The kDRR Scheduling Algorithms for Multi-server Packet Switches

Mei Yang [†], S. Q. Zheng [*†] and Dominique Verchere[‡]

[∗] Department of Electrical Engineering [†] Department of Computer Science
University of Texas at Dallas, Richardson, TX 75083-0688, USA
{sizheng, meiyang}@utdallas.edu
[‡] Research & Innovation, Alcatel USA, Plano, Texas 75075, USA
Dominique.Verchere@alcatel.com

Abstract— **Tremendous increase of Internet traffic demands high speed, large capacity IP switch routers. The introduction of multi-server switch architectures not only makes it possible to scale to larger switch capacities, but also imposes special requirements on scheduling algorithms running in the central scheduler. In this paper, we model the multi-server switch scheduling problem as a maximum bipartite K-matching problem. We generalize iSLIP and DRR to multi-server switch architectures and propose two distributed scheduling algorithms kDRR_RGA and kDRR_RG for finding maximal K-matchings. Through simulations, we show that both algorithms achieve comparable performance with high throughput under uniform traffic. Using programmable k-selectors, kDRR_RGA and kDRR_RG are ready to be implemented at high speed.**

Key words: Multi-server switch architectures, kDRR scheduling algorithms, K-matching, programmable k-selectors.

1 Introduction

The exponential increase of Internet traffic demands high speed, large capacity IP switch routers. As the key to the success of the next generation Internet (NGI), terabit IP switch routers have received attention from both research and commercial communities. In [5], Chao etc. reviewed several gigabit IP switch routers, optical packet switch architectures and opto-electronic packet switches. Combining the strength of both optical and electronic technologies, opto-electronic packet switches are proposed to be a competitive architecture for terabit IP switch routers [5].

In [3], [4], a new opto-electronic packet switch architecture with terabit capacity was proposed. This switch architecture employs an optical switching matrix (OSM) to interconnect multiple electronic input/output modules. IP packets are decomposed into fixed-size cells as they arrive at line cards (LCs), buffered at data concentrator cards (DCCs), transferred through the optical switching matrix, and then reassembled into IP packets before they depart from LCs. Each DCC connects to multiple LCs. On each DCC, cells are buffered into virtual output queues (VOQs) according to their destination DCCs to avoid the problem of head-of-line (HoL) blocking [11]. Within a VOQ, cells are composed into larger, fixed-size composite packets (CPs) to be transferred through the OSM in one switch slot. Using WDM technology, the optical path between a DCC and the OSM can carry multiple optical channels (wavelengths) simultaneously. The number of wavelengths available and bandwidth of each wavelength decide the switch capacity. Taking the example of a switch of 16 DCCs, assume each DCC connects with the OSM with 16 wavelengths, each carrying 10Gbps. The total switch capacity is 2.56Tbps.

We abstract each DCC as a port. Assume the number of wavelengths on each port is equal to the number of LCs connecting to each port and the bandwidth of each wavelength is equal to the aggregated bandwidth of one LC in one switch slot. A group of wavelengths and their associated buffers are composed as a service unit, named a server. Since the total switch capacity is fixed, different number of servers provide different granularity of services. The more number of servers, the finer the granularity [9]. Figure 1 illustrates an abstracted $N \times N$ switch model with each port carrying K servers. A cell slot is the time unit during the arrivals of two successive cells on one LC. A switch slot is the time unit between two successive operations of the optical switching matrix. One switch slot is equal to multiple cell slots. During each switch slot, the central scheduler computes a conflict-free matching between input port servers and output port servers and configures the switching matrix.

The multi-server switch architecture also imposes special requirements on the scheduling algorithm running in the central scheduler. Three scheduling algorithms have been proposed in [9] for the multi-server switch architecture, including multiSLIP, FR-DRR and PDRR. However, the difference between multi-server switch scheduling and single-server scheduling is not clearly studied. And how to implement these algorithms at high speed is still a problem. In [9], implementation schemes using programmable priority encoders (PPE) [10] were discussed. Since PPE can only make one selection each time, we need run PPE K times to select K requests in each phase of the algorithm. Such an implementation is not desired for packet switches with high line rate and/or large K's.

