Distributed scheduling in input queued switches

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Distributed scheduling in input queued switches

Alessandra Scicchitano *, Andrea Bianco †, Paolo Giaccone †, Emilio Leonardi †, Enrico Schiattarella ‡
* Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria, Italy, Email: {ascicchitano}@deis.unical.it
† Dipartimento di Elettronica, Politecnico di Torino, Italy, Email: {firstname.lastname}@polito.it
‡ Gigabit Systems Business Unit, CISCO Systems, San José, USA Email: {eschiatt}@cisco.com

***Abstract*— Dealing with RTTs (Round Trip Time) in IQ switches has been recently recognized as a challenging problem, especially if considering distributed (multi-chip) scheduler implementation which are suited to reduce the hardware complexity in very large, high-speed, switches. Traditional iterative three- or two-phase scheduling algorithms are based on a monolithic implementation, thus allowing instantaneous information exchange among input and output selectors to determine a matching. Multi-chip implementation imply that information exchange among inputs and outputs is delayed by an inter-chip latency. This delay requires non-trivial modifications to scheduling algorithms to allow a fully distributed implementation while keeping good performance. We propose a new scheduling algorithm, named SRR (Synchronous Round Robin), which is suited to a fully distributed implementation and provides good performance if compared with more complex, non fully distributed, previously proposed scheduling algorithms.**

I. Introduction

Despite the fact that synchronous slotted IQ (Input-Queued) switches have been proposed as an innovative architecture for high-speed switches many years ago, the interest of the research community in this type of architectures is still significant. Indeed, IQ switches are suited for several application domains, such as traditional routers/switches, SANs (Storage Area Networks), and HPC (High-Performance Computing) interconnects; in most of these application domains, a large number of ports and high line rates are dominant.

To obtain performance close to output queued switches, IQ switches rely on a VOQ (Virtual Output Queued) architecture at inputs for unicast traffic, as shown in Fig.1: in an $N \times N$ switch, data are stored at each input port in N separate queues, depending on data destination ports. The most challenging problem to be solved in IQ architectures is the definition of a proper scheduling algorithm: to transfer data from VOQs to output ports, at every time slot, a matching between inputs and outputs must be determined to avoid output and input contention, since no buffers are available at output ports and no internal speedup is available in the switching fabric and in memory access speed. The task of the centralized scheduler is to collect information from VOQs, to determine a matching and to configure the switching fabric in each time slot to connect input VOQs to outputs. Increasing high rates imply shorter time slots, thus requiring simple scheduling algorithms, especially in switches with a large number of ports. Indeed, theoretically throughput-optimal solutions such as MWM (Maximum Weight Matching) or MSM (Maximum Size Matching) are not practically feasible; thus, several heuristic algorithms have been proposed [1]–[5].

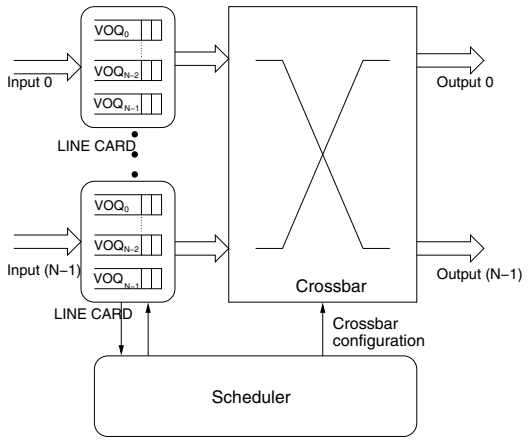

Fig. 1. IQ architecture with a centralized scheduler.

A very popular solution to determine a heuristic matching is to devise parallel, iterative matching algorithms based on either a three-phase (request-grant-accept) [3], [4] or two-phase (request-grant) [5] scheme. In three-phase schemes, in the request phase, inputs issue requests to outputs for data stored in the VOQs. In the grant phase, outputs solve request contentions independently, by choosing a single request to grant. If needed, inputs, independently, solve contentions among multiple received grants. In two-phase schemes, each input issues a single request only to a given output. As such, no accept phase is needed, since no grant contention may arise at inputs. However, a proper request must be chosen at each input and request contention must be solved as in three-phase schemes. Note that an added advantage of two-phase schemes is the reduction in the information exchange among input and outputs, since each input sends a single request to a chosen output in a given time slot, whereas three phase schemes in general rely on a complete view of input VOQ occupancy at outputs, thus requiring higher signalling bandwidth. Iterations are fundamental to improve the matching and obtain good performance.

