Multicast Support in Multi-Chip Centralized Schedulers in Input Queued Switches

Multicast Support in Multi-Chip Centralized Schedulers in Input Queued Switches ⋆

Andrea Bianco [a,*], Alessandra Scicchitano [b]

[a] *Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy*

[b] *IBM Research, Zurich Research Laboratory, 8803 Ruschlikon, Switzerland*

Abstract

IQ switches store packets at input ports to avoid the memory speedup required by OQ switches. However, packet schedulers are needed to determine an I/O (Input/Output) interconnection pattern that avoids conflicts among packets at output ports. Today, centralized, single-chip, scheduler implementation are largely dominant. In the near future, the multi-chip scheduler implementation will be suited to reduce the hardware scheduler complexity in very large, high-speed, switches. However, the multi-chip implementation implies introducing a non-negligible delay among input and output selectors used to determine the I/O interconnection pattern at each time slot. This delay, mainly due to inter-chip latency, requires modifications to traditional scheduling algorithms, which normally rely on the hypothesis that information exchange among selectors can be performed with negligible delay. We propose a novel multicast scheduler, named IMRR, an extension of a previously proposed multicast scheduling algorithm named mRRM, making it suitable to a multi-chip implementation, and examine its performance by simulation.

Key words: Multicast, IQ switches, Multi-chip implementation.

⋆ A preliminary version of this paper was presented at the IT-NEWS 2008 conference in Venice, February 2008, under the title "Multi-chip multicast schedulers in Input-Queued Switches".

* Corresponding author. Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy. Tel. +39 011 0904098. Fax: +39 011 0904149

Email addresses: andrea.bianco@polito.it (Andrea Bianco), als@zurich.ibm.com (Alessandra Scicchitano).

1 Introduction

IQ (Input-Queued) switches were proposed as an innovative architecture for high-speed switches many years ago. However, 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.

The most challenging problem in synchronous slotted IQ architectures is the definition of a proper scheduling algorithm: To transfer fixed-size cells from input queues to output ports, at every time slot, a matching between inputs and outputs must be determined to resolve output and input contention, since no buffers are available at output ports and no internal speedup is available neither in the switching fabric nor in memory access speed. The task of the scheduler is to collect information from input queues, to determine a matching and to configure the switching fabric in each time slot to connect input to output ports. Schedulers often exploit selectors to determine the matching according to the current input queue status. Increasing high rates imply shorter time slots, thus requiring simple scheduling algorithms, especially in high-speed switches with a large number of ports.

The centralized single-chip implementation of scheduling algorithms is largely dominant; however, scalability problems may arise for very large, high speed, switches [1]. For this reason, multi-chip scheduler implementation were recently studied [1,2]. When looking at multi-chip implementation, chip separation implies that decision taken by different devices are known only after an inter-chip latency, named RTT (Round Trip Time) in the remainder of the paper. 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. This additional delay makes traditional scheduling algorithms unsuitable for the multi-chip implementation, worsening scheduler performance. Scheduling algorithms to efficiently deal with the inter-chip latency issue were previously proposed for unicast traffic [1,2].

Multicast traffic has not yet become a significant portion of the current Internet traffic. Nevertheless, its support is anyhow fundamental in routers/switches. Indeed, broadcast traffic is important in Ethernet switches both to support applications that rely on LAN broadcast capability, e.g., ARP, as well as to internally distribute forwarding tables; in SANs, multicast enables data replication, possibly on physically separated site, to improve reliability.

To the best of our knowledge, no studies were proposed to solve the scheduling

problem for multicast traffic in multi-chip scheduler implementation. Thus, we propose extensions/modifications/improvements to previously proposed multicast scheduling algorithms to cope with performance degradation induced by RTTs, without increasing too much the scheduler complexity. More precisely, we first propose extensions to well-known multicast scheduling algorithms to deal with RTTs due to inter-chip latency. Then, we introduce a novel scheduler, named IMRR (Improved Multicast Round Robin), which modifies a previously proposed scheduler named mRRM (Multicast Round Robin Matching) [4]. We show by simulation that IMRR provides good performance in a wide range of traffic scenarios, including uniform multicast traffic, multicast traffic pattern known to be difficult to schedule and traffic patterns including a mix of unicast and multicast traffic.

The remainder of the paper is organized as follows: In Sec. 2 we briefly state the problems, challenges and solutions for multi-chip scheduler implementation and discuss the related work. Sec. 3 describes previously proposed scheduling algorithms and the newly proposed IMRR scheme. Sec. 4 presents performance results obtained by simulation. Finally, Sec. 5 ends the paper and suggests possible future research directions.

2 Problem Statement and Related Work

We fix our attention on synchronous slotted IQ switches. Fixed size data, named cells, are transferred from input to output ports across a non-blocking switching fabric (e.g., a crossbar). To obtain performance close to OQ (Output Queued) switches, IQ switches rely on a VOQ (Virtual Output Queued) architecture at inputs for unicast traffic, as shown in Fig.1. This architecture permits to overcome the well known HoL (Head of the Line) blocking phenomenon. In an $N \times N$ switch, unicast cells are stored at each input port in N separate queues, one for each possible unicast cell destination.

To completely avoid HoL for multicast flows, the optimal queue architecture [3] requires a number of queues which grows exponentially as 2^N with the number N of inputs/outputs. Furthermore, a complex packet re-enqueuing scheme would be required to optimize throughput performance. Being this solution unpractical, and due to the fact that multicast traffic is today a small portion of data traffic, most researchers propose the use of a single FIFO queue for multicast traffic [4,5] at each input port. Other studies propose an architecture with $K < N$ queues and show some performance benefits due to the reduced HoL blocking [6–8]. However, in this paper we mostly consider IQ architectures with a single FIFO queue for multicast traffic.

Scheduling algorithms are needed to resolve output port contention, since no buffering is available at output ports. Unicast scheduling algorithms determine, at each

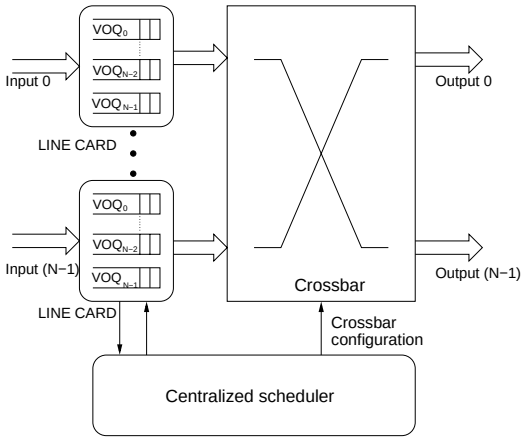

Fig. 1. IQ architecture with a centralized scheduler and VOQ architecture at inputs for unicast traffic.

time slot, a matching, i.e. an interconnection pattern among input and output ports in which each input (output) is connected to at most a single output (input). In the case of multicast schedulers, each input can be connected in a given time slot to several outputs. This does not violate the no-speedup constraint of IQ switches. Indeed, the non blocking switching fabric is assumed to have multicast capability: Each input can transfer a copy of a multicast cell to several outputs at no extra cost with respect to unicast transmission.

Optimal matching algorithms are too complex to be implemented in hardware. Thus, several heuristic algorithms have been proposed for unicast [9–13] and multicast traffic [4–8] in the past. A very popular solution to determine a heuristic matching is to devise parallel, iterative matching algorithms based on a three-phase (request-grant-accept) [11,12] or two-phase (request-grant) [4,5,13] scheme. In three-phase schemes, in the request phase, inputs issue requests to all the outputs for which at least one packet is available. In the grant phase, outputs solve request contentions independently, by choosing a single request to grant. Finally, inputs solve contentions independently by accepting one 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 at each output as in three-phase schemes. Iterations are in general fundamental to improve the matching and to obtain good performance, especially for unicast traffic. As such, the three-step or two-step procedure is often repeated several times in each time slot.

Iterative schedulers, based on three-phase or two-phase information exchange among inputs and outputs, adopt OSs (Output Selectors) to choose among multiple requests received by inputs and ISs (Input Selectors) to choose among multiple grants received by OSs. Furthermore, in two-phase schemes, ISs select a proper request to issue to OSs in each time slot. ISs ands OSs exploit a round-robin mechanism, based on pointers kept at inputs and outputs: The IS (OS) chooses the first eligible

output (input) in a round robin fashion, starting from the position indicated by the pointer.

