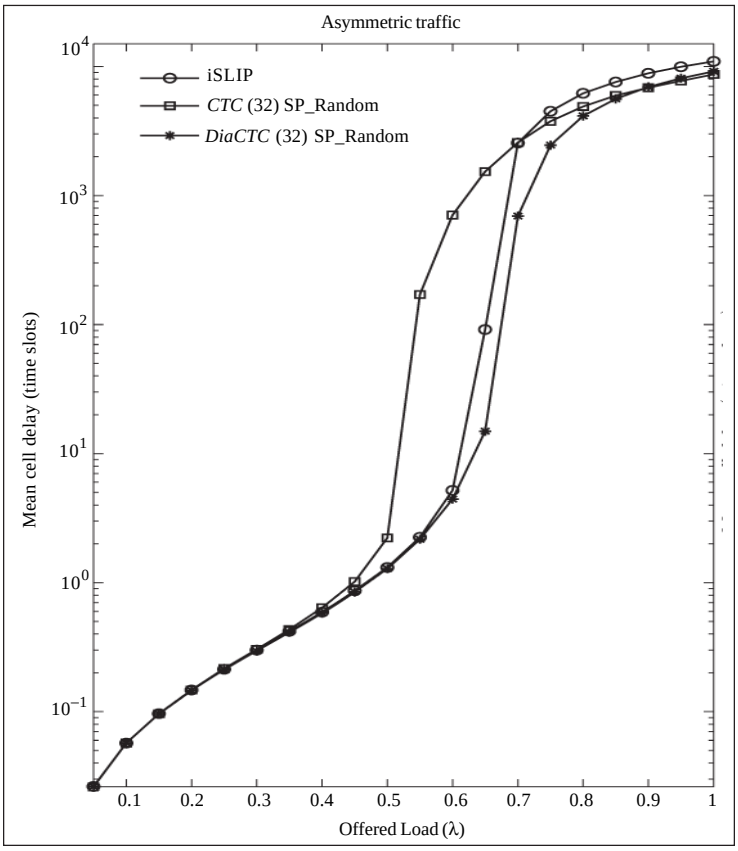
Diagonal traffic model has very skewed loading which defined as

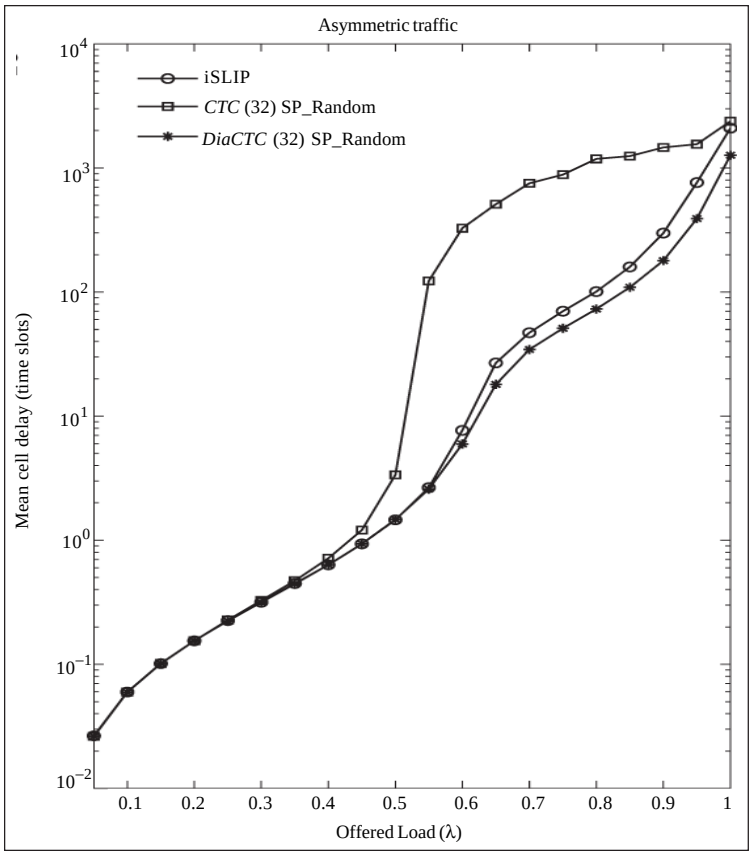
$$\lambda_{i,j} = \begin{cases} \dfrac{2\lambda}{3} & \text{if } i = j; \\ \dfrac{\lambda}{3} & \text{if } j = i+1; \\ 0 & \textit{otherwise} \end{cases}$$

The performance of *DiaCTC* (32) and *CTC* (32) with SP scheduling algorithm and iSLIP are given in figure 7. *DiaCTC* (32) SP Random shows the best performance with both asymmetric and Chang's arrivals. *CTC* (32) performs similar with *DiaCTC* (32) and is better than iSLIP when $0.8 \le \lambda \le 1$ under asymmetric traffic.

