K-Selector-Based Dispatching Algorithm for Clos-Network Switches

Mei Yang[†], Mayauna McCullough[‡], Yingtao Jiang[†], and Jun Zheng[*]
[†] Department of Electrical and Computer Engineering, University of Nevada Las Vegas, NV 89154, USA
[‡] Department of Computer Science, Fort Valley State University, Fort Valley, GA 31030, USA
[*] Department of Computer Science, Queens College, New York, NY, USA
E-mail: [†]{meiyang, yingtao}@egr.unlv.edu, [*]zheng@cs.qc.edu

***Abstract*— In this paper, we address the scheduling problem for Clos-network switches with no buffers at the central stage. Existing scheduling (dispatching) algorithms for this type of switch, such as CRRD and CMSD, are too complex for implementation. We consider efficient and practical dispatching algorithms and propose the k-selector based dispatching (KBD) algorithm. The KBD algorithm differs from other dispatching algorithms in the phase of matching within input modules. In KBD, only one k-selector is used at each input module to select m out of nk requests to send to the corresponding central modules. As such, the interconnection wires are totally removed in input modules and the required time for the first phase is improved significantly. Through simulations, we show that KBD achieves comparable performance to CMSD under Bernoulli and bursty traffic.**

I. Introduction

The exponential increasing of Internet traffic imposes high-speed and large-capacity switches and routers. Two approaches are employed to implement a high-speed switch. One is a single-stage switch architecture [1]. An example of the single-stage switch architecture is crossbar switches. However, crossbar switches are not practical for larger switch size. For example, a 1000 $\times$ 1000 switch would require 1M crosspoints, which is prohibitively expensive to implement.

The other approach is to use a multi-stage switch architecture, such as a Clos-network switch [1]. A Clos-network switch architecture consists of three stages: the input stage, the central stage, and the output stage. Each stage is composed of multiple modules. Each module can be a crossbar switch. Each central stage has one input for each input stage and one output for each output stage. The number of alternative paths between an input port and an output port is equal to the number of central modules [1]. The Clos-network switch architecture is very attractive because of its scalability.

We can classify the Clos-network switch architecture into two types, one has buffers to store cells in the central (stage) modules and the other has no buffer in the central modules [1]. For example, the ATLANTA switch is an example of the second type [5]. The focus of our study is on Clos-network switches with no buffer at the central modules. Since there is no buffer at the central stage to resolve contention, the scheduling (dispatching) between input modules and central modules is critical for the switch performance.

The design goals of scheduling algorithms for multi-stage switches are high throughput, high speed, no starvation, and simple implementation. In the literature, a number of scheduling algorithms for Clos-network switches have been proposed [1], [3], [4], [8]. Even though some of existing solutions achieve very high throughput, they are all too complex for implementation. In this paper, we focus our study on efficient yet practical scheduling algorithms for Clos-network switches. We propose a k-selector-based dispatching (KBD) algorithm which does not require the increase of internal bandwidth. Using k-selectors, the KBD algorithm ensures fairness in scheduling cells from input modules to output modules. Through simulations, we show that the scheduling performance of the KBD algorithm is comparable to CMSD. More importantly, using k-selectors, the KBD algorithm is faster and simpler than existing dispatching algorithms.

The rest of the paper is organized as follows. Section II reviews existing dispatching algorithms. Section III presents the KBD algorithm. Section IV discusses its implementation complexity. Section V presents the simulation results of the KBD algorithm and compares them with CMSD. Section VI concludes the paper.

II. Related Work

Figure 1 shows a three-stage Clos-network switch, which consist of k input modules (IMs), k output modules (OMs), and m central modules (CMs). Each IM/OM has n input/output ports. Each IM maintains nk virtual output queues (VOQs), each dedicated for holding cells coming at the IM destined for a particular output port. The following notations are used in this paper.

i	IM number, where $0 \le i \le k-1$.
j	OM number, where $0 \le j \le k-1$.
h	Input port (IP)/output port (OP) number in each IM/OM, respectively, where $0 \le h \le n-1$.
r	CM number, where $0 \le r \le m-1$.
$IM(i)$	(i + 1)th IM.
$CM(r)$	(r + 1)th CM.
$OM(j)$	(j + 1)th OM.
$IP(i,h)$	(h + 1)th IP at $IM(i)$.
$OP(j,h)$	(h + 1)th OP at $OM(j)$.
VOQ(i, j, h)	VOQ at $IM(i)$ that stores cells destined for $OP(j,h)$.
$L_I(i,r)$	Output link at $IM(i)$ and connected to $CM(r)$.
$L_C(r,j)$	Output link at $CM(r)$ and connected to $OM(j)$.