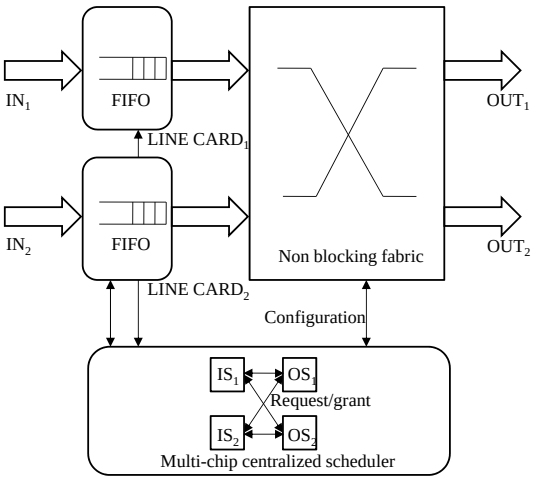

Fig. 2. 2×2 IQ architecture with a centralized multi-chip scheduler and FIFO queuing for multicast traffic.

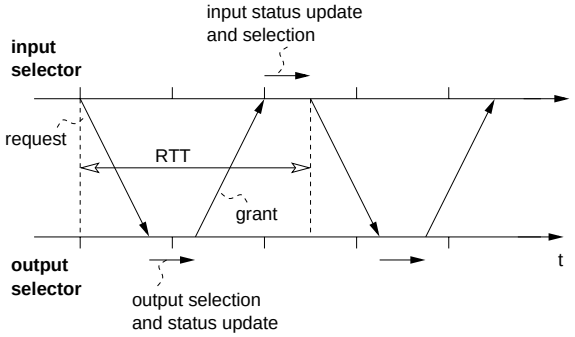

Fig. 3. Round trip time between ISs and OSs.

Scheduler distribution over several chips, needed to cope with increasing scheduler complexity in large-size switches, entails partitioning the selectors used in the centralized scheduler to determine a heuristic matching over physically separated devices, as shown in Fig.2 for a 2×2 switch. In a single-chip implementation, all selectors are tightly coupled and decisions taken at ISs (OSs) are immediately available to OSs (ISs). In a multi-chip implementation, the information exchanged among selectors, which is critical i) to determine the matching, ii) to update the pointers and iii) to issue new requests, is delayed by the inter-chip latency, named RTT (Round Trip Time), as shown in Fig. 3. 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 in this multi-chip scenario [1,2].

Different levels of selectors distribution could be envisioned, which yield to a different number of physically separated devices, as shown in Fig. 4. The first obvi-

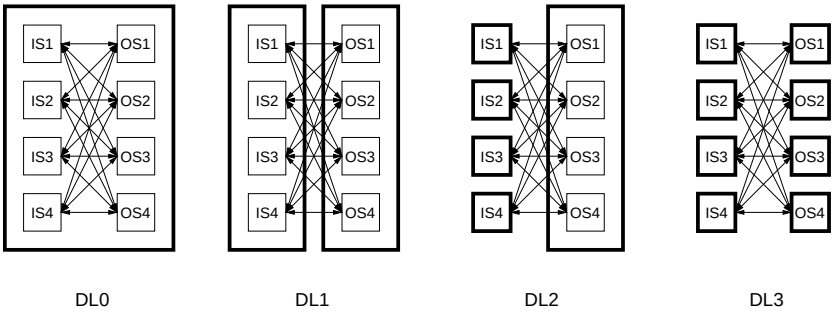

Fig. 4. Distribution levels of a centralized multi-chip scheduler in a 4×4 switch: DL0 is a single-chip implementation, DL3 is a fully distributed, multi-chip, implementation.

ous solution to reduce the scheduler complexity, labeled distribution level DL1, 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, labeled distribution level DL3, is to distribute the selectors over $2N$ separate devices, each device implementing one selector. This is referred to as fully distributed solution, which permits a hardware complexity reduction by a factor of N. As an intermediate step, in [1] 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. This permits to implement schedulers with more iterations in a single cell time, thus preserving good performance for increasing RTTs, especially for unicast traffic. Nevertheless, since the DL2 distribution level limits scheduler scalability, we focus on the distribution level DL3 only, leading to schedulers suited to a fully distributed multi-chip scenario. Indeed, although the distribution level DL3 is defined for a centralized multi-chip scheduler implementation, schedulers devised for this scenario are suited also for a distributed scheduler implementation. However, we do not further tackle this issue in the paper. As such, the reference switch architecture analyzed in this paper is depicted in Fig.2, for $N = 2$ input/output ports, where each IS/OS block is a separate chip.

One possible way to support multicast traffic is to replicate multicast cells at inputs and treat them as unicast cells. This would permit to use schedulers devised for unicast traffic, even in the context of multi-chip implementation [2], to deal with multicast traffic. However, we don't consider multi-copy as a proper solution, and we include it only as a reference case in some simulation results. Indeed, a speed increase at input ports would be required to support data replication, and a poor bandwidth utilization of the switching fabric [3] would be obtained even in very simple scenarios: For example, a broadcast flow incoming in a single input at line speed cannot be supported efficiently due to the no-speedup requirement of IQ switches, which limits the throughput to $1/N$ in an $N \times N$ IQ switch. Finally, the interesting multi-hop approach proposed in [14], which permits to distribute input load to several inputs and obtain high throughput efficiency, has a major drawback in terms of

large delays at low-medium loads, a problem which is further exacerbated by the RTT induced by the inter chip latency penalty in multi-chip implementations.

In summary, since previously proposed schedulers are not suited to the multi-chip implementation scenario, we first describe modifications to previously proposed multicast scheduler [4,5] to adapt them to a multi-chip implementation, and, second, we introduce a new multicast scheduler specifically devised for the multi-chip scenario.

3 Multicast scheduling algorithms

A multicast cell is a cell with multiple destinations, i.e., it must be transferred from a given input port to several output ports. The fanout set of a multicast cell is defined as the set of outputs to which the cell should be transferred. The cell fanout is the number of outputs in the fanout set.

Multicast scheduling implies sending a cell to several output ports in the same time slot to exploit the intrinsic multicast capability of the switching fabric. However, it may be impossible to send a copy of the multicast cell to all the outputs in the cell fanout in the same time slot, due to conflicts among cells at different inputs. Whenever a cell is transferred only to a subset of the output ports in the fanout set, i.e., a so-called fanout-splitting discipline is adopted, a copy of the multicast cell must be transmitted to the remaining output ports in successive time slot. This leads to an increase in the input load and to performance degradation. To enhance performance, most multicast schedulers try to send a copy of the multicast cell to the largest available set of outputs.

We focus on two previously proposed multicast scheduling algorithms, selected due to their ease of implementation and good performance: WBA (Weight-Based Algorithm) [5] and mRRM (multicast Round Robin Matching) [4]. Both schedulers rely on a single FIFO queue at each input for multicast. and are based on a two-phase request-grant algorithm. We briefly remind the WBA and mRRM scheduler behavior referring to a scenario where RTT=0 for simplicity.

WBA assigns weights to cells stored at inputs based on their age and fanout at the beginning of every time slot. Once weights are assigned, they are associated with the request issued by ISs. OSs choose the heaviest request among those received from inputs subscribing to it. More precisely, at the beginning of every time slot, each IS computes the weight of the new multicast cell/residue at its HOL, based on the age of the cell (the older, the heavier), and the fanout of the cell (the larger, the lighter). Then, each IS sends a request with this weight to all outputs in the HoL cell/residue fanout set. Each OS issues a grant for the request with the highest weight, independently of other OSs, breaking ties with a random

choice. Note that a positive weight should be given to cell age to avoid input starvation. However, to maximize throughput, fan-out are weighted negatively. Several weight assignments algorithm can be adopted (see [5] for details). In this paper we use the weight definition chosen in the simulator available on the Web site at http://klamath.stanford.edu/tools/SIM/: The cell weight is equal to the number of inputs minus the cell fanout plus the cell age.

The mRRM scheduler was designed to be simple to implement in hardware. A single pointer to inputs is collectively maintained by all OSs to solve contentions. Each OS selects, among the inputs that issued a request, the next input at or after the pointer, following a round robin order modulo N. At the end of the time slot, the single pointer is moved to one position beyond the first input that is served. The pointer shared among OSs makes it more likely for OSs to choose the same input port, thus reducing the negative effect of fanout splitting. However, the mRRM single pointer update rule at OSs does not permit a fully distributed multi-chip implementation, since OSs must be aware of other OSs choice to properly update the pointer value.

3.1 Novel extensions to deal with RTTs among selectors

Direct extensions of WBA and mRRM can be envisioned to deal with RTTs among input/output selectors induced by the multi-chip implementation. Since grants are received at ISs with a delay of RTT slots, a multicast cell is at the head-of-the-line for at least RTT slots, since in the best case it will be transmitted after a RTT delay, To avoid issuing multiple requests for the same HoL cell in consecutive time slots, thus wasting resources, a number of FIFO queues equal to (RTT+1) at each input port is needed. ISs choose the queue from which the request is issued according to a round-robin fixed scheme among FIFO input queues. As such, at most one request per input queue is issued in each RTT. OSs keep working as in the basic WBA and mRRM scheme.

