Matching From the First Iteration: An Iterative Switching Algorithm for an Input Queued Switch

Saad Mneimneh

***Abstract*—An iterative switching algorithm for an input queued switch consists of a number of iterations in every time step, where each iteration computes a disjoint matching. If input i is matched to output j in a given iteration, a packet (if any) is forwarded from i to j in the corresponding time step. Most of the iterative switching algorithms use a *Request Grant Accept* (RGA) arbitration type (e.g. *iSLIP*). Unfortunately, due to this particular type of arbitration, the matching computed in one iteration is not necessarily maximal (more input and output ports can still be matched). This is exactly why multiple iterations are needed. However, multiple iterations make the time step larger and reduce the speed of the switch.**

We present a new iterative switching algorithm (based on the RGA arbitration) called π-RGA with the underlying assumption that the number of iterations is possibly limited to one, hence reducing the time step and allowing the switch to run at a higher speed. We prove that π-RGA achieves throughput and delay guarantees with a speedup of 2 and one iteration under a constant burst traffic model, which makes π-RGA as good as any maximal matching algorithm in the theoretical sense. We also show by simulation that π-RGA achieves relatively high throughput in practice under uniform and non-uniform traffic patterns with one iteration and no speedup.

***Index Terms*—Input queued switch, iterative switching algorithms, matching algorithms, number of iterations, speedup.**

I. Introduction

A SWITCHING algorithm for an input queued switch computes a matching in every time step. If input i is matched to output j, a packet (if any) is forwarded from i to j. In particular, an iterative switching algorithm computes its matching iteratively over a number of iterations within the time step, where each iteration computes a disjoint partial matching. Therefore, if input i is matched to output j in a given iteration, a packet (if any) is forwarded from i to j in the corresponding time step. The reason for such an iterative approach is to simplify the computation of the matching. In deed most iterative switching algorithms are distributed and use a *Request Grant Accept* (RGA) arbitration in every iteration. Therefore, we mainly focus on this family of iterative switching algorithms. In fact Cisco's routers use such algorithms [9]. Examples of these algorithms include *PIM*[1] [1], *iSLIP* [15], *iLQF* and *iOCF*[14], *DRR*[13], *pDRR* [7], and *LQD* [12]. We provide a brief description of these algorithms and a comparison between them in Section VI. The reader may consult the following references: [2], [7], [13]–[15].

With the RGA arbitration, each iteration is composed of three stages: *Request*, *Grant*, and *Accept*. In the *Request* stage, inputs send matching requests to the outputs. In the *Grant* stage, every output grants at most one request. Finally, in the *Accept* stage, every input accepts at most one granted request.[2] If input i accepts a grant from output j, i is matched to j. Unfortunately, since different inputs might request the same output, and similarly, different outputs might grant the same input, the resulting matching is not necessarily maximal, i.e. more input and output ports can still be matched. This is exactly why multiple iterations are needed. Furthermore, this situation cannot be avoided in general because there is no direct communication among the input ports themselves or among the output ports themselves, as this would lead to a more complicated hardware.

Nevertheless, with additional iterations in which previously matched inputs and outputs do not participate in the RGA arbitration, more inputs and outputs could be matched, thus leading to a larger size matching. A larger size matching will generally imply higher throughput of the switch, because more packets will be transmitted in every time step. However, from the theoretical point of view, the required additional iterations make the time step larger and reduce the speed of the switch. We make the following two observations:

- Many of the iterative algorithms practically achieve acceptable throughput with multiple iterations and no speedup.[3] On average, $O(\log N)$ iterations seem to be enough, see [1] for such an analysis.
- Some of the iterative algorithms, like *iSLIP* and *DRR*, can be proved to theoretically achieve 100% throughput with one iteration but only when the traffic is uniform, i.e. the rate of packets from an input to an output is the same all over the switch.

Therefore, we would like to possibly limit the number of iterations to only one iteration and still provide high throughput for an arbitrary traffic pattern (not necessarily uniform). We describe a new iterative switching algorithms (based on the RGA arbitration) called π-RGA with the underlying assumption that the number of iterations is possibly limited to one, hence re-

Manuscript received May 3, 2004; revised September 30, 2005, and January 18, 2006; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor G. Pacifici.

The author is with the Department of Computer Science, Hunter College of CUNY and the Graduate Center of CUNY, New York, NY 10021 USA (e-mail: saad@hunter.cuny.edu).

Digital Object Identifier 10.1109/TNET.2007.900365

[1] *PIM* uses randomness and reaches a maximal matching with $O(\log N)$ iterations on average. A variation on *PIM*, also presented in [1] and called statistical matching, achieves theoretically 72% throughput with 2 iterations.

[2] The RGA arbitration can also start from the output side like in DRR [13] and *LQD* [12].

[3] Theoretically however, a lower bound on the speedup required to achieve throughput can be proved for a number of iterative algorithms. For instance, it can be shown that *iSLIP*, *iLQF*, *iOCF*, and *DRR* require a speedup of $S \geq 1.5$ to achieve throughput. Therefore, these algorithms (including ours here) cannot achieve more than 66.67% throughput with no speedup [17].

ducing the time step and allowing the switch to run at a higher speed.

In fact, reducing the time step of the switch is a very important problem in today's routers. A number of attempts have been made in that regard. For instance, [9] describes a way to pipeline the *Request* stage and the *Grant Accept* stages, reducing the number of iterations by half. The *WFA* algorithm (and its variant *WWFA*) [18], which is an iterative algorithm not based on RGA, may be also pipelined to reduce the time step to one iteration. We will discuss *WFA* in Section VII and provide a conceptual connection to our algorithm π-RGA. *LQD* [12] is a one iteration RGA based randomized algorithm that achieves 100% throughput with a speedup of 2. However, it has poor performance in practice (as shown in Section VI) because its matching may be far from maximal. Therefore, our approach is to simply allow one iteration to π-RGA with the remaining challenge of computing a good size matching (preferably maximal) using that one and only iteration.

With the limit to one iteration, the π-RGA switching algorithm attempts to maintain parts of the previously computed matching to grow the size of the matching with successive time steps. Therefore, instead of restarting the computation of a matching from scratch in every time step (and hence not attaining a good size matching with the one iteration limit), π-RGA uses information about the previous matching.