Request and grant contentions in three-phase schemes, as well as the choice of the request to send and request contentions in two-phase schemes, are typically solved using N input and N output selectors (labelled IS and OS respectively

1-4244-0353-7/07/$25.00 ©2007 IEEE

in the remainder of the paper), exploiting a round-robin mechanism based on pointers kept at inputs and outputs: the input (output) selector chooses the first eligible output (input) in a round robin fashion, starting from the position indicated by the pointer. Pointer de-synchronization, which depends on the rule followed to update the pointer index, is crucial to obtain good performance.

Recently, several researchers have addressed the problem of scheduling algorithms in input queued switches when considering round trip time latencies in the scheduling process. In [6] round trip latencies are introduced by the need of addressing multi-rack implementation in very large switches. Multi-rack implementation implies that the physical distance between line-cards and the switching fabric is non negligible with respect to the time slot. As such, performance of a centralized scheduler based on the classical iterative three-phase (request-grant-accept) scheme are shown to degrade for large physical distances; a solution to cope with this problem is proposed, being based on a differential signalling scheme and on a slight increase of the scheduler complexity, which is assumed to keep track of VOQs state.

The centralized single-chip implementation of scheduling algorithms is largely dominant; however, scalability problems may arise for very large high-speed switches [7]. When looking at multi-chip implementations, device separation implies that decision taken by input/output selectors belonging to different devices are known only after an inter-chip latency, named RTT (Round Trip Time) in the remainder of the paper. As such, as shown in Fig. 2, information critical i) to determine the matching, ii) to update the pointers and iii) to issue new requests, is delayed by the inter-chip latency, causing performance degradation. RTTs may be significant with respect to the time slot. Indeed, at high speed the time slot is rather short (13ns at 40Gbit/s for a 64bytes packet) if compared with inter-chip communication latencies which include propagation delays, data serialization and pin sharing which may be required to overcome the I/O pin count limit. We assume that RTTs are measured in slots in the remainder of the paper.

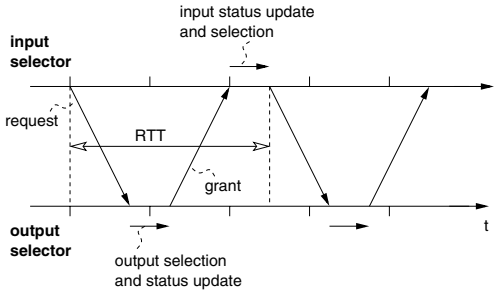

Fig. 2. Round trip time between input and output selectors.

We mainly focus on the problem of inter-chip latency, and we study scheduling algorithms able to cope with performance degradation induced by RTTs, without increasing too much the scheduler complexity. More precisely, we introduce a novel scheduler, named SRR (Synchronous Round Robin), which was previously studied in the context of WDM ring based networks with electronic buffering [8]. We show that SRR provides good performance while avoiding many of the problems of previously proposed two-phase algorithms [7].

The remainder of the paper is organized as follows: Section II states the problems, challenges and solutions for distributed scheduler implementation. Section III describes previously proposed scheduling algorithms and the newly proposed SRR scheme. Section IV present performance results obtained by simulation. Finally, Section V ends the paper and suggests possible future research directions.

II. Distributed Schedulers

Let us focus on iterative schedulers, based on three-phase or two-phase scheme information exchange among inputs and outputs. These schedulers adopt selectors for each output to choose among multiple requests received by inputs and for each input to either choose among multiple grants received by outputs or to select a proper request to be issued in a given time slot. Scheduler distribution entails partitioning the selectors used to determine a heuristic matching over physically separated devices. In a monolithic implementation, all selectors are tightly coupled and decisions taken at inputs (outputs) are immediately available to outputs (inputs). When dealing with multi-chip implementations, the communication latency between devices implies that algorithms devised to run under the hypothesis of having all the scheduling state information available may not be optimal. Indeed, information needed to update the pointer status or to issue new requests may be known with a delay of some tens of cell time. Performance degradation and loss of fairness were already shown to be a possible problem.