Further extensions can be envisioned and are studied later when presenting simulation results. Indeed, the single FIFO scheme when RTT=0, and its direct extension to (RTT+1) FIFO queues when RTT > 0, do not solve the issue of HoL blocking for multicast flows. As such, in some experiments, we introduce a limited number of $k = 2 \times (\text{RTT} + 1)$ FIFO queues. In the traditional single-chip scenario (RTT=0), this translates in slightly reducing the HoL blocking thanks to the 2 available FIFO queues. In this case, whereas OSs operate as in the basic scheme, ISs choose one among the k queues according to queue weights for WBA and to a round-robin scheme for mRRM.

The introduction of more queues at inputs, either to efficiently deal with RTTs or to reduce the HoL blocking, introduces the issue of multicast flow assignment to

queues. Although several possible assignment strategies were studied in the past [7,8], in this paper we simply adopt a packet-by-packet load balancing scheme among available queues. We are aware that this may introduce out-of-sequence delivery, but we prefer to avoid studying assignment scheme in this initial work and concentrate on the issue related to dealing with RTTs.

Besides proposing a modified queue architecture and extensions to known multicast schedulers to deal with RTTs induced by inter-chip latency, we further introduce the novel IMRR (Improved Multicast Round Robin) scheduler. In the IMRR description, we initially disregard issues related to RTTs to simplify the algorithm presentation. Thus, we focus on the case of a single FIFO queue available at each input.

The pointer shared among OSs in mRRM and the use of weights to select among conflicting inputs in WBA make it more likely for OSs to choose the same input port in a given time slot, thus reducing the negative effect of fanout splitting. In IMRR, the same concept of preferential input selection through the pointer is kept as in the mRRM algorithm: At each time slot, all OSs keep a single pointer to a preferential input. However, the pointer update rule is stateless: Regardless of the granted cell, at each time slot, the pointer is increased (modulo N) according to a round robin scheme. This modification is fundamental to guarantee that the scheduler can run in the fully distributed case, since no coordination among OSs is required to update the pointer. Furthermore, ISs issue a request containing a weight equal to the fanout of the selected cell. OSs choose the request from the preferential input, if any. Otherwise, they grant the request corresponding to the smallest cell fanout. This modification further improves scheduler performance, as discussed when presenting the simulation results.

When RTT$\neq 0$, (RTT+1) FIFO queues are used at each input port. Since RTT slots are needed to (possibly) receive a grant for each request, to avoid sending multiple requests for the same multicast cell in one RTT, ISs send requests for cells belonging to different queues according to the value of the queue pointer rtt. The queue pointer rtt is incremented according to a stateless round-robin scheme at ISs to select one among the (RTT+1) FIFOs. OSs keeps working as in the case of RTT=0.

The IMRR pseudo-code is reported in Alg. 1. Line 4 describes the cell selection process at ISs. The corresponding requests are sent to OSs. In the grant phase, OSs try to first grant the preferential input (lines 8-9). Otherwise, OSs grant the request corresponding to the smallest cell fanout (lines 11-19). The pointers to the preferential input and to FIFO queues are incremented in lines 22-23. Finally, cells are removed from the HoL, if completely served (lines 26-31).

When $k = 2\times$(RTT$+1$) queues are present at inputs to reduce the HoL blocking, in each time slot, 2 queues are available, depending on the current value of the

Algorithm 1. iMRR pseudo-code (short version)

Require: rtt=0

```
 1: repeat
 2:   /**** REQUEST PHASE ****/
 3:   for all inp=1 to Number_of_inputs do
 4:     SEND REQUEST to each output in input[inp].queue[rtt].cell.fanout_set
 5:   end for
 6:   /**** GRANT PHASE ****/
 7:   for all output=1 to Number_of_outputs do
 8:     if a REQUEST is received from the preferential_input then
 9:       SEND GRANT to the preferential_input
10:     else
11:       SET min_weight to 0
12:       SET input_to_grant to -1
13:       for all inp = 1 to Number_of_inputs do
14:         if (input_to_grant = -1) OR (input[inp].queue[rtt].cell.fanout <
            min_weight) then
15:           SET input_to_grant to input
16:           SET min_weight to input.cell.fanout
17:         end if
18:       end for
19:       SEND GRANT to input_to_grant
20:     end if
21:   end for
22:   /**** UPDATE POINTERS****/
23:   INCREMENT preferential_input MODULO Number_of_inputs
24:   INCREMENT rtt MODULO RTT
25:   /**** CLEAN UP FIFOs ERASING HoL CELLS ****/
26:   for all inp=1 to Number_of_inputs do
27:     DECREMENT cell fanout for all received grants
28:     if input[inp].queue[rtt].cell.fanout == 0 then
29:       EXTRACT cell from the HoL
30:     end if
31:   end for
32: until (end of simulation)
```

queue pointer rtt. ISs issue a request for the cell with the largest weight among the 2 available cells for the corresponding rtt value. The cell weight is set equal to the queue length plus the cell fanout.

Note that IMRR, differently from mRRM, is suited to run in a fully distribute scenario, since the pointer update rule is stateless. Furthermore, with respect to WBA, IMRR does not require the computation of the cell age, a relatively complex task at high speed.

4 Performance results

We show performance results based on simulation runs exploiting a proprietary simulation environment developed in C language. Statistical significance of the results is assessed by running experiments with an accuracy of 1% under a confidence interval of 95%. The simulator correctness was checked, for previously proposed schedulers, by comparing performance results for RTT=0 against results reported in the literature [5]. Furthermore, overloading the switch with uniform unicast traffic, permits to verify that IMRR correctly degenerates to a random scheduler, thus providing the same performance. Finally, all algorithms correctly provide 100% throughput under overloading broadcast traffic.

We compare IMRR with the previously presented extensions of mRRM and WBA, and with the multi-copy approach, where multicast cells are replicated at inputs and treated as unicast cells. In the multi-copy approach, the unicast SRR scheduler [2] is run, given its good performance and adaptability to the fully distributed multi-chip scenario.

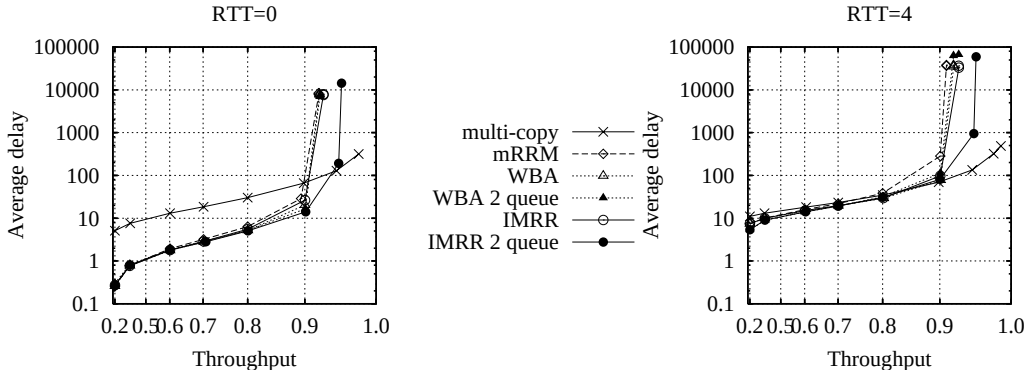

Fig. 5. Performance comparison under uniform Bernoulli cell arrivals.

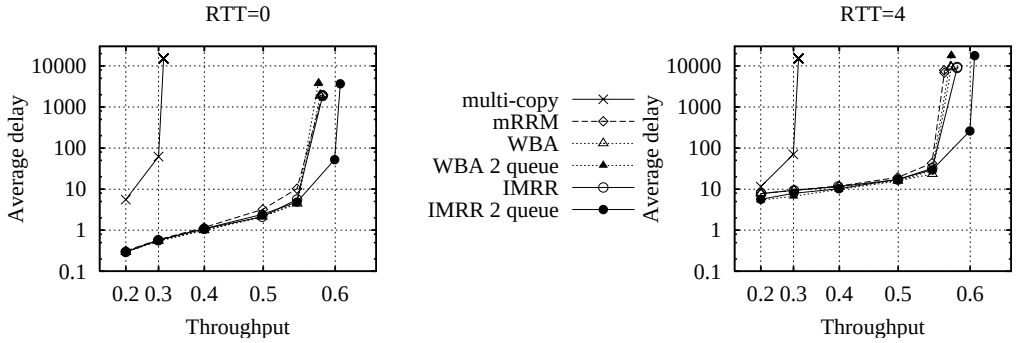

Fig. 6. Performance comparison under gathered Bernoulli cell arrivals.