Under diagonal traffic, the mean cell delay of *CTC* (32) SP Random increases sharply when $0.65 \le \lambda \le 0.75$, and goes up slightly with $0.75 \le \lambda \le 0.8$. The delay of *DiaCTC* (32) SP Random and iSLIP rise smoothly with increasing offered load, and iSLIP outperforms *DiaCTC* (32) SP Random. *DiaCTC* (*N*) tends to scatter the traffic over all of the VOQs in one input by intercepting cells from other inputs. Thus SP scheduling algorithm operates well and the switch achieves high performance. It is good for the situation with heavy and more balanced traffic load, such as Bernoulli traffic, Bursty traffic, Asymmetric non-uniform traffic and so on. However, for diagonal traffic which only has cells forwarding to two output destinations in each input ports, load balancing process in *DiaCTC* (*N*) leads to unexpected delay. Even though *DiaCTC* (32) SP Random has slightly higher delay than iSLIP, considering its low arbitration complexity and fully distributed control feature, the performance is really prominent.

From above simulation results, we can conclude that *DiaCTC* (32) significantly enhances the performance, but has the same good feature and low complexity as *CTC* (*N*).


Asymmetric traffic
iSLIP
CTC (32) SP_Random
DiaCTC (32) SP_Random
Mean cell delay (time slots)
Offered Load (λ)


Asymmetric traffic
iSLIP
CTC (32) SP_Random
DiaCTC (32) SP_Random
Mean cell delay (time slots)
Offered Load (λ)

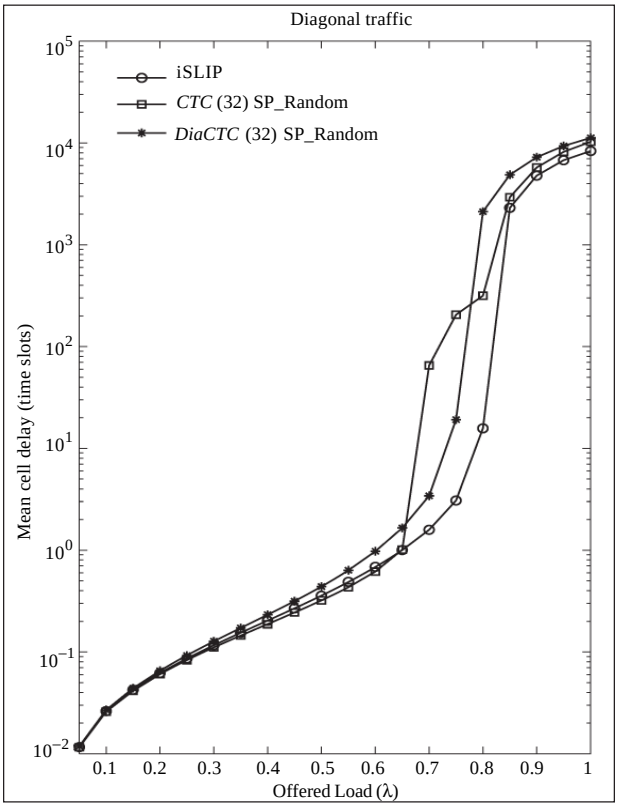


Figure 7. Mean cell delay under three nonuniform traffic schemes

5. Concluding Remarks

In our previous work, we proposed an innovative agile crossbar switch architecture *CTC*(*N*) and proved that its throughput of simple FIFO scheduling under Bernoulli i.i.d. uniform traffic is bounded by 63%. To improve performance, we proposed a fully distributed scheduling algorithm scheme called *staggered polling* (SP in short). Simulation results showed that, using SP scheduling algorithm scheme, the fully distributed schedulers "*smartly*" cooperate with each other and achieve high performance even with zero knowledge of other input ports.

This paper analyzes the main factor influencing performance of *CTC* (*N*). We present an improved contention-tolerant crossbar switch called *diagonalized contention-tolerant crossbar*, denoted as *DiaCTC* (*N*). Since *DiaCTC* (*N*) has the same fully distributed control property and low complexity of *CTC* (*N*), SP scheduling algorithms are able to operate on *DiaCTC* (*N*) without any change. *DiaCTC* (*N*) enhances the performance by balancing upstream traffic load for input ports. Simulation results show the outstanding improvement of performance of *DiaCTC* (*N*) with SP scheduling algorithms. *DiaCTC* (*N*) illustrates a new approach to improving *CTC* (*N*). However, out of sequence problem, which exists in *CTC* (*N*), remains a challenging open problem for *DiaCTC* (*N*). It can be reduced by designing sophisticated scheduling algorithms and queueing management methods.We remain the discussion of this problem in our subsequence papers. On the other hand, more algorithms can be designed for achieving good performance according to other QoS measures.