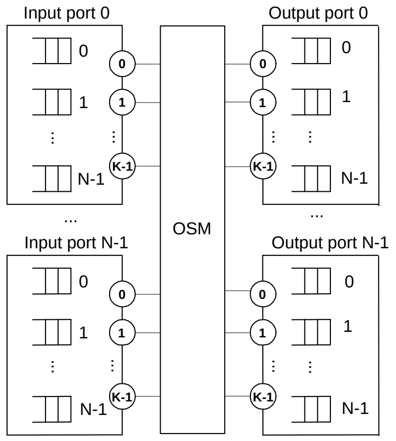

Fig. 1. Abstraction of the multi-server switch architecture.

In this paper, we first define the scheduling problem on the multi-server switch architecture as a maximum K-matching problem on a bipartite graph. Since maximum size K-matching algorithms are hard to implement and can cause unfairness, we generalize iSLIP and DRR to the multi-server switch architecture and present two distributed iterative scheduling algorithms, kDRR_RGA and kDRR_RG, for finding maximal K-matchings between inputs and outputs. Because these two algorithms are equivalent statistically under unform traffic, we take the example of

kDRR_RGA and study its performance of with uniform Bernoulli arrivals and uniform bursty arrivals. Simulation results of 16×16 switch show that kDRR_RGA achieves up to 100% throughput with both Bernoulli arrivals and bursty arrivals. Compared to multiSLIP, kDRR_RGA performs better with one or two iterations and it needs less average number of iterations to converge. We show that the implementation of kDRR algorithms is much simpler than multiSLIP. Using programmable k-selectors [15], both kDRR_RGA and kDRR_RG are ready to be implemented at high speed. The time and area complexity of such an implementation is independent of K.

2 Problem Statement

Our study is focused on scheduling algorithms of an $N \times N$ multi-server switch architecture as shown in Figure 1. In our study, we assume each input/output port connects to N input/output LCs. Each input port maintains N VOQs, denoted as $Q_{i,j}$, where i is the input port no., j is the output port no., $0 \leq i, j \leq N-1$. Each output port maintains N output queues (OQs), each OQ is associated with an output LC. Let T be the switch slot time and T' be the cell slot, we have $T = N'T'$, where $N' \geq N$. In each cell slot, either one cell or none cell comes from one LC. Thus, during one switch slot, at most $N'N$ cells enter in one input port. Assume K is the number of servers on each input/output port, during each switch slot, each server can transfer a CP, which consists of N/K elementary composite packets (ECP). Each ECP is composed of N' cells. The composition of a CP is shown in Figure 2 [9]. Thus, the total service capacity of an input port is $N'N$ cells while the service rate of each server is given by $N'N/K$ cells per switch slot.

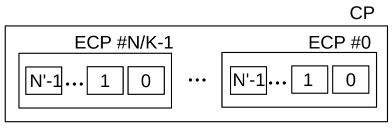

Fig. 2. The structure of a CP

To fully utilize the switch capacity, it is desirable to transfer only full CPs. The delay specification of a high speed packet switch requires that the probability, of the transit time of all incoming data exceeding a statistical delay bound, is limited to a small value [3]. Thus, it may be necessary to sacrifice some switch capacity to transfer packets that will grow stale. Under low traffic load, non-full and non-stale packets should be served if there are still available servers. As a result, the switch scheduling algorithm that decides, in each switch slot, what requests should be generated and what connections should be made between input port servers and output port servers are extremely important for the performance of the switch.

The scheduling problem on the multi-server switch architecture can be modelled as a K-matching problem on the bipartite graph $G = (V, E)$, where

- $V = V_1 \cup V_2$, $V_1 = \{\text{input ports}\}$,$V_2 = \{\text{output ports}\}$, $N =\mid V_1 \mid=\mid V_2 \mid$.
- $E = \{$connection requests from input ports to output ports$\}$, $M =\mid E \mid$.

A K-matching $\mathcal{K} \subseteq E$ such that no node of G is incident with more than K edges in $\mathcal{K}$. A maximum size K-matching is one with the maximum number of edges, while a maximal size K-matching is one that no more edges can be trivially added. A perfect K-matching is one that each node is incident with K edges in $\mathcal{K}$. Figure 3 gives an example of a maximum 2-matching and a maximal 2-matching of a 4×4 switch. With this maximum 2-matching, $Q_{0,0}$, $Q_{0,2}$, $Q_{1,1}$, $Q_{1,3}$ and $Q_{2,1}$ will be served.