The concept of using the previous matching is not entirely new. For instance, it was explored in [19] and [8] (and our previous work in [16] where the matching is held constant for a number of time steps). Both [19] and [8] use weighted matchings where the weight contribution of matching input i to output j represents the number of packets waiting at input i and destined to output j. The work in [19] describes a randomized algorithm by which the previous matching is maintained only if the new randomly computed matching has less weight. The work explores the established fact that computing a maximum weighted matching in every time step achieves throughput [5]. The work in [8] relies on the same concept and describes variants of the above algorithm in addition to some new deterministic algorithms. In this work we do not consider weighted matchings. In fact, π-RGA does not maintain information about the quality of the matching it computes (this would require a centralized scheme that is aware of the whole matching), but simply stabilizes a maximal matching with successive time steps, regardless whether that matching is "good" or "bad". We also do not exploit randomness; however, a simple priority scheme will trigger a change in the matching; but this will not happen frequently, thus creating a balance between two desired extremes: on one hand computing a maximal matching and on the other hand avoiding starvation (possibly resulting from keeping the same maximal matching). Moreover, our work is different from that of [19] and [8] in that it is in the context of iterative switching algorithms that use the RGA arbitration and thus computes a matching in a distributed way at each port.

We describe π-RGA in detail in Section II. Sections III, IV, and V provide the necessary framework to prove that π-RGA achieves throughput and delay guarantees with a speedup of 2 and one iteration under a constant burst traffic model. To say the least, this makes π-RGA as good as any maximal matching algorithm in the theoretical sense. We also show by simulation in Section VI that π-RGA achieves relatively high throughput in practice (better than the rest of the iterative switching algorithms) under uniform and non-uniform traffic patterns with one iteration and no speedup.

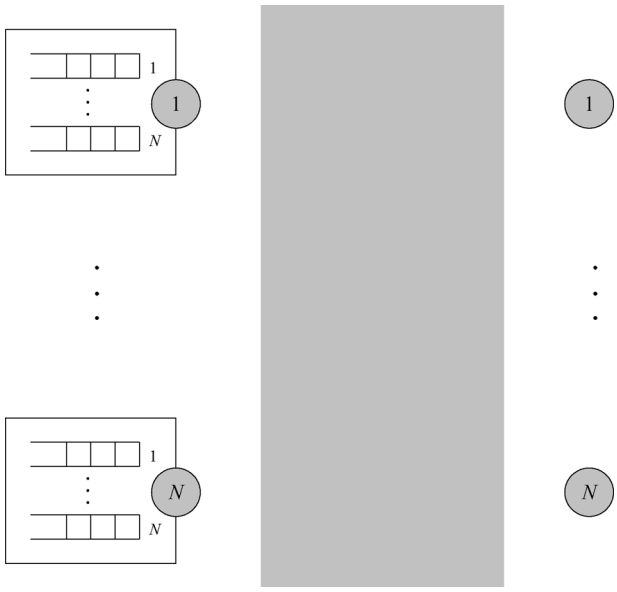

Fig. 1. An input queued switch.

Our presentation of π-RGA is for the standard input queued switch model described in the literature. We briefly describe the model here. Fig. 1 illustrates an input queued switch with N input ports and N output ports and virtual output queues (VOQs). We denote by VOQ_{ij} the j^{th} virtual output queue at input i, which is the queue holding packets originating at input i and destined to output j. We will assume that an input queued switch with a speedup S (S is not necessarily an integer) operates in matching phases of $1/S$ time steps each, with $S = 1$ for no speedup. Therefore, to generalize the operation of an iterative switching algorithm in the presence of speedup, multiple iterations are performed per matching phase (as opposed to per time step). Starting from matching phase 0, each matching phase is composed of an iterative RGA arbitration which results in a matching, and if input i is matched to output j by the iterative RGA arbitration, VOQ_{ij} will be served in that matching phase (a packet will be forwarded from VOQ_{ij} to output j). Therefore, we can refer to matching phases instead of time steps. When the switch has no speedup ($S = 1$), matching phases and time steps are the same.

II. The π-RGA Switching Algorithm

The π-RGA switching algorithm is an iterative RGA arbitration type algorithm. Therefore, for every matching phase (see the description of the input queued switch model above), π-RGA performs an RGA arbitration in a number of iterations (possibly one) within the matching phase. As described earlier, the RGA arbitration has three stages: *Request*, *Grant*, and *Accept*. The π-RGA algorithm differentiates between two kinds of requests: *Strong* and *Weak* requests. Requests that were granted and accepted become *Strong* requests in the following

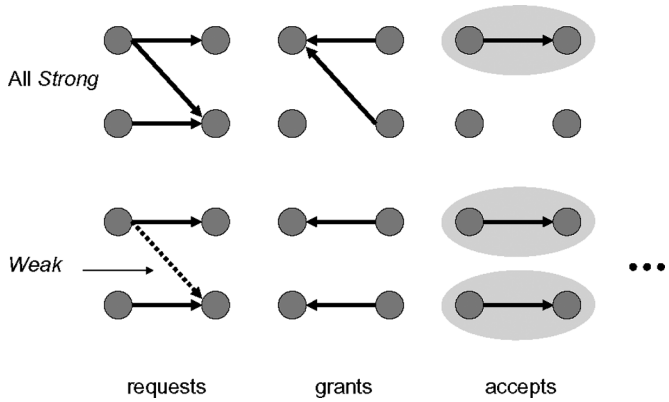

Fig. 2. Example of *Strong* and *Weak* requests.

Algorithm π-RGA *

start with an empty matching $M = \emptyset$
repeat for a number of iterations I (possibly $I = 1$)

R (at unmatched input i): if there is an active VOQ_{ij} that was served in matching phase $m-1$, issue *Strong* requests for all active VOQ_{ik} such that $VOQ_{ij} \not\prec_{\pi_m} VOQ_{ik}$, and *Weak* requests for all other active VOQs; otherwise, issue *Strong* requests for all active VOQs.

G (at unmatched output): with R being the set of requests received, if there are *Strong* requests in R, grant the π-highest *Strong* request in R; otherwise, grant the π-highest *Weak* request in R if any.

A (at unmatched input): with G being the set of grants received, accept the π-highest granted request in G if any.

if input i accepts a grant from output j
$M = M \cup (i, j)$

* All *Weak* requests can be suppressed without affecting any of the theoretical results presented in this paper. However, keeping the *Weak* requests creates more opportunities for matches in practice. Alternatively, a *Weak* request can be considered to be a request with the lowest priority, by overriding its priority as given by π. In both cases, there will be no need for the *Strong* and *Weak* modifiers.