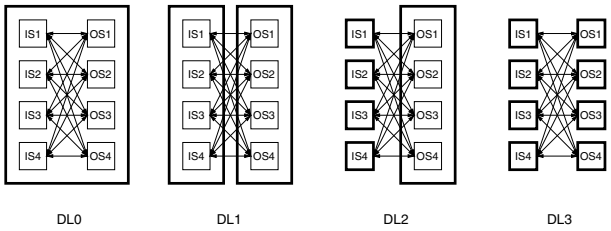

Fig. 3. Scheduler distribution level: DL0 is a monolithic implementation, DL3 is a fully distributed implementation.

Different levels of distribution could be envisioned, which yield to a different number of physically separated devices, as shown in Fig. 3. The first obvious solution to reduce the scheduler complexity is to implement the scheduler in two separate devices, each containing respectively N input and N output selectors. This allows to roughly divide by two the scheduler hardware complexity. Another extreme is to distribute the scheduler over $2N$ separate devices, each device implementing one selector. This is referred to as fully distributed solution, and we are mainly referring to this solution in this work, which allow to obtain a hardware complexity

reduction by a factor of N. As an intermediate step, in [7] a further possible solution is proposed: the N input selectors are physically separated in N devices, whereas all N output selectors are implemented in a single device. This solution has a major drawback: it reduces the hardware complexity by a factor of two only. However, having the output selectors in a single device permits coordination among output selectors with no delay. In particular, as explained later, this permits the implementation of two-phase schedulers with more iterations in a single cell time, thus preserving good performance for increasing RTTs. Although we focus on a centralized multi-chip scheduler implementation, the fully distributed multi-chip architecture could be adopted also when ISs and OSs reside on different line-cards, allowing a completely non-centralized scheduler implementation.

In this paper, we propose a new scheduler, named SRR (Synchronous Round Robin), which is suited to a fully distributed environment, thus allowing a hardware scheduler complexity reduction by a factor of N. At the same time, we show that SRR, without any need of iteration, is able to provide performance comparable with previously proposed two-phase schedulers running with log$N$ iterations, while preserving the signalling reduction property typical of two-phase algorithms. Finally, as it will become clearer later, in contrast with previously proposed schedulers, SRR does not require to keep any state information, whose size typically increaseas linearly with the RTT.

III. Scheduling algorithms with RTT latencies

Dealing with RTTs among devices has a profound impact on scheduler design. In [7], the proposed two-phase scheduler is a direct extension of the DRRM (Dual Round Robin Matching) scheduler [5], originally conceived for a monolithic implementation, to a distributed environment.

Let us briefly summarize DRRM behavior, focusing on an enhanced version of DRRM which achieves lower delays, thanks to a modified pointer update rule similar to that used in FIRM (Fcfs in Round robin Matching) [4]. In every iteration, first a request is sent by any unmatched input to the first unmatched backlogged output in the round-robin order starting from the current request pointer position. If an output receives more than one request, it grants the one that appears first in the round-robin order starting from the current position of the grant pointer. Request pointers are updated, in the first iteration to point to the output selected in the request phase, and further to one position (modulo N) beyond the output selected if and only if the request is granted in the first iteration. Grant pointers are updated to one position (modulo N) beyond the input granted in the first iteration.

In a monolithic implementation, all decisions are taken, and known, in a single time slot, and pointers are updated accordingly. In a distributed implementation, first, request information is delayed by RTT/2 (assuming symmetric RTTs) and the request pointer update cannot be performed immediately, since grants will be available RTT/2 slots later. A straightforward extension of DRRM to a distributed scenario would imply that requests to be issued in the next time slot are based on pointer positions not updated, thus breaking the round-robin de-synchronization mechanism and leading to throughput degradation. Moreover, request selectors would not be able to accurately know the number of underway grants, thus negatively affecting request decisions.