References

[1] Zhang, H. (1995). Service disciplines for guaranteed performance service in packet-switching networks, *In*: Proc. of IEEE, 83, p. 1373C1396, Oct. 1995.

[2] Hopcroft, J. E., Karp, R. M. (1973). An *n*5/2 algorithm for maximum matchings in bipartite graphs, *SIAM, J. Comput.*, 2 (4) 225-231, Dec.

[3] McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J. (1999). Achieving 100% throughput in an input-queued switch, *IEEE Trans. on Commun.*, 47 (8), Aug.

[4] Tamir, Y., Chi, H.C. (1993). Symmetric Crossbar arbiters for VLSI communication switches, *IEEE Trans. on Parallel and Distributed Systems*, 4 (1) 13-27.

[5] Anderson, T. E., Owicki, S. S., Saxe, J. B., Thacker, C. P. (1993). High speed switch scheduling for local area networks, *ACM Trans. on Computer Systems*, 11 (4) 319-352, Nov.

[6] McKeown, N., Varaiya, P., Warland, J. (1993). Scheduling cells in an inputqueued switch, *IEE Electronics Letters*, 29 (25) 2174-2175, Dec.

[7] McKeown, N. (1999). The iSLIP scheduling algorithm for input-queued switches, *IEEE/ACM Trans. on Networking*, 7 (2) 188-201, Apr.

[8] Chao, H. J. (2000). Saturn: A terabit packet switch using dual round-robin, *IEEE Commun. Magazine*, 8 (12) 78-84.

[9] Chuang, S. -T., Goel, A., McKeown, N., Prabhakar, B. (1999). Matching output queuing with a combined input output queued switch, *IEEE J. Select. Areas Commun.*, 17 (6) 1030-1039.

[10] Gale, D., Shapley, L. S. (1962). Colleage admissions and the stability of marriage, *Amer. Math. Monthly*, 69, p. 9-15.

[11] Nabeshima, M. (2000). Performance Evaluation of a Combined Input- and Crosspoint -Queued Switch, *IEICE Trans. on Commun.* E83-B (3), March.

[12] Rojas-Cessa, R., Oki, E., Chao, H. J. (2000). CIXOB-k: Combined input- and crosspoint-queued switch, *IEICE Trans. on Commun.*, E83-B (3) 737-741, Mar.

[13] Mhamdi, L., Hamdi, M. (2003). MCBF: A high-performance scheduling algorithm for buffered crossbar switches, *IEEE Commun. Letters*.

[14] Chrysos, N., Katevenis, M. (2003). Weighted fairness in buffered crossbar scheduling, *In*: Proc. of IEEE Workshop on High Performance Switching and Routing, p. 17-22, June.

[15] Rojas-Cessa, R., Oki, E., Chao, H. J. (2005). On the combined input-crosspoint buffered switch with round-robin arbitration, *IEEE Trans. on Commun.*, 53 (11) 1945-1951, Nov.

[16] Chuang, S.-T., Iyer, S., McKeown, N. (2005). Practical algorithms for performance guarantees in buffered crossbars, *In*: Proc. of IEEE INFOCOM, 2, p. 981- 991. 13-17, Mar.

[17] Lin, M., McKeown, N. (2005). The throughput of a buffered crossbar switch, *IEEE Commun. Letters*, 9 (5) 465-467, May.

[18] He, S. -M., Sun, S. -T., Guan, H. -T., Zhang, Q., Zhao, Y. -J., Gao, W. (2008). On guaranteed smooth switching for buffered crossbar switches, *IEEE/ACM Trans. on Networking*, 16 (3) 718-731.

[19] Qu, G., Chang, H. J., Wang, J., Fang, Z., Zheng, S. Q. (2011). Contentiontolerant crossbar packet switches, *International Journal of Communication Systems*, 24, p. 168-184.

[20] Qu, G., Chang, H. J., Wang, J., Fang, Z., Zheng, S. Q. (2010). Designing fully distributed scheduling algorithms for contention-tolerant crossbar switches", *Proc. of 11th International Conference on High Performance Switching and Routing*, June 13-17.

[21] Qu, G., Chang, H. J., Wang, J., Fang, Z., Zheng, S. Q. (2010). Queueing analysis of multi-layer contention-tolerant crossbar switch, *IEEE Commun. Letters*, 14 (10) 972-974.

[22] Schoene, R., Post, G., Sander, G. (1999). Weighted arbitration algorithms with priorities for input-Queued switches with 100% throughput, *Broadband Switches Symposium*.

[23] Chang, C-S., Lee, D-S., Jou, Y-S. (2001). Load balanced birkhoff-von neumann switches, *IEEE HPSR*, p.276-280, April.

[24] Giaccone, P., Shah, D., Prabhakar, B. (2002). An implementable parallel scheduler for input-queued switches, *Micro, IEEE*, 22, p. 19-25.