Fig. 3. The π-RGA switching algorithm (matching phase m).

matching phase. Precedence is given to the *Strong* requests, and hence the matching will tend to stabilize with successive matching phases towards a matching that grants the *Strong* requests. By not competing with requests at other inputs, *Weak* requests will help the stabilization process to grow the size of the matching with successive matching phases. To guarantee progress, an unmatched input sends only *Strong* requests. Fig. 2 shows an example where the matching converges after two matching phases (one iteration per matching phase) and stabilizes thereafter.

A priority scheme π will be used in conjunction with the *Strong* and *Weak* modifiers in order to ensure that the stabilization process favors connections with high priority, and hence eliminates any possible starvation due to always granting the same set of *Strong* requests. The priority scheme π, therefore, serves as a parameter to the algorithm as suggested in the name π-RGA. The properties of the priority scheme π will be discussed later in the paper. We will show that with an appropriate choice of π, the π-RGA switching algorithm achieves throughput and delay guarantees with a speedup of $S \geq 2$, under certain traffic models.

In order to fully describe the π-RGA algorithm, we introduce the following definitions of an active VOQ and a VOQ transition.

Definition 1: A VOQ is active in matching phase m iff it is non-empty (has at least one packet) at the beginning of matching phase m.[4]

Definition 2: A VOQ transition in matching phase $m > 0$ is a transition of the VOQ from being inactive (active) in matching phase $m-1$ to active (inactive) in matching phase m.

We also define a priority scheme as follows:

Definition 3: A priority scheme π defines for every matching phase m a strict (non-reflexive) partial order relation π_m on the VOQs.

We will use the notation $VOQ_{ij} \prec VOQ_{kl}$ to denote that VOQ_{ij} has higher priority than VOQ_{kl} during matching phase m (VOQ_{ij} is ordered before VOQ_{kl} by π_m). We will also use the notation $VOQ_{ij} \not\prec \pi_m VOQ_{kl}$ to denote that VOQ_{ij} does not have higher priority than VOQ_{kl} during matching phase m (VOQ_{ij} is not ordered before VOQ_{kl} by π_m).

We will assume that for every i, j, k, and for every matching phase m, VOQ_{ij} and VOQ_{ik} are ordered by π_m, and VOQ_{ij} and VOQ_{kj} are ordered by π_m. Therefore, during matching phase m, VOQs that share either an input or an output must be ordered by π_m. This assumption will be crucial for the process of stabilizing the matching.

Definition 4: In a matching phase m, a VOQ_{ij} is π-highest for a set of VOQs Q iff every *active* $VOQ_{kj} \in Q$ satisfies $VOQ_{kj} \not\prec_{\pi_m} VOQ_{ij}$; furthermore, if $VOQ_{ij} \in Q$, we say that VOQ_{ij} is π-highest in Q.

Note that with the RGA arbitration, a request from input i to output j would eventually serve, if any, VOQ_{ij}. Therefore, if such a request exists, we say that input i issues a request for VOQ_{ij}. Similarly, we use a π-highest request to denote a request for a π-highest VOQ.

Fig. 3 shows the π-RGA switching algorithm for matching phase m. In Fig. 3, active implicitly means active in matching phase m.

The sequence input-output-input of Fig. 3 where the three stages of an iteration are performed can be alternatively changed to output-input-output. But since information about VOQs is more naturally obtained at the input side, we adopt the sequence shown in the figure.

The most crucial aspect of the π-RGA algorithm is the way requests are prepared. Every input issues *Strong* requests for active VOQs that have high priority, where the threshold of high priority is the priority of the previously served VOQ. Therefore, an input attempts to request better service based on the priority scheme π. Moreover, since *Weak* requests, regardless of their

[4]We sometimes omit to mention the matching phase m when it is clear from the context or when it is not important, for instance, when a VOQ is always active. Note that this definition of an active VOQ does not capture the real-time state of the VOQ. A simple example would be the following ($S = 1$ and, therefore, one matching phase is the same as one time step): A packet arrives at the beginning of matching phase m, that packet is transmitted by the end of matching phase m, and a packet arrives at the beginning of matching phase $m+1$. In terms of whole packets, the VOQ is non-empty at the beginning of both matching phases m and $m+1$, yet there is a time in between where the VOQ is empty. We can work with a real-time definition of an active VOQ that would capture the above phenomenon, but we choose the current one for simplicity of illustration.

priority, are always considered next, an input which has already accepted a high priority granted request will not prohibit high priority requests at other inputs from being granted (by making its low priority requests *Weak*). Variations on using *Weak* requests are possible (see the bottom note of Fig. 3).

Therefore, to summarize what has been described so far, the approach behind π-RGA can be conceptually visualized as having three different components:

- The *Strong* requests help stabilize the matching by creating requests that will always tend to be granted. Therefore, in the absence of multiple iterations, the matching needs not be computed from scratch.
- The priority scheme helps guide the stabilization process of the matching by determining which requests can become *Strong* (to avoid possible starvation).
- The *Weak* requests help grow the size of the matching with successive matching phases by disallowing these requests, which are unlikely to be accepted anyway, from competing with other requests (also holds when *Weak* requests are suppressed or assigned the lowest priority).

The use of *Weak* requests possesses a corrective mechanism over successive matching phases: an input can be matched to an output (and possibly unmatched from its previously matched output) when the requests prohibiting it from matching to that particular output become *Weak*. Therefore, in the absence of multiple iterations, the use of *Weak* requests emulates the mechanism by which a matched input stops sending requests in future iterations, thus virtually creating multiple iterations from successive matching phases.

When the number of iterations is fairly large (and, therefore, the emulation described above is not needed) π-RGA is not the best algorithm to use. Moreover, one expects π-RGA to produce a relatively high jitter because of the process of stabilizing the matching. Hence, in the presence of jitter sensitive traffic, π-RGA may not be the ideal choice. However, jitter sensitive traffic usually occupies a small fraction of the whole traffic and can be handled separately (see [10] for instance).

We have not yet specified what priority scheme π may be used. For instance, if π changes frequently from one matching phase to another (like round robin for instance), so will the *Strong* requests (and hence the grants), preventing the stabilization of the matching. This will make it difficult to realize the main goal of the algorithm. In the following sections, we discuss formally some of the properties that π may have and their implications on the performance of the π-RGA switching algorithm.