The distributed extension of DRRM proposed in [7] is based on the following ideas. Let us focus on a single iteration case. The key idea to keep the pointer de-synchronization is to ensure that every pointer is updated at most once every RTT slots. As such, each input and output selector keeps a distinct (request and grant) pointer for every RTT slots. Traditional pointer update rules are used: Request pointers are only updated when the corresponding grant arrives, one RTT after issuing the corresponding request, whereas grant pointers are updated immediately after issuing a grant, since issued grants are accepted by definition. In other words, with respect to a monolithic implementation, RTT staggered schedulers are running in pipeline dealing with requests and grants.

This implies that the scheduler complexity (in terms of required pointers) increases linearly with RTT, since RTT pointer registers are required per selector. An additional counter (modulo RTT) is needed to indicate the current pointer to be used, whereas the combinatorial selection logic does not need to be duplicated, since a multiplexer to select the proper register among the RTT registers is enough to permit a correct behavior.

Another issue is related to pending requests. Indeed, only after RTT slots it is possible to know whether an issued request was granted. In the meantime, input selectors should issue further requests. If the number of submitted requests exceeds the number of enqueued cells, it may happen that a slot is reserved for a VOQ that will become empty by the time the grant is received at the input selector, thus wasting system resources. To solve this problem, a PRC (Pending Request Counter) per VOQ plus a request history per input selector are introduced [7]. Basically, new requests are issued only if the number of pending requests is smaller than the number of cells currently stored in the corresponding VOQ. This choice, which further complicates the selector design, is fundamental to obtain good performance at low loads or under heavily unbalanced traffic.

Further problems are related to the issue of dealing with more iterations in a single time slot (see [7] for details). However, since a fully distributed implementation is the goal of this paper, we disregard issues related to iterations, which can be reasonably used only if all output selectors share the same physical device. Indeed, iterations are based on the knowledge of the results of previous iterations in the same time slot; in a fully distributed scheduler with RTTs, knowledge of results of previous iterations requires RTT time slots, thus basically preventing the possibility of iterating in this scenario.

A. Synchronous Round Robin

Let us now describe the SRR scheduler, initially disregarding issues related to RTT for simplicity. The SRR scheme

is based on a cyclic, TDMA-like, preferential scheduling of VOQs. This preferential scheduling is obtained by logically numbering the slots with an incremental counter s, ranging from 0 to $N-1$, i.e., a modulo N counter. Slots are logically organized in frames, named SRR frames; each frame comprises N slots.

Let us describe the input and output selectors behavior at time slot s:

1) the input scheduler associated to input i preferentially selects for a transmission the VOQ with destination output $|i+s|_N$. In other words, a preferential request is issued for output $|i+s|_N$.
2) if the preferential VOQ is empty, a request for the longest VOQ is attempted; ties among VOQs are broken according to a round-robin scheme.

Output schedulers grant the preferential request, if issued; otherwise, a randomly chosen request is granted among non-preferential conflicting requests.

Other possible solutions, not pursued in this paper, exist to break ties among VOQs at input selectors, such as random or round-robin choice among non-preferential VOQs; however, the longest VOQ choice provides the best compromise between complexity and performance, as discussed later. Note that the random choice among conflicting non-preferential requests at output selectors may be not the most natural choice. Indeed, given that input selectors choose non preferential requests exploiting a longest queue first algorithm, output selectors could select the request corresponding to the longest VOQ among conflicting requests. However, this would require an increase in signalling complexity, since VOQ lengths should be sent to output selectors, and also a complexity increase at output selectors, since conflicting request should be compared according to their length. Selecting longest queues at inputs is easier, since queues could be kept simply ordered by length; indeed, in a given time slot, at each input at most one departure and one arrival can occur. Moreover, performance results show that the benefit of the longest queue selection at outputs is marginal for both balanced and unbalanced traffic.

Regardless of the fact that the request is granted or not, a new selection is made in the next time slot, according to the above described algorithm. In summary, for low switch loads SRR behaves similarly to a single-queue FIFO strategy. At high loads, when all queues are always non-empty, the SRR preferential scheduling deterministically orthogonalizes input request attempts, so that a single preferential request is received by any output in a given time slot. More precisely, for network loads larger than or equal to the channel capacity, input selectors behave exactly like in a Time Division Multiple Access (TDMA) scheme: during a SRR frame, whose length is equal to N slots, all inputs have exactly one access opportunity for transmissions toward each output, and they exploit this access opportunity deterministically, thereby avoiding potential conflicts at outputs. In this sense, SRR is throughput optimal under overloaded uniform traffic conditions.