In the literature, dispatching algorithms proposed for Clos-network switches include the distributed and random dispatching schemes [1], concurrent round-robin dispatching scheme (CRRD) [8], concurrent master-slave round-robin dispatching

Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)
0-7695-2497-4/06 $20.00 © 2006 IEEE

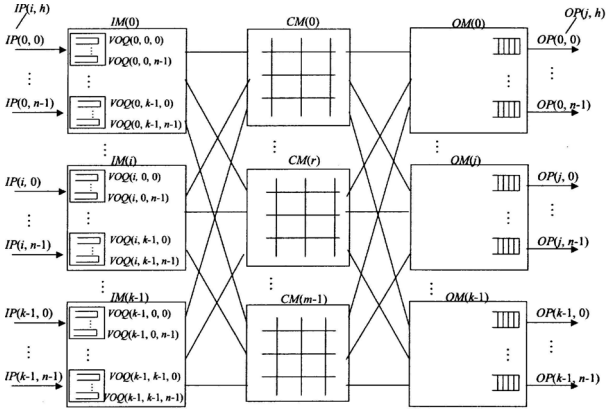

Fig. 1. Clos-network switch with VOQs in the IMs.

scheme (CMSD) [8], frame-based exhaustive matching algorithm (FEM) [3], and matching algorithm for Clos-network switches (MAC) [4].

The random dispatching algorithm was proposed for the ATLANTA switch [5]. There are two phases of the random dispatching algorithm: matching within IMs and matching between IMs and CMs.

Phase 1: Each IM selects up to m requests out of nk nonempty VOQs in the round-robin manner and sends the selected requests to m randomly selected CMs.

Phase 2: The arbiter associated with the output link of each CM randomly selects one request among up to k requests. Each CM sends up to k grants back to the corresponding IMs. If a VOQ at the IM receives the grant from the CM, it sends the corresponding cell in the next time slot. Otherwise, the VOQ will send a request again in the next time slot.

As one can see, random dispatching can send off cells evenly to the CMs. However, a high switch throughput cannot be achieved because of contention at the CMs [1]. One way to solve that problem is to increase the size of the internal bandwidth [1], which makes it difficult to implement a high-speed switch in a cost-effective manner.

The concurrent round-robin dispatching (CRRD) scheme [8] improves the random dispatching scheme by using round-robin arbiters at both IMs and CMs. Each IM has nk VOQ arbiters and m output-link arbiters. Each CM has m output link arbiters. Each of these arbiter is associated with a pointer indicating the selection starting point. CRRD also has two phases, matching within IMs and matching between IMs and CMs. Phase 1 is composed of a series of iterative steps.

Phase 1: Matching within IMs

Step 1: Each nonempty VOQs sends a request to every output-link arbiter.

Step 2: Each output link arbiter selects a request from nonempty VOQs in the round-robin fashion starting from the position pointed by its pointer. It then sends the grant to the selected VOQ.

Step 3: The VOQ arbiter selects one grant to accept starting from the position pointed by its pointer. The granted requests are sent to their corresponding CMs.

Phase 2: Matching between IMs and CMs

Step 1: The arbiter associated with the output link of each CM selects one request among up to k requests in the round-robin manner starting from the position pointed by its pointer. Each CM sends up to k grants back to the corresponding IMs.

Step 2: If a VOQ at the IM receives the grant from the CM, it sends the corresponding cell in the next time slot. Otherwise, the VOQ will send a request again in the next time slot.

Due to the desynchronization effect, CRRD achieves 100% throughput under both uniform and nonuniform traffic [8].