III. Stable Priority Scheme π

In this section we define a stable priority scheme:

Definition 5—Stable π: A priority scheme π is a stable priority scheme iff it satisfies the following: if VOQ_{ij} and VOQ_{kl} remain active during $[m_1, m_2]$, then $VOQ_{ij} \prec_{\pi_{m_1}} VOQ_{kj} \Leftrightarrow VOQ_{ij} \prec_{\pi_m} VOQ_{kl}$ for any matching phase $m \in [m_1, m_2]$.

Therefore, once two VOQs are active in the same matching phase, they retain their relative priorities until one of them transitions to inactive. When π is stable, we claim that the π-RGA algorithm will *attempt* to stabilize a maximal matching that favors higher priority VOQs. More precisely, it will attempt to reach a π-*stable* matching as defined below (inspired by the stable marriage matching [6]):

Definition 6: For a given priority scheme π, a matching computed in matching phase m is π-*stable* iff it satisfies the following condition: for every i, j, if VOQ_{ij} is active, then an active $VOQ_{kj} \not\succ_{\pi_m} VOQ_{ij}$ is served, with $k = i$ or $l = j$.

Therefore, a π-*stable* matching implies that if an active VOQ_{ij} is not served, then this is only because it is blocked by another served active VOQ with no less priority.[5] Note that a π-*stable* matching is a maximal matching. In fact, Definition 6 is a special case of the stable marriage matching. Our use of the term π-*stable* is, therefore, in reference to the stable marriage matching and has nothing to do with the stability of the switch itself.

To understand the stabilization claimed above when π is stable, consider first the case where no VOQ transitions occur.

If no VOQ transitions occur, a VOQ is either active or inactive for all matching phases. Moreover, since the priority scheme π is stable, it defines the same order relation on all active VOQs for all matching phases. Recall that, by assumption, in a matching phase m, VOQs sharing an input or an output are ordered by π_m, and hence a π-highest VOQ in matching phase m is uniquely determined at each port. Let Q be the set of all active VOQs. A π-highest VOQ_{ij} in Q will be issued a *Strong* request which will be granted and accepted, say in matching phase m. Therefore, VOQ_{ij} will be served in matching phase m and will continue to be served forever. Let Q' be the subset of all VOQs in Q that can still be served while serving VOQ_{ij}. All VOQs in $Q - Q'$ are therefore blocked by a higher priority VOQ, namely VOQ_{ij}. A π-highest VOQ_{kl} in Q' will be issued a *Strong* request in matching phase $m + 1$. The reason for this is the following: If no VOQ at input k was served in matching phase m, then input k will issue *Strong* requests for all its active VOQs in matching phase $m + 1$. If, on the other hand, a VOQ at input k was served in matching phase m, then it was a VOQ with no more priority than VOQ_{kl} (possibly VOQ_{kl} itself), since all VOQs in Q with higher priority than VOQ_{kl} are in $Q - Q'$. As a result, input k will issue a *Strong* request for VOQ_{kl} in matching phase $m + 1$, which will be granted and accepted; this is guaranteed by the *Request* stage which will issue *Weak* requests for all active VOQs of output l that are in $Q - Q'$. Therefore, VOQ_{kl} will be served in matching phase $m + 1$ and will continue to be served forever. Let Q'' be the subset of VOQs in Q' that can still be served while serving VOQ_{kl}. Again, all VOQs in $Q' - Q''$ are now blocked by a higher priority VOQ, namely, VOQ_{kl}.

Since the size of a matching is at most N, this continues until the matching is stabilized after at most N matching phases. In this resulting matching (obtained in at most N matching phases), an active VOQ that is not served is blocked by a higher priority active VOQ that is served, and hence the matching is π-*stable*.

The above argument assumed that no VOQ transitions occur; however, if VOQ transitions do occur, the matching might be perturbed every time there is such transition. This is why we

[5]In our case, it will be of higher priority because we assume that, for every matching phase m, VOQs that share an input or an output must be ordered by π_m.

claimed that π-RGA *attempts* to stabilize a maximal matching, which *might* not happen. Nevertheless, we can still have a notion of stability for a particular VOQ even in the presence of VOQ transitions. This notion is captured in the following definition.

Definition 7: For a priority scheme π, a matching computed in matching phase m is π-*stable with respect to* VOQ_{ij} iff it satisfies the following condition: If VOQ_{ij} is active, then an active $VOQ_{kj} \not\prec_{\pi_m} VOQ_{ij}$ is served, with $k = i$ or $l = j$.

Note that the above definition is a relaxation of Definition 6 in the sense that the matching satisfies the property with respect to VOQ_{ij} only instead of all VOQs. Note also that if no VOQ transitions occur, as argued above, π-RGA will reach a π-*stable* matching with respect to all VOQs in at most N matching phases. The interesting observation is that a transition for VOQ_{kl} will not affect the notion of stability in Definition 7 for VOQ_{ij} if VOQ_{kl} does not have higher priority than VOQ_{ij}. More precisely, we state Lemma 1 below.

Lemma 1: Given a stable priority scheme π and a VOQ_{ij} that remains active during $[m_1, m_2]$ ($m_2 \geq m_1 + N - 1$), if all VOQ transitions during $[m_1 + 1, m_2]$ occur only in a set of VOQs for which VOQ_{ij} is π-highest during $[m_1, m_2]$, then the matching computed by the π-RGA switching algorithm is π-*stable* with respect to VOQ_{ij} for every matching phases $m \in [m_1 + N - 1, m_2]$.

Proof: Let Q be the set containing VOQ_{ij} and all VOQ_{kl} active in matching phase m_1 such that $VOQ_{kj} \prec_{\pi_{m_1}} VOQ_{ij}$. Let $\bar{Q}$ be the complement of Q. By the condition of the lemma, a transition to active for a $VOQ_{kl} \in \bar{Q}$ in matching phase $m' \in [m+1, m+N-1]$ implies $VOQ_{kj} \not\prec_{m'} VOQ_{ij}$. Therefore, any VOQ in Q is π-highest for $\bar{Q}$ during $[m_1, m_1+N-1]$. This means that a π-highest VOQ in any subset $S \subseteq Q$ is also π-highest in $S \cup \bar{Q}$ (so we can ignore $\bar{Q}$ when considering Q). Since π is stable, no transitions occur for VOQs in Q during $[m_1 + 1, m_2]$ by the condition of the lemma. The proof is now similar to the previous argument when no VOQ transitions occur. This time however, starting from the first matching phase m_1, we only consider for the argument the VOQs in Q. The same argument works since no VOQ transitions occur in Q until matching phase $m_2 + 1$ (except possibly for matching phase m_1 itself). Therefore, a π-*stable* matching with respect to all VOQs in Q will be reached in at most N matching phases, i.e. in matching phase $m_1 + N - 1$, and remains as such during $[m_1 + N - 1, m_2]$. ∎

Therefore, Lemma 1 establishes the property that π-RGA with a stable priority scheme π will be able to sustain a π-*stable* (also maximal) matching for a set of highest priority VOQs, as long as they retain their priorities and remain active.