A nice property of SRR is that it can be used without any modification i.e., with no complexity increase, to deal with RTTs. The only difference is that grants will be received one RTT later with respect to request issue, thus negatively affecting delays. However, we will show that even without exploiting any pending request counter, performance at low loads are comparable with those of iterative algorithms exploiting pending request counters.

IV. Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results are assessed by running experiments with an accuracy of 1% under a confidence interval of 95%. We compare SRR with the distributed extension of DRRM presented in [7]. The switch has $N = 16$ inputs and outputs running at the same speed, and is loaded with either Bernoulli or Bursty traffic with geometrically distributed burst sizes with an average burst size of 10 cells; cells in the burst are all directed to the same output. Performance indices are either average delays vs normalized switch throughput or maximum achievable normalized throughput in overload. SRR performance are reported as solid lines with black dots, the modified version of DRRM, which accounts for multiple pointers to deal with RTTs, is labeled MP (Multi Pointer) and plotted using dashed lines; different symbols refer to a variable number of iterations.

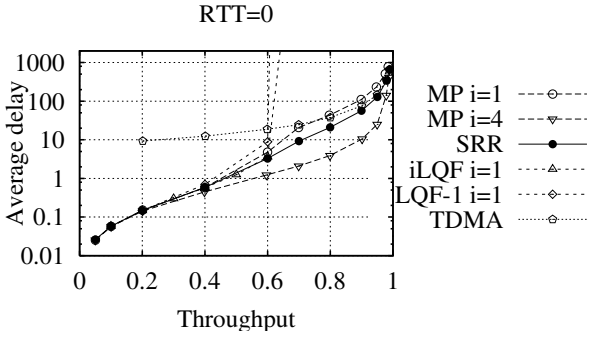

Fig. 4. Performance comparison between SRR, MP, iLQF and TDMA under uniform Bernoulli traffic in a monolithic scheduler implementation (RTT=0).

Let us first examine, in Fig.4, a scenario in which RTT=0, which corresponds to the traditional single-chip implementation of schedulers, under uniform Bernoulli traffic. Besides DRRM and SRR, we plot also a TDMA scheme, iLQF (Longest Queue First) [9] with one iteration, and LQF-1, a simplified version of iLQF with one iteration where inputs send a single request only (for the longest queue) per time-slot to outputs. Note that the TDMA scheme corresponds to SRR using the preferential scheduling only whereas LQF-1 corresponds to SRR using the non-preferential scheduling only. Clearly, the preferential scheduling scheme of SRR is fundamental to obtain good performance results at high loads: indeed, iLQF with $i = 1$ iteration saturates at 0.65, whereas LQF-1 saturates at 0.61. At low loads the non-preferential