The concurrent master-slave round-robin dispatching (CMSD) scheme is an improved version of CRRD that preserves CRRD's advantages and provides more scalability [8]. CMSD differs from CRRD in Phase 1. Each IM has m master output-link round-robin arbiters, mk slave output-link round-robin arbiters, and nk VOQ round-robin arbiters. Each arbiter is associated with a pointer to indicate the selection starting point. The master output-link arbiter associated with $L_I(i,r)$ is denoted as $ML(i,r)$. The slave output-link arbiter associated with $L_I(i,r)$ and VOQ group $G(i,j)$ is denoted as $SL(i,j,r)$. The phase 1 of CMSD works iteratively as follows.

Phase 1: Matching within IMs

Step 1: There are two sets of requests sent to the output-link arbiters of an IM. One set of requests are sent from a nonempty $VOQ(i,j,h)$ to every associated slave arbiter $SL(i,j,r)$ within $G(i,j)$. The other set is a group-level request sent from $G(i,j)$ that has at least one nonempty VOQ to every master arbiter $ML(i,r)$.

Step 2: Each $ML(i,r)$ chooses a request among k VOQ groups independently in a round-robin fashion starting from the position pointed by its pointer and sends the grant to $SL(i,j,r)$. $SL(i,j,r)$ selects one VOQ request in a round-robin fashion starting from the position pointed by its pointer and sends the grant to the selected VOQ.

Step 3: The VOQ arbiter selects one grant to accept starting from the position by its pointer. The granted requests are sent to their corresponding CMs.

Figure 2 [8] shows an example of the Phase 1 of CMSD. The steps of Phase 2 of CMSD is as same as those of CRRD. The pointers of each arbiter in the IM is updated to one position after the granted position only if the matching within the IM is achieved at the first iteration in Phase 1 and the request is also granted by the CM in Phase 2.

Similar to CRRD, CMSD also decreases contention at the CM because the pointers are desynchronized. CMSD provides 100% throughput under both uniform and nonuniform traffic as well. Since the master output-link arbiter of CMSD can choose a VOQ group, not a VOQ itself in CRRD, the master output-link arbiter of CMSD affects the matching between the IM and CM more than the output-link arbiter of CRRD [8]. Therefore, CMSD makes the matching between the IM and CM more efficient than CRRD when the number of iterations in the IM increases. The throughput of CMSD is also independent of the

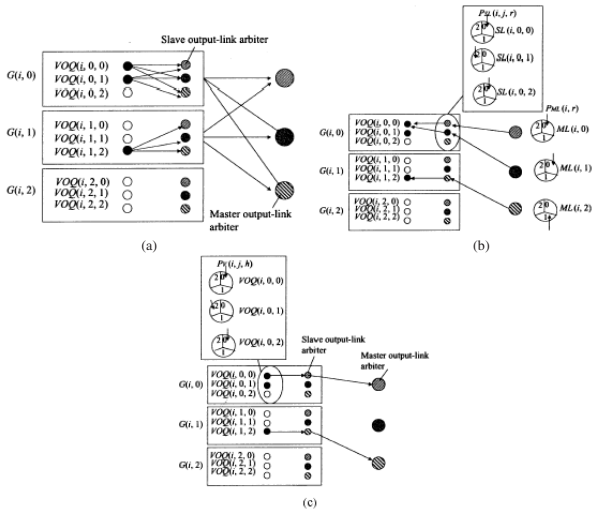

Fig. 2. An example of phase 1 of the CMSD algorithm for $n = k = m = 3$.

number of iterations of the IM. The tradeoff is the much higher complexity of CMSD than CRRD.

The FEM scheme uses the exhaustive dual round-robin matching (EDRRM) scheme to improve the performance under unbalanced traffic. FEM is composed of two phases. In Phase 1, it uses EDDRM to find a matching between input ports and output ports. In Phase 2, a parallel matching scheme is employed to find routing paths between matched input ports and output ports. The EDDRM scheme improves throughput by maintaining the existing matching pairs between the input ports and output ports so that the number of unmatched inputs and outputs is dramatically decreased [6]. FEM further improves the throughput by solving the starvation problem of some input ports. The way FEM overcomes this problem is by setting a timer for each head-of-line (HOL) frame. When the timer expires, the request from the "expired" frame has the highest preference to be granted [6]. The output port arbiter will favor the request with higher preference. And the input port arbiter will accept the grant with higher preference first. By such a scheme, FEM achieves very high throughput under both uniform and unbalanced traffic. However, finding routing paths in Phase 2 is difficult. Also the timer introduces extra complexity to the scheduler design.