The following section defines another property of the priority scheme π that will be useful for the operation of the π-RGA switching algorithm.

IV. Bounded Bypass Priority Scheme π

In this section we define a bounded bypass priority scheme:

Definition 8—Bounded Bypass π: A priority scheme π is a bounded bypass priority scheme iff it satisfies the following: if VOQ_{ij} remains active during $[m_1, m_2]$, then VOQ_{kl} transitions to active in some matching phase $m \in [m_1 + 1, m_2]$ with $VOQ_{kj} \prec_{\pi_m} VOQ_{ij}$ (i.e. VOQ_{kl} "bypasses" VOQ_{ij}) at most a bounded number of times b.

Note that the bounded bypass property of the priority scheme does not restrict the traffic. Therefore, a VOQ can transition to active at any time; however, it acquires a priority that satisfies the bounded bypass property. A trivial priority scheme that satisfies the bounded bypass property is the one that assigns the lowest priority to a newly active VOQ. In fact, this priority scheme will be investigated in Section V.

As mentioned in the previous section, with only a stable priority scheme, the matching might be perturbed every time there is a VOQ transition. The bounded bypass property limits the number of perturbations experienced by a particular VOQ. Recall (from Lemma 1) that a VOQ will experience perturbation only when a transition for a VOQ with higher priority occurs.

We can loosely bound the number of all VOQ transitions that a VOQ_{ij} experiences for higher priority VOQs as follows:

Lemma 2: Given a stable bounded bypass priority scheme π and a VOQ_{ij} that remains active during $[m_1, m_2]$, the number of VOQ transitions during $[m_1 + 1, m_2]$ occurring in any set of VOQs for which VOQ_{ij} is not π-highest during $[m_1, m_2]$ is at most $(2b + 1)(N^2 - 1)$.

Proof: Since π is a bounded bypass priority scheme, a VOQ can "bypass" VOQ_{ij} at most a bounded number of times b. More precisely, if VOQ_{ij} remains active during $[m_1, m_2]$, VOQ_{kl} can transition to active in some matching phase $m \in [m_1 + 1, m_2]$ with $VOQ_{kj} \prec_{\pi_m} VOQ_{ij}$ only a bounded number of times b. This implies that VOQ_{kl} can transition to inactive in some matching phase $m \in [m_1 + 1, m_2]$ with $VOQ_{kj} \prec_{\pi_{m-1}} VOQ_{ij}$ at most $b + 1$ times; since otherwise, VOQ_{kl} bypasses VOQ_{ij} more than b times by the following reasoning: for any two consecutive matching phases $m \leq m' - 2$ in which VOQ_{kl} transitions to inactive, there must exist a matching phase $m_0 \in [m + 1, m' - 1]$ in which VOQ_{kl} transitions to active, and $VOQ_{kj} \prec_{\pi_{m_0}} VOQ_{ij} \Leftrightarrow VOQ_{kj} \prec_{\pi_{m'-1}} VOQ_{ij}$ by the stable property of π.

Since we have at most $N^2 - 1$ VOQs other than VOQ_{ij}, if VOQ_{ij} remains active during $[m_1, m_2]$, then during $[m_1 + 1, m_2]$ at most $(2b + 1)(N^2 - 1)$ transitions occur in a set of VOQs for which VOQ_{ij} is not π-highest during $[m_1, m_2]$. ∎

If the packet arrival rate of every VOQ is strictly less than 1, an active VOQ that remains π-highest will be served until it becomes inactive. With this added bounded bypass property, a VOQ_{ij} that remains active will eventually become (and remain due to the stable property of π) π-highest, since there will be only a bounded number of transitions for VOQs with higher priority than VOQ_{ij}. As a result, with a stable bounded bypass priority scheme π, π-RGA guarantees that a VOQ will be empty (more precisely inactive) infinitely many times, if the packet arrival rate of every VOQ is strictly less than 1. Therefore, π-RGA avoids starvation (every VOQ becomes inactive infinitely many times) for any admissible traffic satisfying $\sum_k r_{ik} \leq 1$ and $\sum_k r_{kj} \leq 1$ for every i and j, where r is the $N \times N$ rate matrix. We will revisit the issue of starvation later in Section VI.

V. Theoretical Results

Lemma 1 and Lemma 2 imply that the π-RGA switching algorithm satisfies the following *local stability* property with a stable bounded bypass priority scheme π.

Theorem 1—Local Stability: For any stable bounded bypass priority scheme π, if VOQ_{ij} remains active during $[m_1, m_2]$, then the matching computed by the π-RGA switching algorithm is π-*stable* with respect to VOQ_{ij} for every matching phase $m \in [m_1, m_2]$ except for at most a bounded number K of matching phases.

Proof: If π is stable and bounded bypass, then the number of VOQ transitions during $[m_1 + 1, m_2]$ occurring in any set of VOQs for which VOQ_{ij} is not π-highest during $[m_1, m_2]$ is at most $(2b + 1)(N^2 - 1)$, by Lemma 2. Since π is a stable priority scheme, a π-*stable* matching with respect to VOQ_{ij} can be reached in at most N matching phases in the absence of such transitions, as stated in Lemma 1. Therefore, the number of matching phases in $[m_1, m_2]$ for which the matching is not π-*stable* with respect to VOQ_{ij} is at most $(N-1)[(2b+1)(N^2-1)+1] < (2b+1)N^3$. Hence, $K = O(N^3)$. ■

The *local stability* property implies a *local maximality* property. For instance, it implies that, while VOQ_{ij} remains active, either input i is matched or output j is matched except for a bounded number of matching phases. Note that π-RGA might not succeed in computing a maximal matching (unless the number of iterations is sufficient); however, for a particular active VOQ, the matching will always be "locally" maximal except for a bounded number of matching phases.

Next, we enumerate the guarantees of π-RGA under different traffic models. We will describe three traffic models: SLLN, Weak Constant Burst, and Strong Constant Burst. For the traffic models described below, we let $A_{ij}(t)$ denote the number of packets that arrive at input i by time t and are destined to output j. We also let α denote the maximum loading at any input or output port.