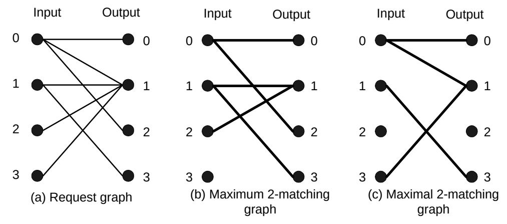

Fig. 3. A maximum and maximal 2-matching of a 4×4 switch.

The maximum size K-matching (MKM) problem can be taken as a special case of the bipartite b-matching problem [8], [17]. The MKM problem can be solved by the generalized augmenting path algorithm with time complexity of $O(KNM)$ [14]. It can also be transformed to a maximum-flow problem in $O(M)$ time. Since the transformed flow network is a unit network [16], we can use Dinic's algorithm to solve the corresponding maximum-flow problem in $O(\sqrt{N}M)$ time [16]. However, these MKM algorithms are too complex to be implemented in high speed packet switches. Another noticeable problem with maximum size K-matching is that it may cause unfairness. For example, in Figure 3, if $Q_{0,0}$, $Q_{0,2}$, $Q_{1,1}$, $Q_{1,3}$ and $Q_{2,1}$ continue having requests and other VOQs continue having no requests in successive switch slots, then $Q_{0,1}$ may get starved since edge $(0, 1)$ is not in any maximum 2-matchings.

For practical use, we desire scheduling algorithms with high throughput, starvation free, fast speed and simple implementation [13]. A maximal K-matching is a good option other than a maximum K-matching. Maximal matching scheduling algorithms proposed for input-buffered switches include PIM [1], RRM, iSLIP [13], DRR [6], and iterative ping-pong arbitration scheme [7], etc.. Among these scheduling algorithms, iSLIP and DRR can achieve up to 100% throughput under uniform traffic and are readily implemented in hardware at high speed. In the following, we generalize the idea of iSLIP and DRR and present the k-server dual-round-robin (kDRR) scheduling algorithms for multi-server switch architectures. These two scheduling algorithms use a distributed approach to find a maximal K-matching between input ports and output ports.

3 kDRR Algorithms

As iSLIP and DRR, kDRR algorithms are iterative and use rotating priority ("round-robin") arbitration to schedule each active input and output in turn. Both kDRR_RGA and kDRR_RG adopt the *conservative* policy [4], in which requests are first generated for full CPs and non-full CPs with time-limit approaching and then for other non-full CPs if there are still available servers. Due to the space limit, here we only present kDRR with request, grant and accept (kDRR_RGA). Another version, kDRR with request and grant (kDRR_RG), can be constructed symmetrically.

We use the following notations. Let I_i and O_j be the abstraction of N input ports N output ports respectively, where $0 \leq i, j \leq N-1$. For each I_i, let a_i be its accept pointer, indicating its accept starting point, where $0 \leq a_i \leq N-1$. For output O_j, let ga_j be its issued grant pointer, indicating its issued grant starting point, where $0 \leq ga_i \leq N-1$. Let an $N \times N$ 0-1 matrix R be the request matrix, where $R_{i,j} = 1$ denotes $Q_{i,j}$ has a request. Let KI_i and KO_j denote the number of available servers at I_i and at O_j respectively. For the first iteration, $\forall 0 \leq i, j \leq N-1$, $KI_i = K$, and $KO_j = K$. The three steps of kDRR_RGA operate as follows.

Request: $\forall I_i, 0 \leq i \leq N-1$, if I_i has available servers and unresolved requests (requests to outputs with available servers), it sends all unresolved requests to their corresponding O_j's.

Grant: $\forall O_j, 0 \leq j \leq N-1$, if O_j has available servers and receives requests from any inputs, it issues min$\{KO_j$, number of requests to $O_j\}$ grants to requests, starting from ga_j. The issued grants are sent to their corresponding I_i's. ga_j is updated to one beyond (modulo N) the last granted input (starting from ga_j in a circular manner) if and only if there are any grants accepted in the **Accept** phase of *the first iteration*. KO_j is updated to the number of available servers at O_j.

Accept: $\forall I_i, 0 \leq i \leq N-1$, if I_i has available servers and receives grants from any outputs, it accepts min$\{KI_i$, number of grants to $I_i\}$ issued grants starting from a_i. a_i is updated to one beyond(modulo N) the last accepted output. KI_i is updated to the number of available servers at I_i.