The MAC algorithm has two phases: superframe matching and superframe decomposition [5]. The superframe matching phase determines the superframe matrix. Each entry in the matrix represents matching opportunities between each scheduling IM and each scheduling OM in one superframe. The superframe matrix is determined by the iterative request/grant/accept algorithm [5]. In the superframe decomposition phase, the superframe matrix is decomposed into $F * k$ module matching matrices. Each module-matching matrix records the matching status between the scheduling IMs and OMs in one matching cycle of the next superframe [5].

Once the module-level matching algorithm has been completed, the matching sequence between the scheduling IMs and OMs is determined for the next superframe. The port-level matching algorithm consists of k matching cycles in a frame. In each matching cycle, the port-level matching algorithm includes two steps: port-to-port matching assignment and CM assignment [5]. MAC can achieve high performance and maintain good scalability. However, the problem with MAC is the extra delay introduced in the k matching cycle and and high implementation complexity.

III. The KBD Algorithm

In this section, we propose a k-selector-Based Dispatching (KBD) algorithm for Clos-network switches. In KBD, each IM uses a k-selector [10] to select m out of nk requests to send to CMs. Each CM has k round-robin arbiters, each used to select one request to its corresponding OM. We use $P_I(i)$ to indicate the pointer associated with the k-selector at $IM(i)$ and $P_C(r, j)$ to indicate the pointer for the arbiter for $OM(j)$ in $CM(r)$.

The KBD algorithm is an iterative algorithm with each iteration consisting of two phases. The first phase of KBD matches the VOQ requests in each IM with the output links of the IM by selecting up to m out of nk requests using the k-selector. The second phase of KBD matches the requests sent through the output links of each IM with the output links of the corresponding CMs. The detailed steps of each phase are described as follows.

Phase 1: Matching within IMs

Step 1: Each nonempty VOQ sends a request to the k-selector of the IM.

Step 2: The k-selector selects up to m requests in the round-robin manner starting from P_i for $IM(i)$.

Step 3: The selected requests are sent to the corresponding CMs through $L_I(i, r)$'s.

Phase 2: Matching between IMs and CMs

Step 1: The round-robin arbiter associated with $OM(j)$ selects one request by searching from the position of $P_C(r, j)$. $P_C(r, j)$ is updated to the request after the granted one.

Step 2: Each $CM(r)$ sends the grants back their corresponding IMs through $L_I(i, r)$'s.

If the IM receives the grant from the CM, it sends a corresponding cell from that VOQ in the next time slot. If $IM(i)$ receives any grant from CMs, its k-selector pointer, $P_I(i)$, is updated to the request next to the last granted one or the first request which is not granted if there is no request granted. The update of the k-selector pointers is only performed in the first iteration.

Figure 3 shows an example of one iteration of the KBD algorithm for $n = k = m = 2$. Assume that the k-selector pointers of $IM(0)$ and $IM(1)$ are initialized as 0 and the round-robin pointers at $CM(0)$ and $CM(1)$ are all initialized as 0. In Step 1 of Phase 1, $VOQ(0, 0)$, $VOQ(0, 2)$ and $VOQ(0, 3)$ send requests to the k-selector at $IM(0)$ while $VOQ(1, 0)$, $VOQ(1, 1)$, and $VOQ(1, 2)$ send requests to the k-selector at $IM(1)$. In Step 2 of Phase 1, the requests from $VOQ(0, 0)$ and $VOQ(0, 2)$ are selected while the requests from $VOQ(1, 0)$ and $VOQ(1, 1)$ are selected. In Step 3 of Phase 1, the selected requests are sent to CMs following the selection order.

In Step 1 of Phase 2, the arbiter for $L_C(0,0)$ grants the request sent through $L_I(0,0)$ (i.e., from $VOQ(0,0)$), the arbiter for $L_C(1,0)$ grants the request sent through $L_I(1,1)$ (i.e., from $VOQ(1,1)$), and the arbiter for $L_C(1,1)$ grants the request sent through $L_I(0,1)$(i.e., from $VOQ(0,2)$). The pointer of each arbiter is updated as shown in the figure. The grants are sent back to the corresponding VOQs. According to the grant situation, $P_I(0)$ is updated to 3 while $P_I(1)$ is updated to 2. In the next iteration, the ungranted VOQ requests will be considered.