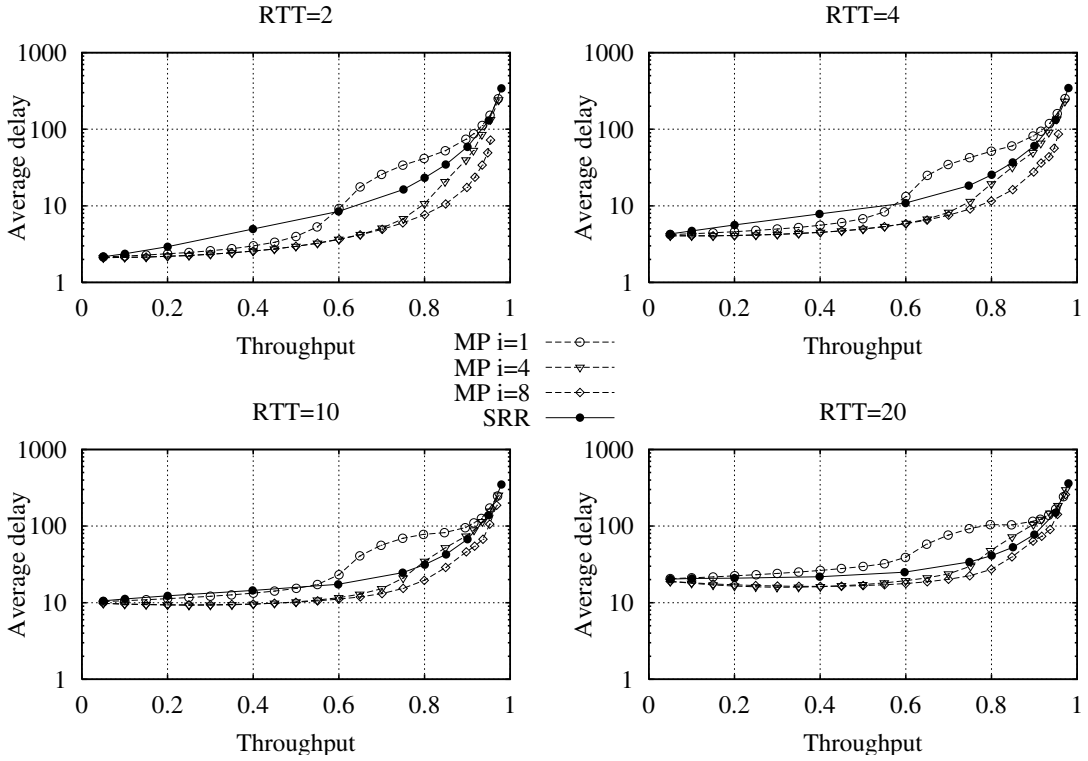

Fig. 5. Performance comparison between SRR and MP under uniform Bernoulli traffic for variable RTTs.

scheme of SRR becomes dominant, so that performance are close to those of modified DRRM and much better than those of a pure TDMA.

In Fig.5 we report delays as a function of the switch throughput for variable RTTs under uniform Bernoulli traffic. No throughput limitations are observed for both SRR and modified DRRM; SRR shows remarkably low delays, improving performance as RTT increases. Only a slight delay impairment can be noticed for low-medium loads and small RTTs; this is due to the missing pending request counters, which are instead used in the modified DRMM. Recall that pending request counters are fundamental to avoid issuing too many request for the same VOQ; indeed, if the number of requests issued exceeds the number of enqueued cells, which may happen at low-medium loads and for large RTTs, it may happen that the VOQ becomes empty by the time the grant is received, thus wasting system resources. Indeed, modified DRRM without pending request counters performs much worse, as shown in Fig.6, where SRR clearly outperforms the modified DRRM. Moreover, recall that SRR, besides not exploiting any additional counter, does not require any iteration, thus allowing a fully distributed implementation; nevertheless, delay performance are comparable with those of modified DRRM with $\log N = 4$ iterations, and improve with increasing RTTs. Minor differences exist in this scenario if the non-preferential VOQ choice at inputs in SRR is done on a round-robin basis instead of using a longest queue choice.

Similar performance are shown by SRR for Bursty traffic, as reported in Fig.7. Delay properties are remarkable, since SRR shows much better performance than modified DRRM with $i = 1$ iteration also under correlated traffic. Note that SRR with the random choice of non-preferential VOQs at input selectors, labelled SRR-RND in the plot, show much worse performance than SRR for medium loads, thus justifying the choice of the longest VOQ when the preferential queue is empty.

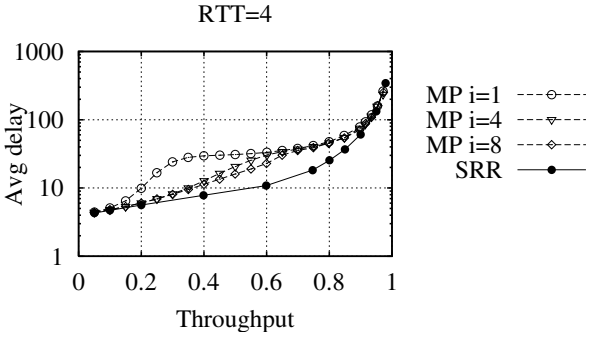

Fig. 6. Performance comparison between SRR and MP under uniform Bernoulli traffic when MP is not using the PRC counters.