Figure 4 shows how kDRR_RGA works for a 4×4 switch with saturated load. At the start of switch slot 0, assume $\forall 0 \leq i \leq 3$, $ga_i = 0$, $\forall 0 \leq j \leq 3$, $a_j = 0$. Then after the scheduling, we get $ga_0 = 2, ga_1 = 2, ga_2 = 0, ga_3 = 0$, and $a_0 = 2, a_1 = 2, a_2 = 0, a_3 = 0$. Due to the desynchronization of issued grant pointers, we get a perfect 2-matching at switch slot 1 and thereafter. Under saturated load, kDRR_RGA actually adapts to a time-division multiplexing scheme.

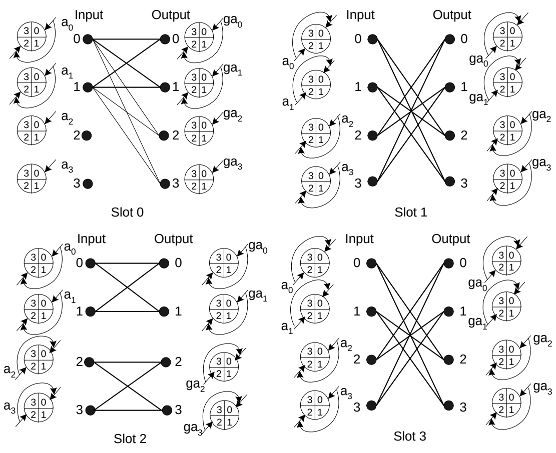

Fig. 4. Illustration of desynchronization effect of request pointers of kDRR_RGA.

kDRR_RGA has the following properties.

Property 1: At each output, due to the property of round-robin, the lowest priority element is set as the last input that accepts its issued grant in the first iteration.

Property 2: Under saturated load, all VOQs with a common output have the same throughput. The issued grant pointer moves to each input in a fixed order (every $\frac{N}{K}$ switch slots), thus providing each with the same throughput.

Property 3: kDRR_RGA converges in at most $N - K + 1$ iterations. Convergence means that either there is no more requests or all input/output ports are matched, i.e., a maximal K-matching is founded. The explanation is as follows. We use $N(R)$ to denote the total number of requests. There two cases: (1) If $N(R) \leq K^2$, then all requests will be accepted in one iteration. (2) If $N(R) > K^2$, the maximal K-matching size is at most NK. The first iteration will accept at least K^2 requests. The last iteration will accept at least 1 request. In other iterations, at least K requests will be accepted. Thus, the total number of iterations needed is at most $\lfloor \frac{NK-K^2-1}{K} \rfloor + 2$, which is given by $N - K + 1$.

Figure 5 shows an example for an 8×8 switch with 2 servers at each input/output port with saturated load. In switch slot 0, kDRR_RGA takes 4 iterations to converge. It takes 3 and 2 iterations for kDRR_RGA to converge in switch slot 1 and 2 respectively. After switch slot 3, all arbitration components have become totally desynchronized and kDRR_RGA converges in a single iteration.

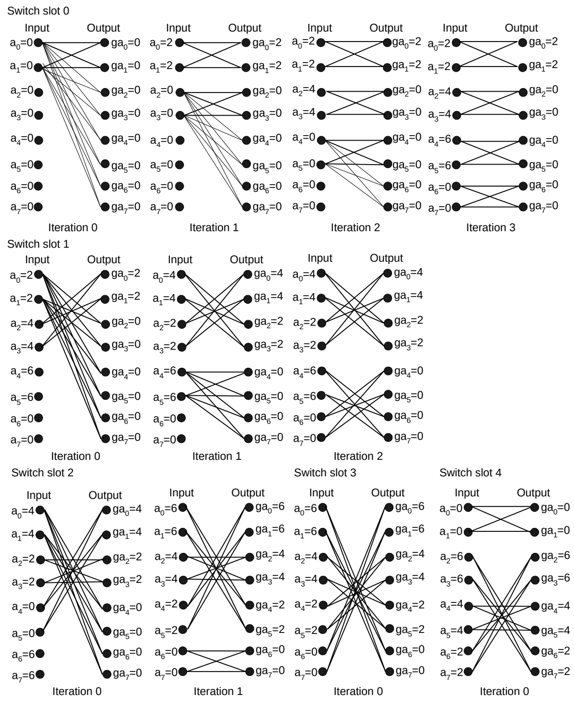

Fig. 5. Example of the number of iterations to converge for an 8×8 switch with saturated load

4 Performance Evaluation

In the following, we will evaluate the performance of kDRR algorithms in terms of average transit time under uniform traffic. Before we present simulation results, we first give an analysis of the minimum average transit time on the given switch model.