The switch has $N = 16$ inputs and outputs running at the same speed; cells are generated according to an i.i.d. Bernoulli process, i.e., at each time slot, an input port receives a cell with probability ρ, $0 \leq \rho \leq 1$, equal to the input load. Later, we

also consider packet arrivals, i.e., trains of cells willing to reach the same output, the number of cells being drawn from a uniform distribution ranging from 1 to 16 cells. Performance indices are either average delays vs normalized switch throughput or maximum achievable normalized throughput in overload.

In terms of traffic distribution among input and output ports, we initially consider both i) a uniform scenario, in which all input (and output) ports are equally loaded and the fanout set of a new cell is generated randomly, according to a uniform distribution, ii) a gathered scenario, where the traffic is gathered over few ($M = 5$) active input ports and equally distributed over all $N = 16$ output port. In the gathered scenario, the fanout set is chosen according to a non-uniform binomial distribution, with mean fanout $h_m = 3.66$. More precisely, the probability P_f of choosing a fanout set of size f is $P_f = N/h_m \binom{N}{f} (h_m/N)^f (1 - h_m/N)^{N-f}$. This is a traffic pattern well known to be hard to schedule [7]. Indeed, when all inputs are equally loaded, the maximum sustainable traffic leads to a normalized input load which is at most $1/E[f]$, $E[f]$ being the average cell fanout size. If instead the traffic is gathered among few inputs, the normalized input load for sustainable traffic can approach 1, so that the efficiency in serving cells queued at the inputs becomes important on performance. Note that the considered gathered traffic scenario is far from being unrealistic. Multicast applications often generate sustained and long-lasting flows, that may only engage few inputs and several outputs at a given router or switch.

In both uniform and gathered traffic scenarios, unicast traffic is not given any special attention and is treated as a special case of multicast traffic with fanout set equal to one. To analyze switch behavior in a more realistic environment which includes a significant percentage of unicast traffic, we also examine two mixed traffic scenarios, the first one comprising a variable mix of unicast and broadcast traffic only, the second one with a variable percentage of unicast vs multicast traffic, highlighting delays for 50% of unicast and 50% of multicast traffic,

IMRR performance are reported as solid lines with circles, the modified version of WBA is plotted using dotted lines with triangles, whereas the modified version of mRRM is identified by dashed lines with diamonds; the multi-copy approach is plotted with a solid line with crosses. White symbols refer to the case of a single FIFO for RTT=0 and RTT$+1$ queues for RTT> 0, whereas black symbols are used for $k = 2$ FIFOs for RTT+0 and $k = 2 \times$ (RTT$+1$) FIFOs for RTT> 0.

In Fig.5 we report delays as a function of the switch throughput for variable RTTs under uniform multicast Bernoulli traffic with cell arrivals. Only the multi-copy approach permits to obtain 100% throughput; however, the price to be paid is a significant increase in the average delay at low-medium loads when RTT=0. When increasing RTT to 4 time slots, all algorithms show similar delay performance. No major differences are evident among multicast schedulers, apart from a slight throughput increased obtained by the proposed IMRR when using k FIFO queues

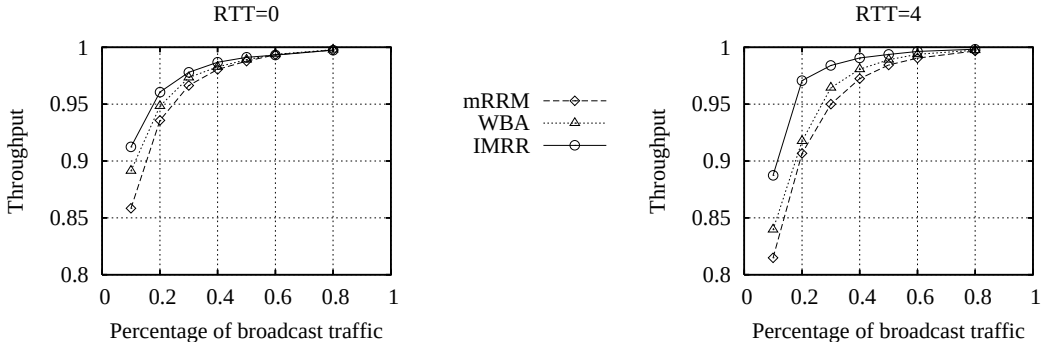

Fig. 7. Performance comparison for uniform Bernoulli cell arrivals with a variable percentage of unicast and broadcast traffic.

per RTT.

Despite its good performance in the uniform scenario, recall that the multi-copy approach is reported only as a reference case; indeed, it cannot be used in practice, since, for example, it is unable to sustain a broadcast flow overloading a single input. Indeed, when studying the performance of the switch under gathered traffic in Fig.6, the throughput limitation of the multi-copy approach becomes evident. In this scenario, the proposed IMRR approach provides improved throughput especially when using $k = 2$ FIFO queues per RTT. The WBA scheduler, despite its higher complexity, is unable to exploit $k = 2$ FIFO queues to obtain performance benefits.

IMRR shows good throughput performance also when considering the traffic scenario with unicast and broadcast traffic only, reported in Fig.7. When broadcast traffic becomes dominant, all algorithms show similar, good, performance. However, when unicast traffic becomes more significant, differences among algorithms become evident even when using a single FIFO queue per RTT, and IMRR outperforms the other algorithms.

Similar observations hold when looking at Fig.8, which reports the throughput as a function of the percentage of uniform traffic. All curves show a maximum throughput of 0.6 for unicast traffic only, due to the HoL blocking of the single FIFO queue. Throughput increase to about 0.95 for multicast traffic only. IMRR provides non marginal throughput benefits with respect to WBA and mRRM. Performance benefits are more evident for RTT> 0. When comparing the left-hand side with the right hand side plot, all algorithms show a slight throughput decrease due to the RTT among selectors; the penalty is minimal for IMRR. In Fig.9 we report delays as a function of the switch throughput when considering the uniform Bernoulli cell arrivals with 50% of multicast and 50% of unicast. Besides throughput benefits, IMRR provides slightly better delays with respect to WBA and mRRM both when RTT=0 and RTT > 0; performance benefits are larger for RTT> 0.

Finally, we study switch performance when considering packet arrivals at inputs instead of cell arrivals. This traffic is also named bursty traffic in the literature.

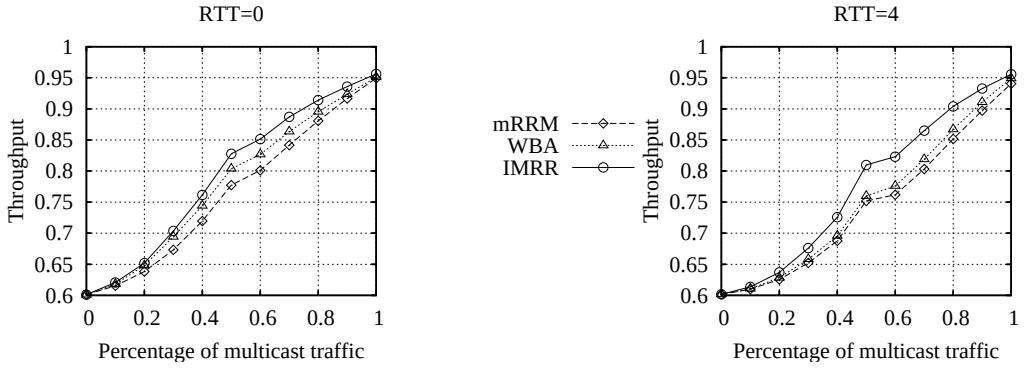

Fig. 8. Throughput comparison for Bernoulli cell arrivals for a variable percentage of unicast and multicast traffic.

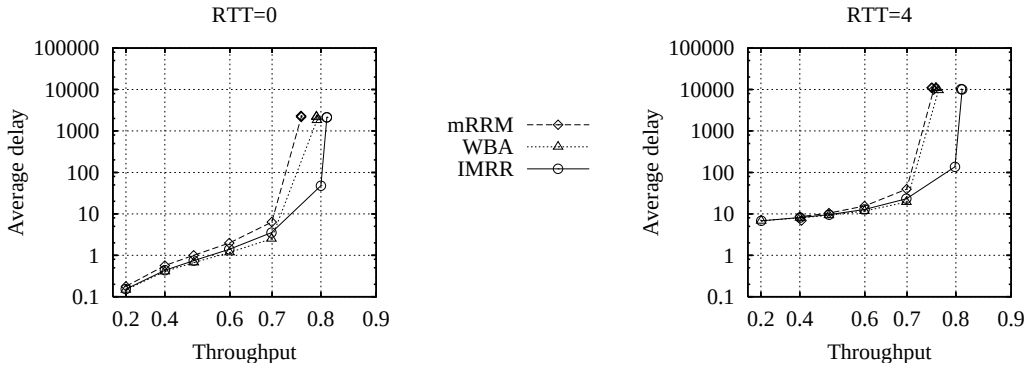

Fig. 9. Performance comparison for uniform Bernoulli cell arrivals: 50% unicast and 50% multicast traffic.