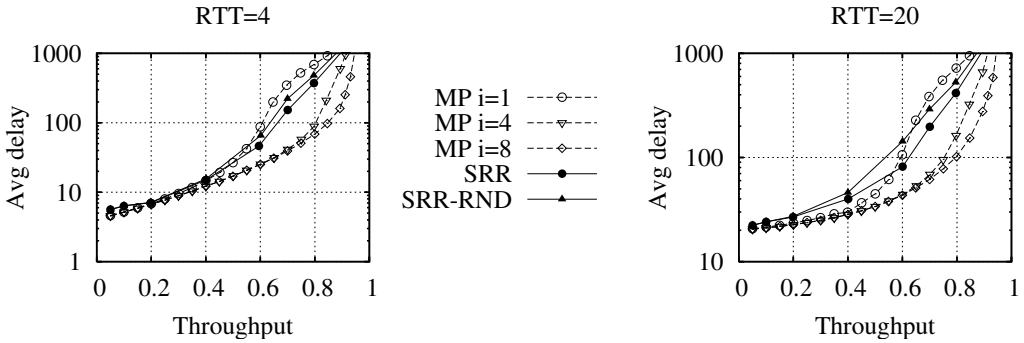

Fig. 7. Performance comparison between SRR and MP under uniform Bursty traffic for variable RTTs.

V. Conclusions

We deal with the problem of fully distributed multi-chip scheduling implementation in input queued switches. Multi-chip implementation implies that i) decisions taken by input and output selectors should be independent, being the selectors realized in different devices and ii) any information exchange among selectors implies a RTT delay, which may be larger than few tens of slot time.

The newly proposed scheduler, named SRR, i) is suited to a fully distributed implementations, ii) does not require any complexity increase as a function of increasing RTTs, iii) does not require any iteration to improve matching selection.

SRR shows performance comparable with those of a previously proposed distributed scheduler, a modified version of DRRM able to deal with RTT among devices. This is a remarkable achievement, since the modified DRRM scheme is not suited to a fully distributed implementation, requiring i) all output selectors in the same device to permit iterations, ii) a number of pointers and counters linearly increasing with RTTs. The only SRR penalty is the need of keeping ordered by queue length the VOQs at input selectors, a relatively easy task given that at most one cell can arrive and at most one cell can depart in each time slot.

Acknowledgment

This work was performed in the framework of the Italian PRIN project BORA-BORA and partly supported by IBM under joint research contract no.93830515.

References

[1] M.Ajmone Marsan, A.Bianco, E.Leonardi, L.Milia, "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches", IEEE Transactions on Communications, Vol.47, No.12, pp.1921-1933, Dec. 1999

[2] A.Smiljanic, R.Fan, G.Ramamurthy, "RRGS-round robin greedy scheduling for electronic/optical terabit switches", IEEE GLOBECOM'99, Rio De Janeiro, Brazil, Dec. 1999

[3] N.McKeown, "The iSLIP scheduling algorithm for input-queued switches", IEEE/ACM Transactions on Networking, v.7, n.2, pp.188-201, Apr.1999

[4] D.Serpanos, P.Antoniadis, "FIRM: A class of distributed scheduling algorithms for high-speed ATM switches with multiple input queues", IEEE INFOCOM'00, Tel Aviv, Israel, Mar 2000

[5] Y.Li, S.Panwar, H.Chao, "On the performance of a dual round-robin switch", IEEE INFOCOM'01, Anchorage, AK, USA, Apr 2001

[6] C.Minkenberg, "Performance of i-SLIP scheduling with large round-trip latency", HPSR'03, Turin, Italy, June 2003

[7] C.Minkenberg, F.Abel, E.Schiattarella, "Distributed crossbar schedulers", HPSR'06, Poznan, Poland, June 2006

[8] M.Ajmone Marsan, A.Bianco, E.Leonardi, M.Meo, F.Neri, "MAC protocols and fairness control in WDM multi-rings with tunable transmitters and fixed receivers", IEEE/OSA Journal of Lightwave Technology, Special issue on Multiwavelength Optical Technology and Networks, Vol.14, No.6, pp.1230-1244, June 1996

[9] N. McKeown, "Scheduling algorithms for input-queued cell switches", Ph.D. Thesis, University of California at Berkeley, 1995