4.1 Analysis

The minimum average transit time is obtained under such a situation that the service rate to each VOQ is proportional to its arrival rate. Under uniform traffic, each VOQ has the same average arrival rate. Thus, the minimum average transit time under uniform traffic is achieved with a scheduling algorithm that could serve each VOQ equally. An ideal case is that each VOQ is associated with a deterministic server, which can transmit 1 ECP (or N' cells) per switch slot.

The average transit time through the switch is the sum of the average response time at the input port and at the output port. The average response time of a cell at the input port has three components:

(1) The residual service time of cells to be transmitted, it is half a switch slot for deterministic server in average.

(2) The waiting time of an incoming cell. It is computed as follows.

The input sources to each input port are N independently, identically distributed (i.i.d.) Bernoulli sources. Let p denote the probability of each input source to generate a cell during each cell slot, where $0 \leq p \leq 1$. For uniform traffic, each cell has a probability $\frac{1}{N}$ to enter a particular VOQ. Each switch slot contains N' cell slots. The arrival process of N input sources in a switch slot is equivalent to $N'N$ sources generating cells in one cell slot. The prob. of n arrivals to a particular VOQ during a switch slot is given by:

$$\forall n \in [0 \cdot \cdot N'N], P_n = \binom{N'N}{n} \left(\frac{p}{N}\right)^n \left(1 - \frac{p}{N}\right)^{N'N-n}.$$

Then we have the average arrival rate as $\lambda = \sum_{n=0}^{N'N} nP_n = N'N\frac{p}{N} = N'p$. For large $N'N$ and small $\frac{p}{N}$, we can use a poisson process with arrival rate λ to approximate this binomial distribution [12]. Since the average service rate to a VOQ $\mu = N'$, the average load is determined as $\rho = \frac{\lambda}{\mu} = p$.

This particular VOQ can be modelled as an M/D/1 queue in which the average number of cells in queue is given by

$$E(N) = \frac{\rho^2}{2(1-\rho)} = \frac{p^2}{2(1-p)}.$$

From Little's law, we get the waiting time of a cell as

$$E(T) = \frac{E(N)}{\lambda} = \frac{p}{2N'(1-p)}.$$

(3) The service time for the incoming cell, which is another switch slot.

Thus, the average response time of a cell at the input port is given by

$$T_{in} = \frac{1}{2} + E(T) + 1 = E(T) + \frac{3}{2}.$$

The average response time at the output port is obtained with the following assumptions. Under uniform traffic, the probability of a cell going to all LCs is the same. Each OQ receives $\rho N'$ cells during each switch slot and these cells are sent out with a constant rate of one cell per cell slot, which is $\frac{1}{N'}$ switch slot. The average response time at the output port is thus computed by:

$$T_{out} = \frac{1}{\rho N'} \sum_{i=1}^{\rho N'} \frac{i}{N'} = \frac{1}{pN'^2} \frac{pN'(pN'+1)}{2} = \frac{p}{2} + \frac{1}{2N'}.$$

The average transit time through the switch (in switch slots) under M/D/1 model is given by

$$T_{in} + T_{out} = \frac{p}{2N'(1-p)} + \frac{3}{2} + \frac{p}{2} + \frac{1}{2N'}.$$

Measured by cell slots, the average transit time is $N'(T_{in}+I_{out})$, i.e.,

$$T_{M/D/1} = \frac{p}{2(1-p)} + \frac{3N'}{2} + \frac{pN'}{2} + \frac{1}{2}. \quad (1)$$

4.2 Simulated performance of kDRR algorithms

Simulations have been done for kDRR algorithms under uniform traffic for switch size ranging in 4, 8, 16 and 32 with different number of servers per port and iterations allowed set as 1, 2, 4 and unlimited. Each switch slot equals to the switch size number of cell slots, that is, $N' = N$. Two traffic models are used in these simulations: Bernoulli traffic and bursty traffic. In the following, we present simulation results with the example of 16×16 switch. Without loss of generality, in our simulations, all pointers in kDRR algorithms are initialized randomly.