Packet size is uniformly chosen among 1 and 16 cells. When a packet is generated, it is segmented in cells and cells are sequentially stored in the proper queue for successive transmission. Cells access the switching fabric and are transmitted independently.

In Fig. 10 we report delays under uniform Bernoulli traffic. The surprising result is that WBA provides worse performance than IMRR both when RTT=0 and RTT=4. Recall that WBA requires a significant complexity increase with respect to the two other schedulers, due to the need of computing the cell age. This behavior is highlighted not only by the uniform packet size distribution, but also by other variable packet size distributions such as a trimodal distribution similar to the distribution of Internet packet size [15]. The reason for the throughput decrease can be explained by looking at the matching size PDF (Probability Density Function) reported in Fig. 11 for RTT=0 (similar results not shown hold for RTT=4) in overload. The matching size is computed as the number of edges that the scheduler selects in each time slot. The uniform packet size distribution makes it difficult for the multicast schedulers to select large size matchings (compare the left plot for fixed packet size with the right plot). This is due to the following phenomenon. When two badly conflicting packets reach the HoL of their respective queue, they start competing for

the same set of resources (output ports). Non conflicting packets in other queues can be transferred, but, since this conflicting situation lasts for several time slots due to the packet size, there is an increasing chance that other conflicting packets reach the HoL of other queues, creating a self-sustaining conflicting situation. This behavior is shared by all schedulers, and the maximum throughput is reduced from 0.9 to 0.8.

WBA further exacerbates this problem, since the weighted metric increases the probability of making the same matching choice at output ports in consecutive time slots. This does not hold for IMRR, thanks to the round-robin mechanism, since independent choices are made in each time slot. To confirm this intuitive observation, in Fig. 12 the matching persistency PDF is reported when RTT=0 (similar results, not shown, hold for RTT=4). The matching persistency is computed according to the following algorithm: Considering two consecutive time-slots, compute the persistency as the number of edges that are selected in both matchings. Clearly, the matching persistency ranges from 0 to 16. WBA shows an increase in matching persistency due to the memory effect created by the weighted metric when selecting the matching edges. When using a fixed packet size distribution, with a packet size equal to the cell size, this memory effect disappears. As such, weighted metrics, often proposed when running multicast scheduler, should be reconsidered, since they seem to offer good performance but only when considering cell-based arrivals, a scenario not really significant in today networks.

In summary, IMRR presents equivalent or better performance than other previously proposed multicast schedulers. IMRR does not require the computation of any delay based metrics, a rather complex task in today high-speed switches. Furthermore, it is suited to a fully distributed multi-chip implementation, the only scalable solution for large-size high-speed switches. Notwithstanding this additional constraints, IMRR shows good and improved performance both in the traditional case of monolithic implementation as well as when considering RTT induced by the multi-chip implementation of multicast schedulers.

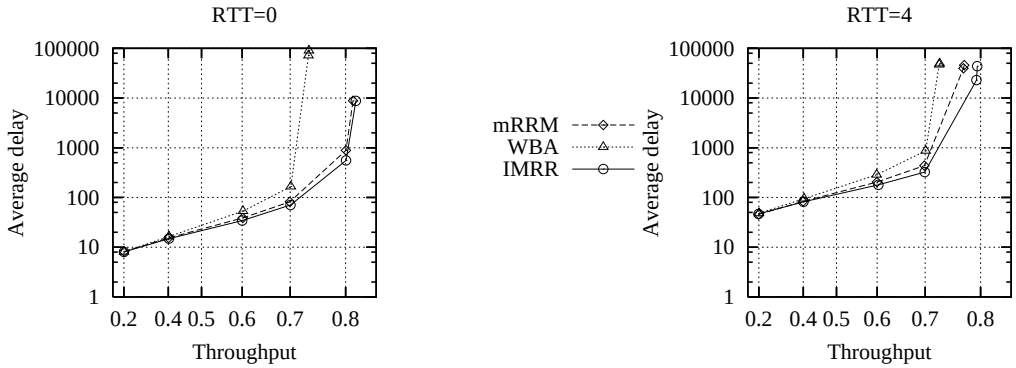

Fig. 10. Performance comparison under uniform Bernoulli packet arrivals.

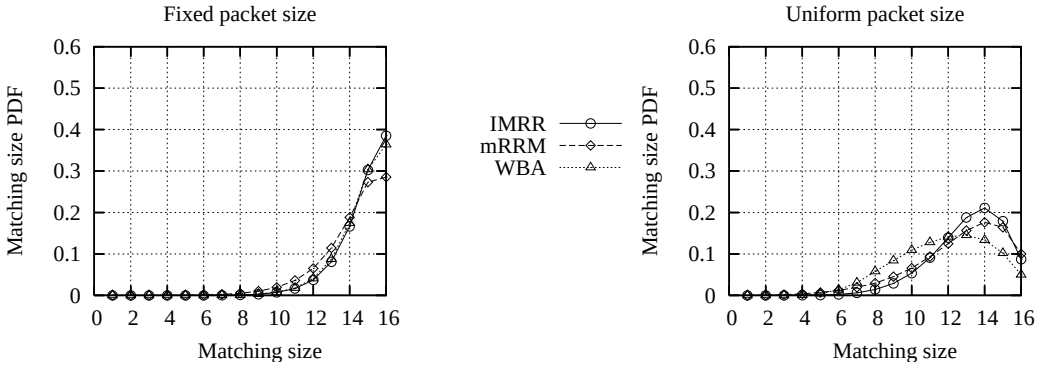

Fig. 11. Matching PDF for RTT=0 under uniform Bernoulli packet arrivals.

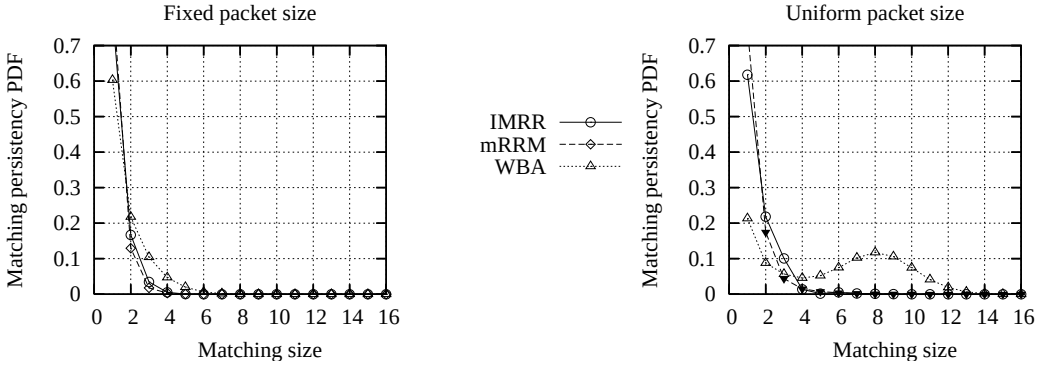

Fig. 12. Matching persistency for RTT=0 under uniform Bernoulli packet arrivals.

5 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 proposed modified multicast scheduler, named IMRR, is suited to a fully distributed multi-chip implementation, and shows performance improvements with respect to previously proposed multicast schedulers adapted to the multi-chip scenario. Unfortunately, all algorithms show increased average delays at low loads for increasing RTTs, a problem that would be nice to study and solve in the future.