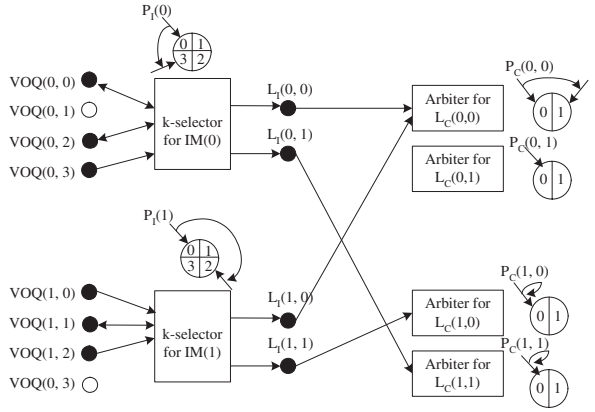

Fig. 3. An example of the KBD algorithm for $n = k = m = 2$.

Similar to CRRD and CMSD, the k-selector pointers at each IM and the arbiter pointers at each CM have desynchronization effect. In Figure 4, we demonstrate how the pointers are desynchronized using an example of $n = k = m = 2$. Assuming that no VOQ is empty, each VOQ sends a request at every time slot to the k-selector at the IM. All the pointers are initialized as 0. At time slot $T = 0$, the selected VOQ requests at $IM(0)$ are $VOQ(0,0)$ and $VOQ(0,1)$, which are sent to $CM(0)$ and $CM(1)$ respectively. The selected VOQ requests at $IM(1)$ are $VOQ(1,0)$ and $VOQ(1,1)$, which are sent to $CM(0)$ and $CM(1)$ respectively. At $CM(0)$, $VOQ(0,0)$ is granted and $P_C(0,0)$ is updated to 1, while at $CM(1)$, $VOQ(0,1)$ is granted and $P_C(1,0)$ is updated to 1. Hence $P_I(0)$ is updated to 2. At $T = 1$, due to the desynchronization of $P_I(0)$ and $P_I(1)$, requests from $VOQ(0,2)$, $VOQ(0,3)$ and requests from $VOQ(0,0)$ and $VOQ(0,1)$ will be selected. All requests are granted at $CM(0)$ and $CM(1)$. All the pointers are updated as shown in the figure. Due to the desynchronization effect, in the following time slots, there is no contention at all CMs and the largest size of matching is obtained.

	T	0	1	2	3	4	5	6	7
IM(0)	P_i(0)	0	2	0	2	0	2	0	2
IM(1)	P_i(1)	0	0	2	0	2	0	2	0
CM(0)	P_c(0, 0)	0	1	0	1	0	1	0	1
	P_c(0, 1)	0	0	1	0	1	0	1	0
CM(1)	P_c(1, 0)	0	1	0	1	0	1	0	1
	P_c(1, 1)	0	0	1	0	1	0	1	0

Fig. 4. Desynchronization effect of the pointers in the KBD algorithm for $n = k = m = 2$.

	Timing (-gate delay)	Area (gates)	Interconnection (wires)
CRRD	$O(i(\log nk + \log m) + \log k)$	$O(nkm)$	$\frac{3}{4}nkm(nk-1)(m-1)$
CMSD	$O(i \max(\log k, \log n) + \log m) + \log k)$	$O(nkm)$	$\frac{3k}{4}nm(n-1)(m-1)+\frac{3}{4}km(k-1)(m-1)$
KBD	$O(i(\log nk))$	$O((nk)^2)$	N/A

TABLE I
IMPLEMENTATION COMPLEXITY OF CRRD, CMSD, AND KBD.