Figure 6 shows the performance of one iteration kDRR algorithms with uniform bernoulli arrivals in terms of average transit time measured by the number of cell slots. We observe that kDRR_RGA and kDRR_RG perform almost the same under uniform traffic. The average transit time of both kDRR_RGA and kDRR_RG improves with the number of servers at each port increasing. And we find that both kDRR_RGA and kDRR_RG with 16 servers per port perform close to $T_{M/D/1}$ given in Equation (1).

Since kDRR_RGA and kDRR_RG perform equivalently under uniform traffic, in the following we evaluate their performance with the example of kDRR_RGA. Figure 7 shows the average transit time vs. load of iterative kDRR_RGA algorithm with $K = 2, 4$ and 8 and 1, 2 and 4 iterations. For $K = 16$, one iteration is enough to find the maximal K-matching. For other cases, iterative kDRR_RGA does reduce the average transit time and improve the throughput for all server settings under heavy load. With 2 and 4 iterations, kDRR_RGA achieves 100% throughput with all server settings.

We then study the performance of kDRR_RGA under bursty traffic using 2-state modulated Markov-chain sources [13]. Figure 8 illustrates the performance of iterative kDRR_RGA under bursty arrival with average burst length ranging from 16, 32, and 64 for a 16×16 switch with $K = 8$ and 1, 2, and 4 iterations. As we can see, the increased number of iterations leads to higher throughput and lower average transit time at high load while the increased average burst length increases the average transit time.

Notice that kDRR_RGA is different from the multiSLIP algorithm presented in [9]. In multiSLIP, each output port server employs a round-robin arbiter, each one keeps its own pointer. While in kDRR_RGA, each outport port uses one round-robin arbitration component. Due to the property of round-robin, kDRR_RGA eliminates the problem of output server pointer synchronization that multiSLIP might encounter, especially when the number of servers is large. Here server pointer synchronization means that more than one server's arbiter pointers pointing to the same position. Figure 9 compares the performance of kDRR_RGA and multiSLIP with 1 iteration for 2, 4, 8 and 16 servers per port. As we can see from the figure, kDRR_RGA performs better than multiSLIP at low loads for all server settings and high loads for $K = 8$ and 16. It can be shown that with more iterations, multiSLIP performs more close to kDRR_RGA. However, as shown in Figure 10, multiSLIP needs more number of iterations to converge than kDRR_RGA. Averagely kDRR_RGA converges in less than $\log_2 N$ iterations for $K \neq N$. For $K = N$, it converges in one iteration.

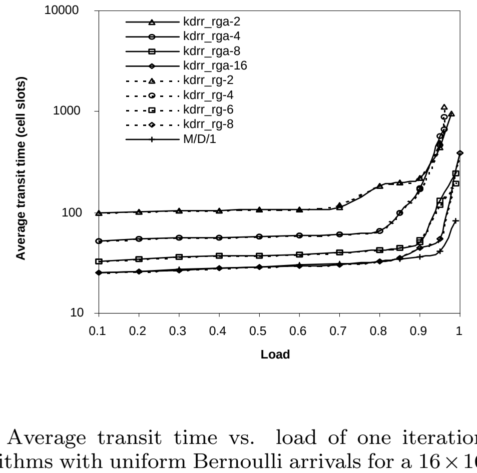

Fig. 6. Average transit time vs. load of one iteration kDRR algorithms with uniform Bernoulli arrivals for a 16×16 switch.

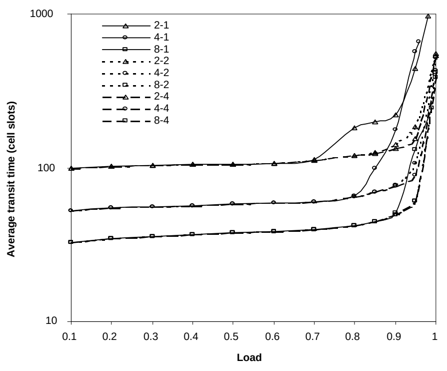

Fig. 7. Average transit time vs. load of kDRR_RGA with multiple iterations with uniform Bernoulli arrivals.