References

[1] C.Minkenberg, F.Abel, E.Schiattarella, “Distributed crossbar schedulers”, HPSR’06, Poznan, Poland, June 2006

[2] A.Scicchitano, A.Bianco, P.Giaccone, E.Leonardi, E.Schiattarella, “Distributed scheduling in input queued switches”, IEEE ICC 2007, Glasgow, Scotland, June 2007

[3] M.Ajmone Marsan, A.Bianco, P.Giaccone, E.Leonardi, F.Neri, “Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput”, IEEE/ACM Transactions on Networking, Vol.3, No.11, pp.465-477, June 2003

[4] N.McKeown, B.Prabhakar, “Scheduling Multicast Cells in an Input-Queued Switch”, IEEE INFOCOM ’96, San Francisco, CA,USA, March 1996

[5] B.Prabhakar, N.McKeown, R.Ahuja “Multicast Scheduling for Input-Queued Switches”, IEEE JSAC (Journal on Selected Areas in Communications), Vol.15, No.5, pp.855-866, June 1997

[6] S.Gupta, A.Aziz, “Multicast scheduling for switches with multiple queues”, Hot Interconnects02, Stanford, CA, August 2002

[7] A.Bianco, P.Giaccone, E.Leonardi, F.Neri, C.Piglione “On the Number of Input Queues to Efficiently Support Multicast Traffic in Input Queued Switches”, HPSR’03, Torino, Italy, June 2003

[8] A.Bianco, P.Giaccone, C.Piglione, S.Sessa “Practical Algorithms for Multicast Support in Input Queued Switches”, HPSR’06, Poznan, Poland, June 2006

[9] 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

[10] 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

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

[12] 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

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

[14] A.Smiljanic, “Scheduling of multicast traffic in high-capacity packet switches”, IEEE Communications Magazine, Vol.40, No.11, pp.72-77, Nov 2002

[15] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, C. Diot, “Packet-Level Traffic Measurements from the Sprint IP Backbone”, IEEE Network, Vol.17, No.6, pp.6-16, November 2003

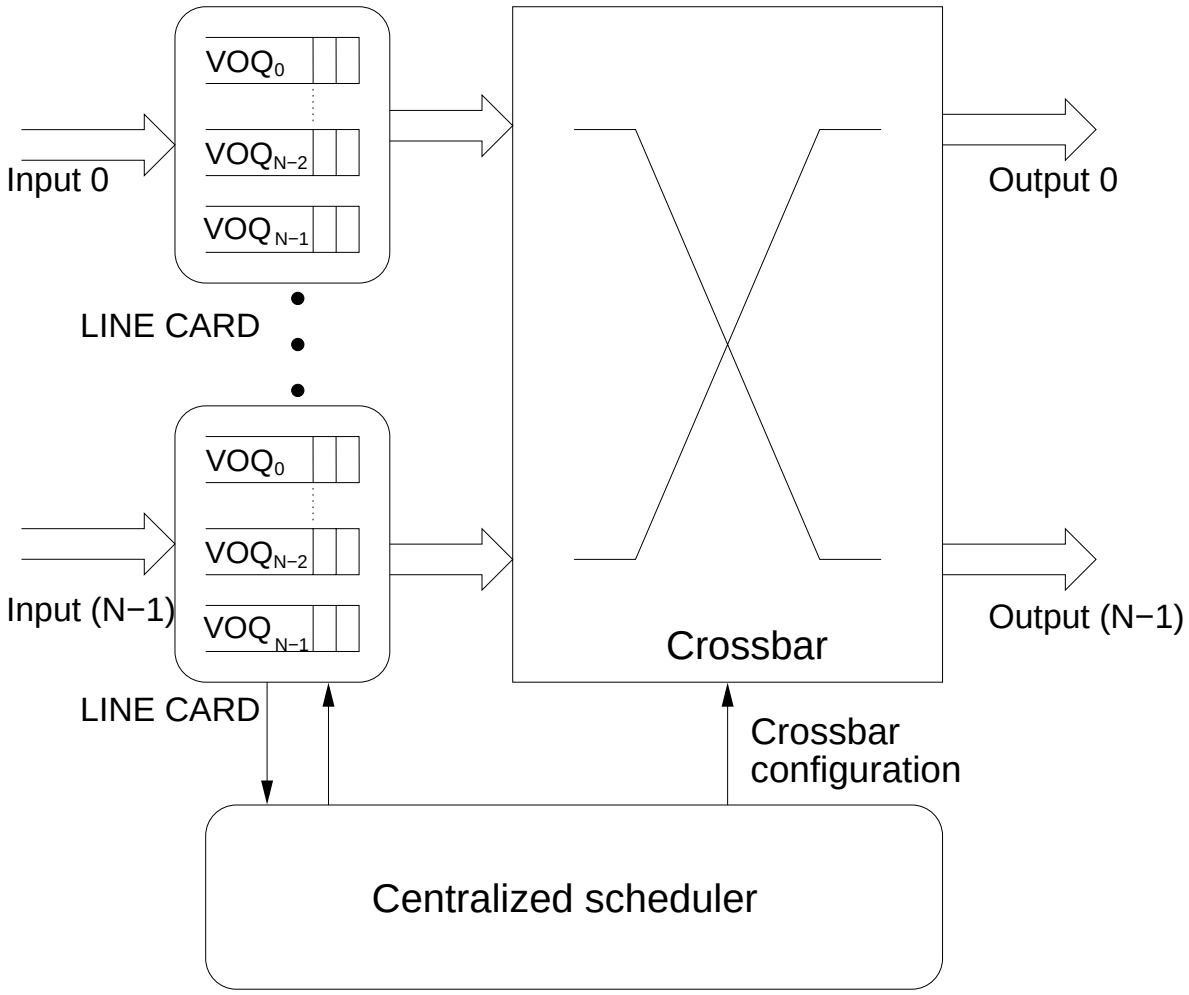

Fig. 1. IQ architecture with a centralized scheduler and VOQ architecture at inputs for unicast traffic.

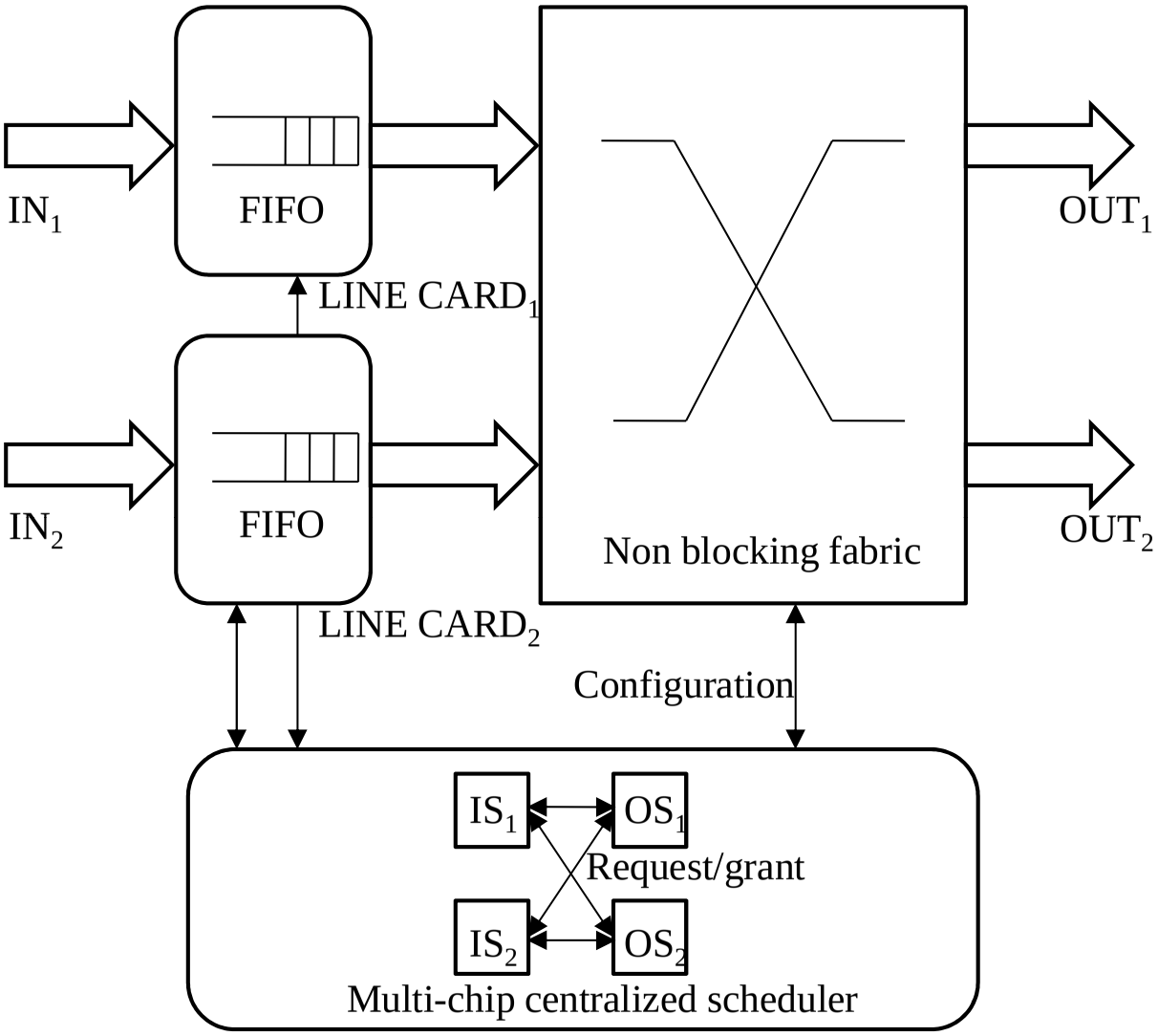

Fig. 2. 2 × 2 IQ architecture with a centralized multi-chip scheduler and FIFO queuing for multicast traffic.