IV. IMPLEMENTATION COMPLEXITY

In this section, we analyze the implementation complexity of the KBD algorithm. In specific, we analyze the required time, area cost, and interconnection complexity of the KBD algorithm and compare them with those of CRRD and CMSD. As one can see, at each CM, there are k output-link arbiters for all the three algorithms. There are up to k requests to each output-link arbiter. Using the parallel round-robin arbiters proposed in [11], the required time and area cost for the output-link arbiter are $O(\log k)$-gate delay and $O(k)$-gates respectively. At each CM, all the output link arbiters consume $O(mk)$-gates.

The three algorithms differ in the complexity at each IM. For CRRD, each IM has nk VOQ arbiters and m output-link arbiters. For each VOQ arbiter, there are at most m requests. Hence, the required time and area cost of each VOQ arbiter are $O(\log m)$-gate delay and $O(m)$-gates, respectively. For each output-link arbiter at each IM, there are at most nk requests. Therefore, the required time and area cost of each output-link arbiter at each IM are $O(\log nk)$-gate delay and $O(nk)$-gates, respectively. Thus the total time complexity of the CRRD is $O(i(\log nk + \log m) + \log k)$-gate delay, where i is the number of iterations performed in Phase 1, and the total area cost of the arbiters at each IM is $O(nkm)$-gates. In CRRD, each VOQ arbiter is connected to all output-link arbiters with three groups of wires. As analyzed in [8], the interconnection complexity of CRRD is determined by $\frac{3}{4}nkm(nk-1)(m-1)$.

For CMSD, each IM has nk VOQ arbiters, nk slave output-link arbiters, and m master output-link arbiters. For each VOQ arbiter, there are at most m requests. Hence, the required time and area cost of each VOQ arbiter are $O(\log m)$-gate delay and $O(m)$-gates, respectively. For each slave output-link arbiter, there are at most nk requests. Therefore, the required time and area cost of each slave output-link arbiter at each IM are $O(\log nk)$-gate delay and $O(nk)$-gates, respectively. For each master output-link arbiter, there are at most m requests from slave output-link arbiters. Therefore, the required time and area cost of each master output-link arbiter at each IM are $O(\log m)$-gate delay and $O(m)$-gates, respectively. Thus the total time complexity of CMSD is $O(i(\max(\log k, \log n) + \log m) + \log k)$-gate delay and the total area cost of the arbiters at each IM is $O(nkm)$-gates. In CMSD, each VOQ arbiter is connected to its own slave output-link arbiters with three groups of wires and each VOQ group is connected to all the master output-link arbiters with three groups of wires. The total interconnection complexity of CMSD is $\frac{3k}{4}nm(n-1)(m-1) + \frac{3}{4}km(k-1)(m-1)$[8].

For KBD, each IM has one k-selector. For each k-selector,

there are at most nk requests. As shown in [10], the required time and area cost for each k-selector are $O(\log nk)$-gate and $O((nk)^2)$ gates. Since only one k-selector is needed, no interconnection wire is needed for KBD. Table I summarizes the implementation complexity of the three algorithms. Compared with CRRD and CMSD, KBD is much faster and has no interconnection cost. The tradeoff is that the area cost of KBD is higher than CRRD and CMSD.

V. Simulation Results

In this section, we evaluate the scheduling performance of the KBD algorithm in terms of the average cell delay. The cell delay is the summation of the cell's waiting time in the VOQs at IMs and the transit time of cell through the switch. We simulate the KBD algorithm and compare it with the performance of the CMSD algorithm. Two traffic models, uniform Bernoulli traffic and uniform bursty traffic, are considered in our simulations. The bursty traffic is modelled following the two-state Markov chain model as in [7], [9]. In the following, we show the average cell delay of CMSD and KBD for a Clos-network switch of size $n = k = m = 8$.

Fig. 5 shows that under uniform traffic, KBD with one iteration achieves similar performance to CMSD with one iteration under load less than 0.60. For load above 0.60, the performance of KBD is not as good as CMSD. The performance of KBD is improved with more number of iterations. With four iterations, KBD achieves nearly the same performance as CMSD.

Fig. 6 shows that under bursty traffic, KBD with one iteration achieves similar performance to CMSD with one iteration under load less than 0.60. For load above 0.60, the performance of KBD is not as good as CMSD. The performance of KBD is improved with more number of iterations. With four iterations, KBD achieves even better performance as CMSD.