A. ASLLN Traffic

The SLLN (Strong Law of Large Numbers) traffic model satisfies the following:

SLLN:
- $\lim_{t\to\infty}(A_{ij}(t)/t) = r_{ij}$ with probability 1
- $\sum_k r_{ik} \leq \alpha$
- $\sum_k r_{kj} \leq \alpha$
- $\alpha \leq 1$

It has been shown in [5] that a maximal matching algorithm guarantees throughput with probability 1 under any SLLN traffic with a speedup $S \geq 2\alpha$. The analysis in [5] is based on a fluid model of the switch, and all what is required from the maximality of the matching is the condition $\dot{C}_{ij}(t) \leq \sum_k r_{ik} + \sum_k r_{kj} - S$ if $Z_{ij}(t) > 0$, where $Z_{ij}(t)$ is the (fluid) number of packets in VOQ_{ij} and $C_{ij}(t) = \sum_k Z_{ik}(t) + \sum_k Z_{kj}(t)$. Intuitively, the condition states that if VOQ_{ij} is not empty, the number of packets at input i and output j changes at a rate $\sum_k r_{ik} + \sum_k r_{kj} - S \leq 0$ when $S \geq 2\alpha$. Note that $\sum_k r_{ik} + \sum_k r_{kj}$ is the combined packet arrival rate at input i and output j, and S (the speedup) is the service rate for those ports. Therefore, the same is true for any switching algorithm that satisfies the *local stability* property: for a large time interval, if VOQ_{ij} remains active (and contains enough packets), S is the service rate of input port i and output port j, since the matching is maximal except for a constant number of matching phases. We will not provide a rigorous proof here since this is not the main result of the paper, the reader may refer to [5] for details on the basic maximality proof. We have the following result: For any stable bounded bypass priority scheme π, the π-RGA switching algorithm guarantees throughput with probability 1 under an SLLN traffic with a speedup $S \geq 2\alpha$.

B. Weak Constant Burst Traffic

The weak constant burst traffic model satisfies the following:

Weak Constant Burst
- $\forall t_1 \leq t_2, \sum_k A_{ik}(t_2) - A_{ik}(t_1) \leq \alpha(t_2 - t_1) + \sigma$
- $\forall t_1 \leq t_2, \sum_k A_{kj}(t_2) - A_{kj}(t_1) \leq \alpha(t_2 - t_1) + \sigma$
- $\alpha \leq 1$

The burst σ is a constant independent of time. It has been shown in [4] that a maximal matching algorithm guarantees a delay bound on every packet under a weak constant burst traffic with a speedup $S > 4\alpha$. The result in [4] relies on the fact that, with a maximal matching, if VOQ_{ij} remains active, at least SD packets are served in D time steps (for input i and output j combined). With a switching algorithm satisfying the *local stability* property, at least $SD - K$ packets will be served. The result in [4] still holds as long as K is a constant independent of time (which is the case here). Therefore, we have the following result: For any stable bounded bypass priority scheme π, the π-RGA switching algorithm guarantees a delay bound on every packet under a weak constant burst traffic with a speedup $S > 4\alpha$. By strengthening the burst condition, we can provide guarantees with a less stringent speedup requirement.

C. Strong Constant Burst Traffic

The strong constant burst traffic model satisfies the following:

Strong Constant Burst
- $\forall t_1 \leq t_2, A_{ij}(t_2) - A_{ij}(t_1) \leq r_{ij}(t_2 - t_1) + \sigma$
- $\sum_k r_{ik} \leq \alpha$
- $\sum_k r_{kj} \leq \alpha$
- $\alpha \leq 1$

Again, the burst σ is a constant independent of time. We will prove stronger results stated in Theorem 2 and Theorem 3.

Let the priority scheme π_0 be the *Earliest Activation Time*[6] priority scheme defined as follows: For any matching phase m, $VOQ_{ij} \prec_{\pi_m} OQ_{kl}$ iff VOQ_{ij} has an earlier activation time than that of VOQ_{kl}, where the activation time of a VOQ is simply the last time at which the VOQ transitioned to active. Ties are broken arbitrarily, but consistently for π_0 to be a stable priority scheme, e.g. using the indices i and j of the input and output ports respectively. It is easy to show that π_0 is a stable bounded bypass priority scheme. In fact, although it is not the only stable bounded bypass priority scheme, π_0 is the most natural one: a newly active VOQ takes the lowest priority. Two important facts about π_0 should be noted.

[6]If activation times are stored explicitly, and a centralized clock is not desired, the effect of a centralized clock can be obtained if each port keeps a local counter and the values of the counters are communicated in the request and grant messages between the ports [11].

First, selecting the highest priority VOQ at a port can be done in parallel in $O(\log N)$ time using $O(N)$ 2-input comparators: The priorities (i.e. activation times) are compared pairwise and the results are successively merged in a tree structure of height $O(\log N)$. Moreover, updating the activation times and the active/inactive status for all VOQs at a port takes $O(1)$ time since at most one packet can arrive at an input port in one time step and at most S (constant) packets can be forwarded to and from a port in one time step. Therefore, with the additional overhead of storing the activation times, π-RGA is as efficient as other simple priority schemes like round robin (used in *iSLIP* and *DRR* for instance).

Second, a particular concern about π_0 is the problem of unbounded activation times. To solve this problem, every port can keep a balanced binary search tree of all its active VOQs. When a VOQ becomes active, it is inserted in the tree. Upon insertion, the activation time of the VOQ is not needed simply because it is the largest (it can be assumed to be ∞). Similarly, when a VOQ becomes inactive, it is deleted from the tree. Upon deletion, the activation time of the VOQ is not needed because deletion only works on the structure of the tree. Both insertion and deletion take $O(\log N)$ time. Determining the highest priority VOQ at a port is equivalent to finding the minimum in the tree (leftmost VOQ), which also takes $O(\log N)$ time. Requests at an input can be issued as follows: Given the previously served VOQ in the tree, the VOQ sends a *Strong* signal to itself and its right subtree, and a *Weak* signal to its left subtree. Moreover, going up the tree towards the root, each parent receives a *Strong* or a *Weak* signal depending on whether it is reached from its left or right child respectively, and in turn propagates the same signal to its remaining subtree. A VOQ that receive a *Strong* signal is issued a *Strong* request, and a VOQ that receives a *Weak* signal is issued a *Weak* request. Since the height of the tree is $O(\log N)$, issuing all requests can be done in parallel in $O(\log N)$ time. Therefore, with the additional overhead of maintaining a balanced binary search tree structure, activation times can be eliminated without sacrificing performance (at least in the theoretical sense).