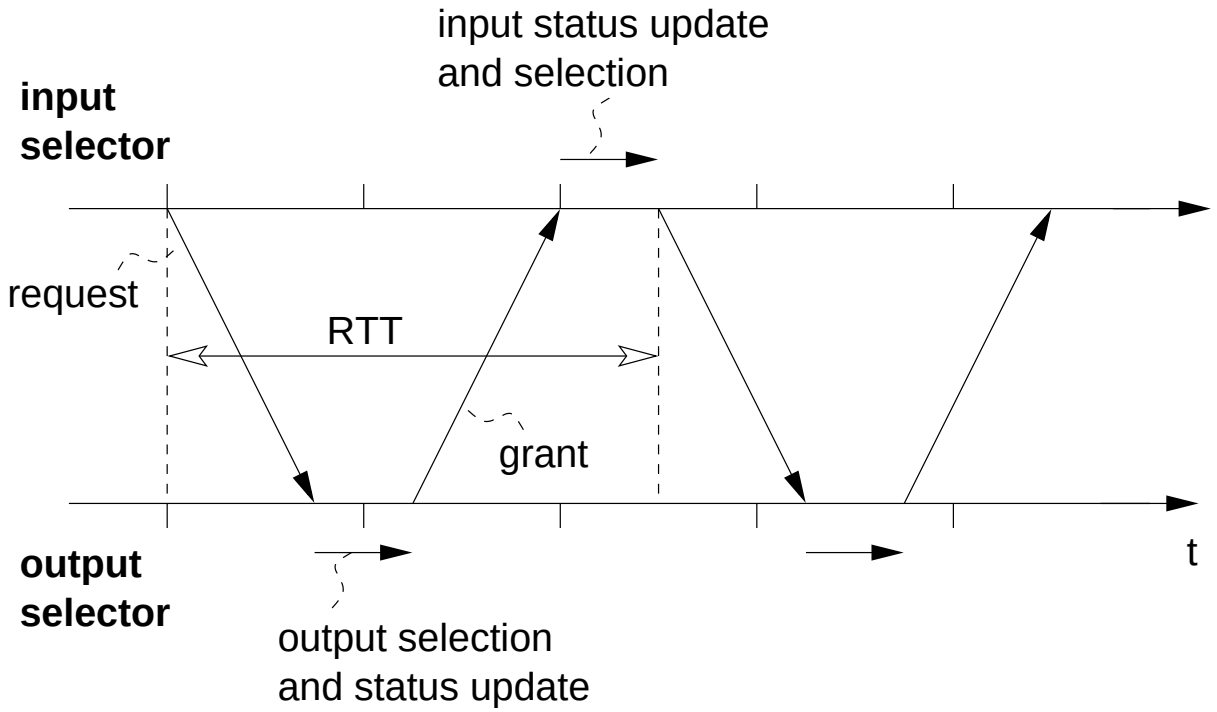

Fig. 3. Round trip time between ISs and OSs.

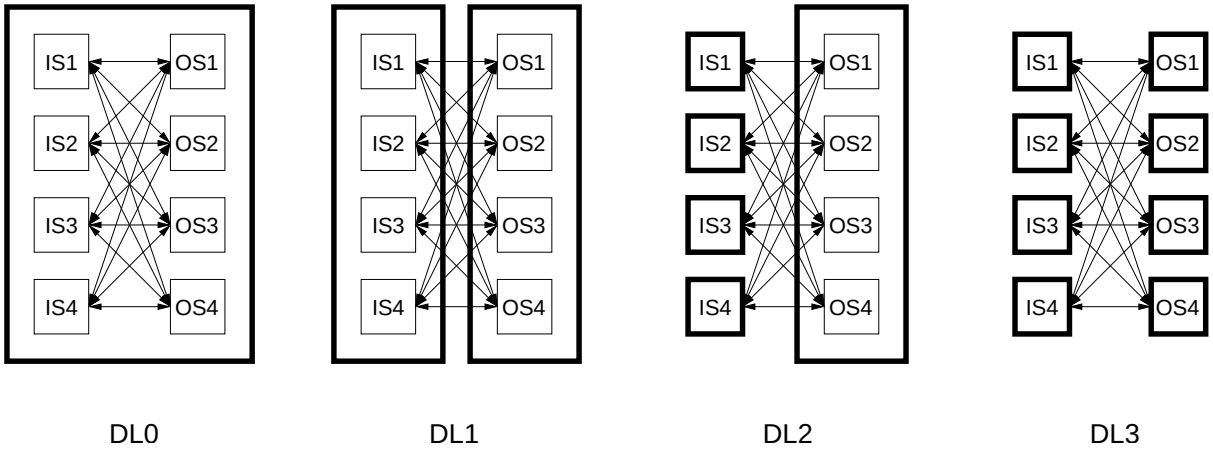

Fig. 4. Distribution levels of a centralized multi-chip scheduler in a 4×4 switch: DL0 is a single-chip implementation, DL3 is a fully distributed, multi-chip, implementation.

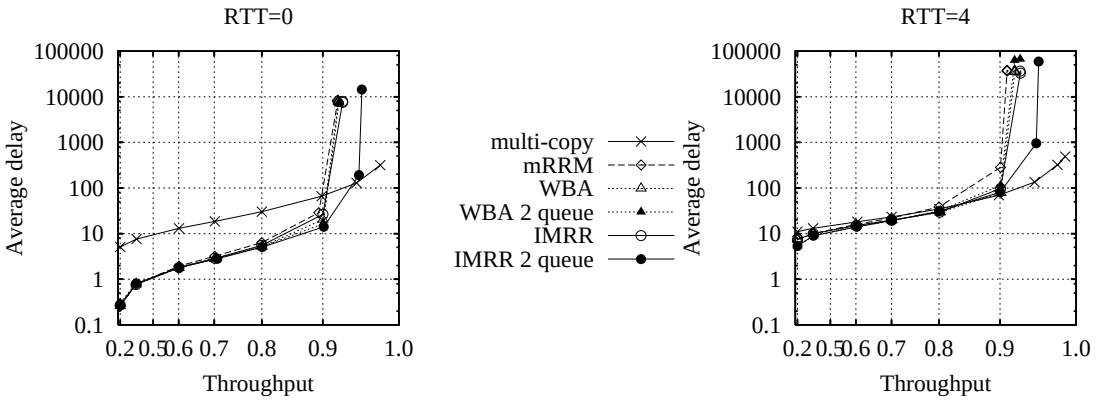

Fig. 5. Performance comparison under uniform Bernoulli cell arrivals.

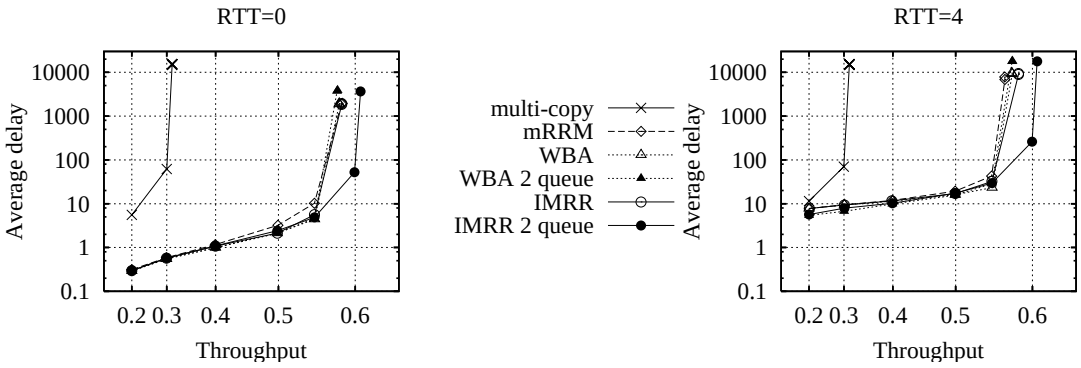

Fig. 6. Performance comparison under gathered Bernoulli cell arrivals.