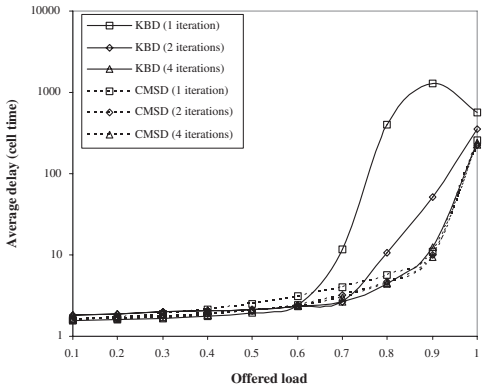

Fig. 5. Delay performance of KBD and CMSD for n=m=k=8 under uniform Bernoulli traffic.

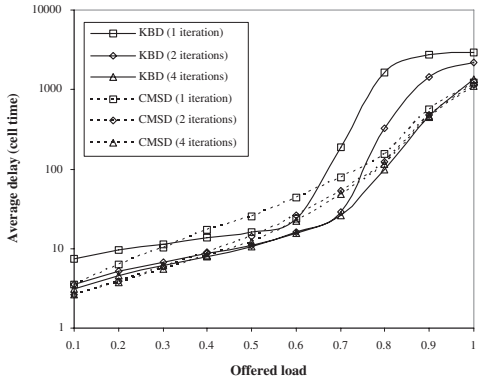

Fig. 6. Delay performance of KBD and CMSD for n=m=k=8 under bursty traffic with average burst size = 16.

VI. Conclusion

In this paper, we propose a k-selector-based dispatching (KBD) algorithm for Clos-network switches. The KBD algorithm is different from other dispatching algorithms (such as CRRD and CMSD) in the first phase, i.e., matching within input modules. It uses only one k-selector to select m out of nk requests to send to the corresponding central modules. Hence, there is no need for interconnection wires. Also we show that the required time of KBD is also better than CRRD and CMSD. Through simulations, we show that KBD achieves performance comparable to CMSD under both uniform Bernoulli and bursty traffic. In a sum, KBD is an efficient and practical scheduling algorithm for multistage switches. Future work includes evaluation of KBD under nonuniform traffic and extension of KBD to provide differentiated services.

References

[1] H. J. Chao, C. Lam, and E. Oki, *Broadband packet switching technologies-a practical guide to ATM switches and IP routers*, John Wiley & Sons, 2001.

[2] H. J. Chao, "Next generation routers," *Proc. of IEEE*, vol. 90, no. 9, Sept. 2002, pp. 1518-1558.

[3] H. J. Chao, Z. Jing, and K. Deng, "Packet scheduling scheme for a 3-stage Clos-network photonic switch," in *Proc. ICC*, 2003, vol. 2, pp. 1293-1298.

[4] H. J. Chao, Z. Jing, S. Y. Liew, "Matching algorithms for three-stage bufferless Clos network switches," *IEEE Communications Magazine*, vol. 41, no. 10, Oct. 2003, pp. 46-54.

[5] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-cost scalable switching solutions for broadband networking: the ATLANTA architecture and chipset," *IEEE Commun. Mag.*, pp. 44-53, Dec. 1997.

[6] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, pp. 406-424, Mar. 1953.

[7] N. McKeown, "The *i*SLIP scheduling algorithm for input-queued switches," IEEE/ACM Trans. Networking, vol. 7, no. 2, pp. 188-201, Apr. 1999.

[8] E. Oki, Z. Jing, R. Rojas-Cessa, H. J. Chao, "Concurrent round-robin-based dispatching schemes for clos-network switches," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, Dec. 2002, pp. 830-844.

[9] Mei Yang and S. Q. Zheng, "An efficient scheduling algorithm for CIOQ switches with space-division multiplexing expansion", in *Proc. IEEE INFOCOM*, 2003, pp.1643-1650.

[10] S. Q. Zheng, Mei Yang, and F. Masetti-Placci, "Constructing schedulers for high-speed, high-capacity switches/routers," *Int'l J. Computers and Applications*, vol. 25, pp. 264-271, Nov. 4, 2003.

[11] S. Q. Zheng and Mei Yang, "Parallel round-robin arbiters," to appear in *IEEE Trans. Parallel and Distributed Systems*.