Theorem 2: With the particular stable bounded bypass priority scheme π_0, the π_0-RGA switching algorithm achieves a bounded length for every VOQ under a strong constant burst traffic with a speedup $S \geq 2\alpha$.

Proof: For a given VOQ_{ij}, if $r_{ij} = 0$, then by the definition of a strong constant burst traffic, $A_{ij}(t) \leq r_{ij}t + \sigma = \sigma$ for any time t. Therefore, the length of VOQ_{ij} cannot exceed σ. So let us assume that $r_{ij} \neq 0$. We will prove that VOQ_{ij} cannot remain active for more than a bounded number of matching phases.

If the traffic satisfies the strong constant burst model, a simple argument can show that if VOQ_{ij} remains active for D matching phases, it acquires less than $1 + r_{ij}(D/S) + \sigma$ packets (complete packets).

Let $Q = \{VOQ_{ij} | r_{ij} > 0\}$. Consider the VOQ_{ij} in Q that is the first to remain active for D matching phases (if more than one VOQ satisfy the property, choose one arbitrarily). Therefore, if VOQ_{ij} transitions to active in matching phase m_0, it remains active during $[m_0, m_0 + D - 1]$.

Recall that the switching algorithm will compute a π_0-*stable*} matching with respect to VOQ_{ij} in every matching phase $m \in [m_0, m_0 + D - 1]$ except for at most a bounded number of matching phases K. As a result, in every matching phase $m \in [m_0, m_0 + D - 1]$, except for at most K matching phases, an active $VOQ_{kj} \not\succ_{\pi_{0m}} VOQ_{ij}$ is served, with $k = i$ or $l = j$. Therefore, and by definition of π_0, in every matching phase $m \in [m_0, m_0 + D - 1]$, either a packet is forwarded from VOQ_{ij}, or a packet is forwarded from VOQ_{ik} with an activation time no later than VOQ_{ij}, or a packet is forwarded from VOQ_{kj} with an activation time no later than VOQ_{ij}.

As a result, at least $D - K$ packets that satisfy the above criterion are forwarded from input i or to output j during $[m_0, m_0 + D - 1]$. A packet pertaining to a VOQ that becomes active during $[m_0 + 1, m_0 + D - 1]$ does not satisfy the above criterion. Moreover, since by the choice of VOQ_{ij}, up to matching phase $m_0 + D - 1$, all VOQs have been active for at most D matching phases, the number of these packets can be bounded as follows:

$$\begin{aligned}
&\left(r_{ij}\frac{D}{S}+\sigma+1\right)+\sum_{\substack{k\neq j\\ VOQ_{ik}\in Q}}\left(r_{ik}\frac{D}{S}+\sigma+1\right)\\
&\quad+\sum_{\substack{k\neq i\\ VOQ_{kj}\in Q}}\left(r_{kj}\frac{D}{S}+\sigma+1\right)+\sum_{\substack{k\neq j\\ VOQ_{ik}\notin Q}}\sigma+\sum_{\substack{k\neq i\\ VOQ_{kj}\notin Q}}\sigma\\
&\quad\leq\left(r_{ij}\frac{D}{S}+\sigma+1\right)+\sum_{k\neq j}\left(r_{ik}\frac{D}{S}+\sigma+1\right)\\
&\qquad+\sum_{k\neq i}\left(r_{kj}\frac{D}{S}+\sigma+1\right)\\
&\quad\leq(2\alpha-r_{ij})\frac{D}{S}+(2N-1)(\sigma+1)
\end{aligned}$$

The bound above is obtained by the property of the strong constant burst traffic, by the fact that all VOQs in Q up to matching phase $M_0 + D - 1$ have been active for at most D matching phases, and by the fact that all VOQ_{kl} not in Q, i.e. with $r_{kl} = 0$, have always a length of at most σ as argued before.

We reach a contradiction if $D - K > (2\alpha - r_{ij})(D/S) + (2N - 1)(\sigma + 1)$ or if $D > S(((2N - 1)(\sigma + 1) + K)/(S - 2\alpha + r_{ij}))$. If we define $r_0 = \min_{VOQ_{ij} \in Q} r_{ij}$, then $D \leq S(((2N - 1)(\sigma + 1) + K)/(S - 2\alpha + r_0))$. If $S \geq 2\alpha$, D is at most $S(((2N-1)(\sigma+1)+K)/(r_0))$ and no VOQ can be the first to remain active for more than D matching phases. As a result, the length of VOQ_{ij} cannot exceed $r_{ij}(((2N - 1)(\sigma + 1) + K)/(r_0)) + \sigma + 1$ by the property of the strong constant burst traffic. ■

Theorem 3: With the particular stable bounded bypass priority scheme π_0, the π_0-RGA switching algorithm achieves a bounded delay for every packet under a strong constant burst traffic with a speedup $S > 2\alpha$.

Proof: If we change the definition of the set Q in the proof of Theorem 2 to be the set of all VOQs, then we prove that any VOQ_{ij} cannot remain active for more than $D = S(((2N - 1)(\sigma + 1) + K)/(S - 2\alpha + r_0))$ matching phases, where possibly now $r_0 = 0$. Therefore, for any $S > 2\alpha$, D is a well defined bound. As a consequence, a packet cannot remain in its VOQ for more than D matching phases; otherwise, its VOQ will remain active for more than D matching phases, a contradiction. ■

If we add a technical assumption to the strong constant burst traffic, we can obtain the delay guarantee of Theorem 3 with $S \geq 2\alpha$ (instead of $S > 2\alpha$). The technical assumption is the following: If $r_{ij} = 0$, then $A_{ij}(t) = 0$ at all times. By adding this assumption, we only worry about VOQ_{ij} such that $r_{ij} \neq 0$. We know from Theorem 2 that such a VOQ cannot remain active for more than $D = S(((2N-1)(\sigma+1)+K)/(S-2\alpha+r_0))$ matching phases, where $r_0 \neq 0$. For any $S \geq 2\alpha$, this is a well defined bound, and thus a packet cannot remain in its VOQ for more than D matching phases as argued in the proof of Theorem 3.