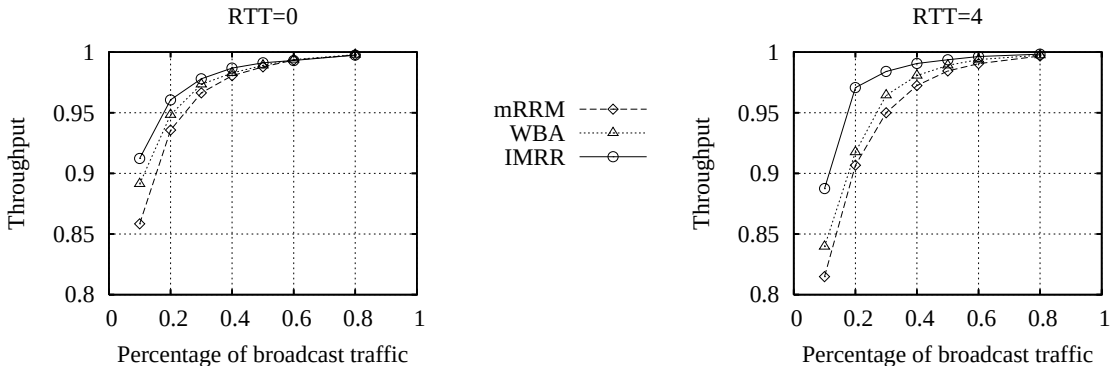

Fig. 7. Performance comparison for uniform Bernoulli cell arrivals with a variable percentage of unicast and broadcast traffic.

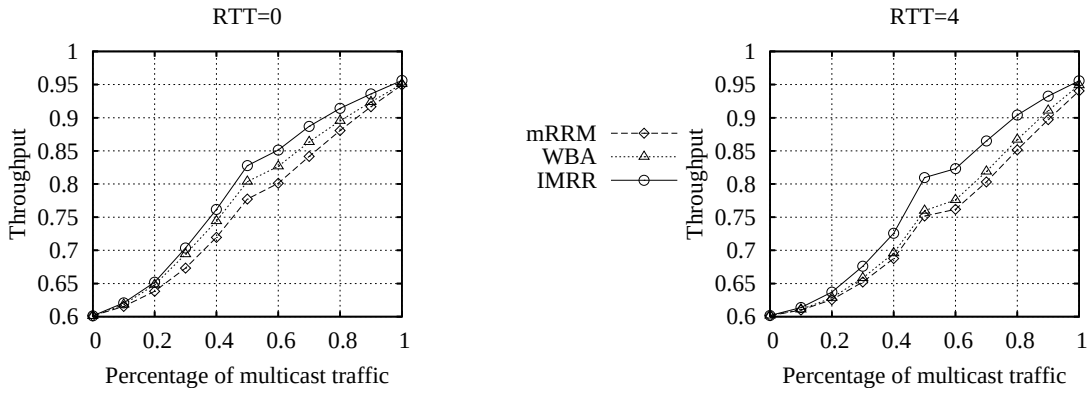

Fig. 8. Throughput comparison for Bernoulli cell arrivals for a variable percentage of unicast and multicast traffic.

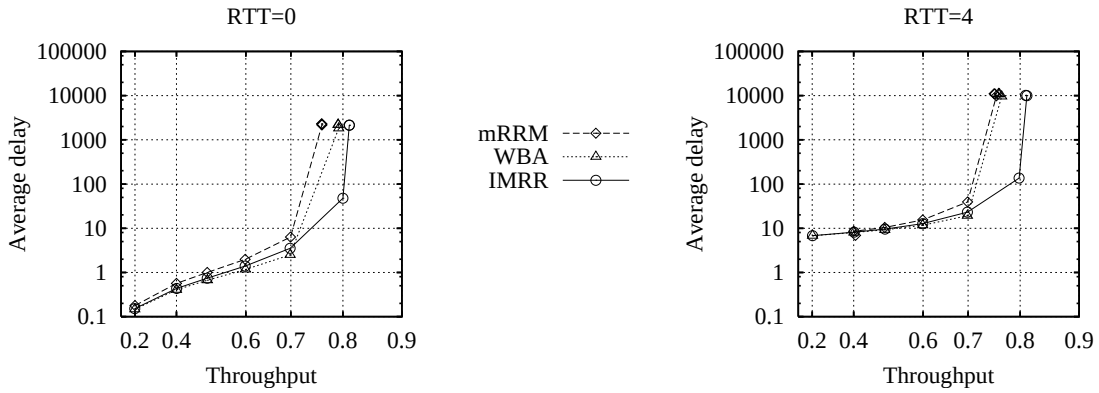

Fig. 9. Performance comparison for uniform Bernoulli cell arrivals: 50% unicast and 50% multicast traffic.

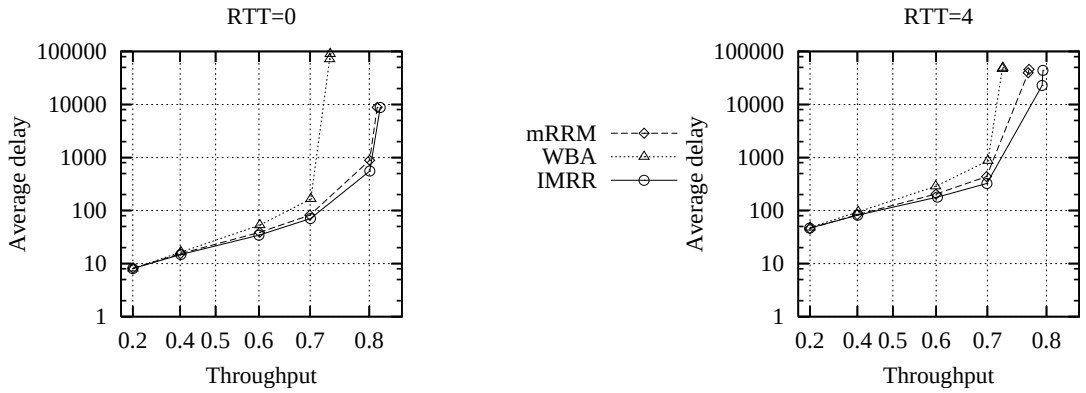

Fig. 10. Performance comparison under uniform Bernoulli packet arrivals.

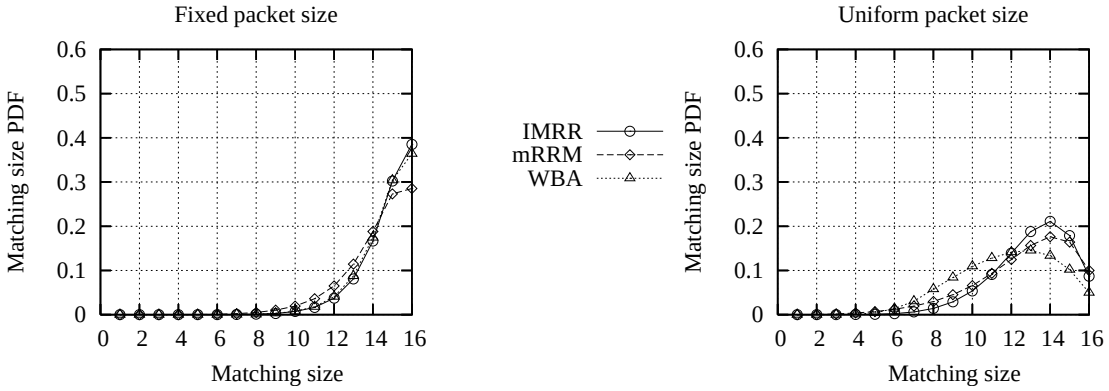

Fig. 11. Matching PDF for RTT=0 under uniform Bernoulli packet arrivals.

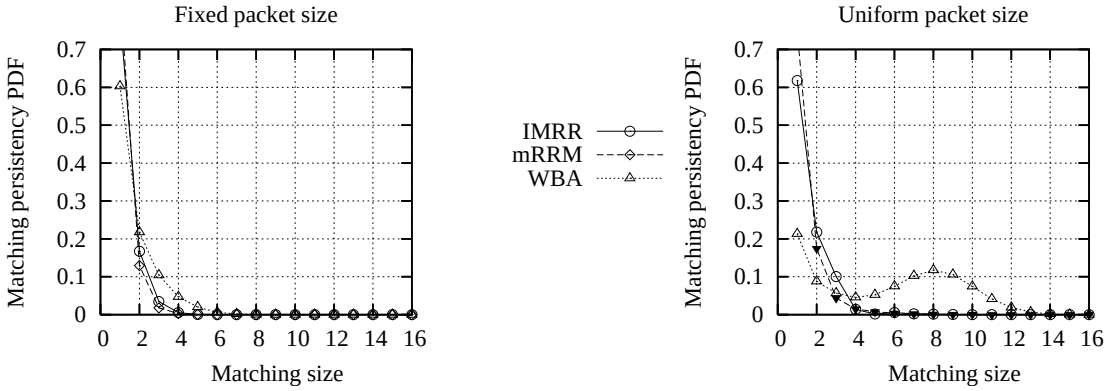

Fig. 12. Matching persistency for RTT=0 under uniform Bernoulli packet arrivals.