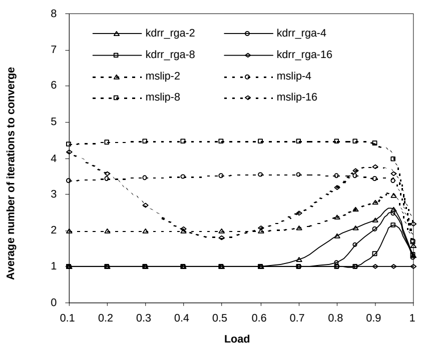

Fig. 10. Average number of iterations for convergence vs. load of kDRR_RGA and multiSLIP with uniform Bernoulli arrivals.

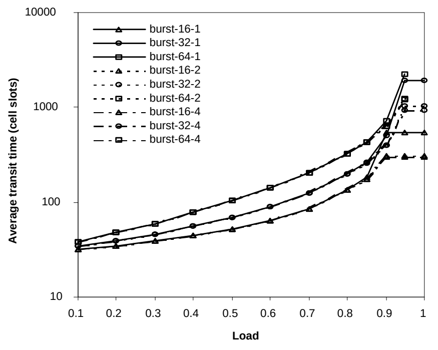

Fig. 8. Average transit time vs. load of kDRR_RGA with multiple iterations with bursty arrivals.

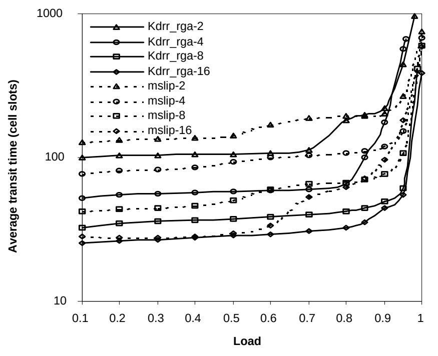

Fig. 9. Average transit time vs. load of kDRR_RGA and multi-SLIP with one iteration under uniform Bernoulli arrivals.

5 Hardware Implementation of kDRR algorithms

An important property of an efficient scheduling algorithm is simple to implement. In this section, we consider the complexity of implementing kDRR algorithms in hardware. For both kDRR_RGA and kDRR_RG, we need assign a round-robin arbitration component for each input/output port. Since the implementation of kDRR_RGA and kDRR_RG is symmetric, in the following we will illustrate the hardware implementation scheme using the example of kDRR_RGA.

One possible design of a round-robin port arbitration component is employing the programmable priority encoder(PPE) proposed in [10]. Since the PPE can only make one selection each time, we have to run up to K times to pick up K requests. The time complexity of one iteration scheduling is $2K$ times delay of the PPE. This is not fast enough for high speed packet switches.

We propose our design of a round-robin port arbitration component, which could select K requests in one time. The basic building block of a port arbitration component is a programmable k-selector, which was discussed in [15] in detail. Figure 11 shows how $2N$ port arbitration components and a state update logic together with a $2(N^2 + N)$-bit state memory are interconnected to construct a kDRR_RGA scheduler for an $N \times N$ switch. At the start of each switch slot, the scheduler receives an N-bit 0-1 request vector(0 as no requests, 1 as non-zero requests) from each input port. The one iteration kDRR_RGA scheduler operates as follows.

Step 1: Each grant arbitration component selects up to K unresolved requests from the transposed request vector. The issued grants are sent to N accept arbitration components.

Step 2: Each accept arbitration component selects up to K issued grants. These grants are saved into a decision register and passed to the state memory and update logic, where the issued grant pointers are updated.

For an iterative kDRR_RGA scheduler, the arbitration components used are almost identical to those used for a one iteration kDRR_RGA scheduler except the following differences: (1) The request matrix should be updated after each iteration. (2) The number of available servers at each arbitration component should be updated after each iteration. (3) Once an input/output is matched, its arbitration component should be disabled in subsequent iterations of the same switch slot. These three modifications make an

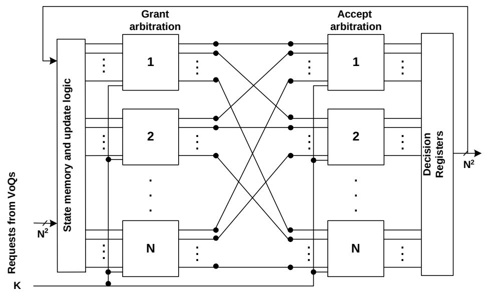

Fig. 11. Block diagram of a one iteration kDRR scheduler for an $N \times N$ switch.

Design	N=8	N=16	N=32	N=64
multiSLIP ($k = 1$)	13.8	17.5	30.0	39.9
kDRR_RGA	20.6	33.5	49.0	66.6
Improvement of kDRR_RGA over multiSLIP when $k = N/2$	62.7%	76.1%	89.8%	94.8%

TABLE I

Timing improvement of kDRR_RGA arbitration component over multiSLIP arbiter.

iterative kDRR_RGA scheduler a little bit complex than a one iteration kDRR_RGA scheduler.

Table I compares the latency of the multiSLIP arbiter (using PPE logic) and kDRR_RGA arbitration component (using SHIFT_PS logic) with $K = N/2$ simulated on Altera's CPLD series ACEX1K [15], [2]. We can see that the timing improvement of kDRR_RGA arbitration component over multiSLIP arbiter is more than 60%. And the time and area complexity of kDRR_RGA arbitration component is independent of K.

6 Conclusion

The major contributions of this paper include: (1) We presented a theoretical model for the scheduling problem of the multi-server switch architecture. (2) We generalized iSLIP and DRR to the multi-server switch architecture and proposed two practical distributed scheduling algorithms, kDRR_RGA and kDRR_RG. Simulations have shown that the kDRR algorithms achieve high throughput for both Bernoulli arrivals and bursty arrivals under uniform traffic. (3) We proposed a hardware implementation scheme for kDRR algorithms. Using our programmable k-selectors, both kDRR_RGA and kDRR_RG can be implemented in hardware at high speed. Possible extension of this work are variations of kDRR algorithms, such as prioritized kDRR algorithms, weighted kDRR algorithms and other schemes with QoS support. We expect the implementation of these variations to be more complex than kDRR algorithms.

References

[1] T. Anderson, S. Owicki, J. Saxie, and C. Thacker, "High speed switch scheduling for local area networks", *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319-352, Nov. 1993.

[2] Altera, *ACEX 1K Programmable Logic Device Family Data Sheet*, Sept. 2001.

[3] J. Blanton, H. Badt, G. Damm, and P. Golla, "Iterative scheduling algorithms for optical packet switches", ICC 2001 Workshop, Helsinki, June 2001.

[4] J. Blanton, H. Badt, G. Damm, and P. Golla, "Impact of Polarized Traffic on Scheduling Algorithms for High Speed Optical Switches", ITCom2001, Denver, August 2001.

[5] H. J. Chao, "A terabit IP switch router using optoelectronic technology", *IEEE Communications Magazine*, pp. 78-84, Dec. 2000.

[6] J. Chao, "Saturn: a terabit packet switch using dual round-robin", *IEEE Communications Magazine*, Dec. 2000.

[7] H. J. Chao, C. H. Lam, and X. L. Guo, "A fast arbitration scheme for terabit packet switches", *Globecom'99*, pp. 1236-1243, 1999.

[8] C. H. Paradimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-hall Inc., 1985.

[9] G. Damm, J. Blanton, P. Golla, D. Verchere, and M. Yang, "Fast scheduler solutions to the problem of priorities for polarized data traffic", *IST 2001*, Tehran, Iran, September 2001.

[10] P. Gupta, N. Mckeown, "Designing and Implementing a Fast Crossbar Scheduler", *IEEE Microelectronics*, Vol. 19, No. 1, 1999, pp. 20-29.

[11] M. J. Karol, M. G. Hluchyj and S. P. Morgan, "Input vs. output queueing on a space-division packet switch", *IEEE Transaction on Communications*, Vol. 35, No. 12, 1987, pp. 1347-1356.

[12] T. G. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation*, 3rd edition, Springer-Verlag, 2000.

[13] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches", *IEEE/ACM Transactions on Networking*, Vol. 7, No. 2, pp. 188-201, April 1999.

[14] M. Yang and S. Q. Zheng, "Sequential and parallel algorithms for bipartite K-matching problems,", manuscript.

[15] S. Q. Zheng, M. Yang and F. Masetti, "Hardware switch scheduling for high speed, high capacity IP routers", submitted to Globecom 2002.

[16] R. E. Tarjan, *Data structures and network algorithms*, Bell laboratories, 1983.

[17] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons Inc., 1997.