Note that π_0 is a zero bypass priority scheme, i.e. a VOQ can never bypass another VOQ; simply because a newly active VOQ will always have the lowest priority (the largest activation time). Hence, once a matching is maximal, it will remain maximal until a VOQ transitions to inactive. This was experimentally found to be useful. The following section provides some of the experimental results comparing the relative performance of π-RGA to other iterative switching algorithms found in the literature.

VI. Experimental Results

While the results mentioned above hold for a speedup of 2, we simulated the π-RGA switching algorithm with no speedup and with the priority scheme $\pi = \pi_0$ defined earlier. In the rest of this paper, π-RGA actually refers to π_0-RGA. The simulations for π-RGA represent a slight variation from the algorithm depicted in Fig. 3, namely, we chose to capture the state of a VOQ more accurately (see footnote 4). In our simulations, if a VOQ is empty by the end of matching phase m and non-empty at the beginning of matching phase $m+1$, we count this as a transition to active in matching phase $m+1$, even if the VOQ was active at the beginning of matching phase m. Therefore, the activation time of the VOQ is updated and, as a result, the VOQ will have the latest activation time (i.e. lowest priority) among all the VOQs at the beginning of matching phase $m+1$.

The simulations showed that π-RGA with no speedup is capable of sustaining fairly high loads with only one iteration. We will show performance comparisons between π-RGA and *iSLIP*, *DRR*, *PIM*, *LQD*, *iLQF*, *pDRR*, and *iOCF*.

iSLIP: Each input issues requests for all its non-empty VOQs. If an output receives any requests, it grants the one that appears next in a fixed round robin order, starting from the current position of the pointer. The pointer is incremented to one location beyond the granted input if and only if the grant was accepted. If an input receives any grants, it accepts the one that appears next in a fixed round robin order, starting from the current position of the pointer. The pointer is incremented to one location beyond the accepted output.

DRR: *DRR* is the same as *iSLIP* except that the RGA arbitration is started at the output side.

PIM: Each input issues requests for all its non-empty VOQs. If an output receives any requests, it grants one uniformly at random. If an input received any grants, it accepts one uniformly at random.

LQD: *LQD* is a one iteration algorithm where the RGA arbitration is started at the output side. Each output issues requests for all its non-empty VOQs. If an input receives any requests, it grants one request, say for VOQ_{ij}, with probability $|VOQ_{ij}|/\max(\sum_k |VOQ_{ik}|, \sum_k |VOQ_{kj}|)$. If an output receives any grants, it accepts one uniformly at random.

iLQF: Each input issues requests for all its non-empty VOQs. If an output receives any requests, it grants the one corresponding to a VOQ with the largest number of packets and ties are broken uniformly at random. If an input receives any grants, it accepts the one corresponding to the VOQ with the largest number of packets and ties are broken uniformly at random.

pDRR: Each input issues requests for all its non-empty VOQs. If an output receives any requests, it grants the one corresponding to the VOQ with the largest number of packets and that appears next in a fixed round robin order, starting from the current position of the pointer. The pointer is incremented to one location beyond the granted input if and only if the grant was accepted. If an input receives any grants, it accepts the one corresponding to the VOQ with the largest number of packets and that appears next in a fixed round robin order, starting from the current position of the pointer. The pointer is incremented to one location beyond the accepted output.

iOCF: *iOCF* is the same as *iLQF* except that instead of using the (largest) length of a VOQ as a priority, it uses the (smallest) timestamp of the head of line packet of a VOQ.

Our comparisons were done using a $16 \times 16 (N = 16)$ switch with **one iteration only**[7] and five different traffic patterns, all of which are SLLN traffic: In each time step, a packet arrives at a given input i with probability α, where $0 \leq \alpha \leq 1$ represents the loading of the switch. If a packet arrives at input i, it is placed in VOQ_{ij} with probability p_{ij}, where $\sum_j p_{ij} = 1$. Therefore, $r_{ij} = \lim_{t\to\infty}(A_{ij}(t)/t) = \alpha p_{ij}$. The five traffic patterns are as follows:

- Uniform Balanced: For every input i and every output j, $r_{ij} = \alpha/N$.
- Non-uniform Balanced: For every input i, it is possible to order the outputs from 1 to N such that $r_{i1} = \rho r_{i2} = \rho^2 r_{i3} = \ldots = \rho^{N-1} r_{iN}$, where $0 < \rho \leq 1$. The same is true for every output j. When $\rho = 1$, this is a uniform balanced traffic. We used $\rho = 1/2$.
- Uniform Unbalanced: Each input sends packets to $n < N$ outputs and each output receives packets from n inputs. For every input i and every output j, if i sends packets to j, then $r_{ij} = \alpha/n$. We used $n = N/2 = 8$ and the $nN = 128$ flows $(1,9)\ldots(1,16),(2,10)\ldots(2,1),\ldots,(16,8)\ldots(16,15)$.
- Non-uniform Unbalanced: Each input sends packets to $n < N$ outputs and each output receives packets from n inputs. If input i sends packets to output j, then call j a participating output of i. A participating input is defined in a similar way. For every input i, it is possible to order the

[7]It is not fair to compare π-RGA with one iteration to another algorithm with multiple iterations. Otherwise, the latter will outperform π-RGA simply because it is likely to reach a maximal matching. Furthermore, it is not the premise of this work to allow both algorithms to use multiple iterations, because if multiple iterations exist, one could design a better algorithm than π-RGA.

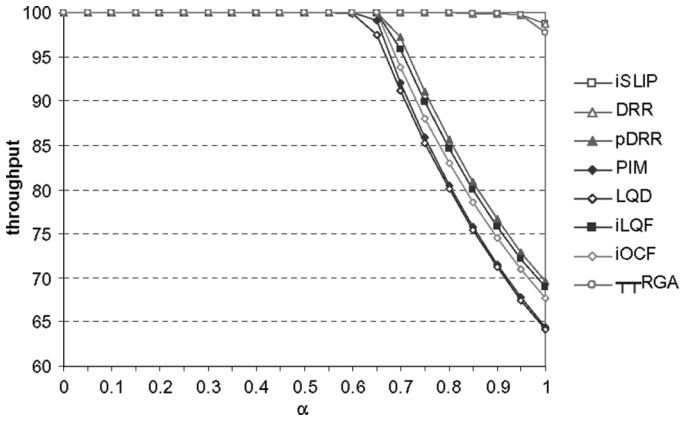

Fig. 4. Throughput with uniform balanced traffic.

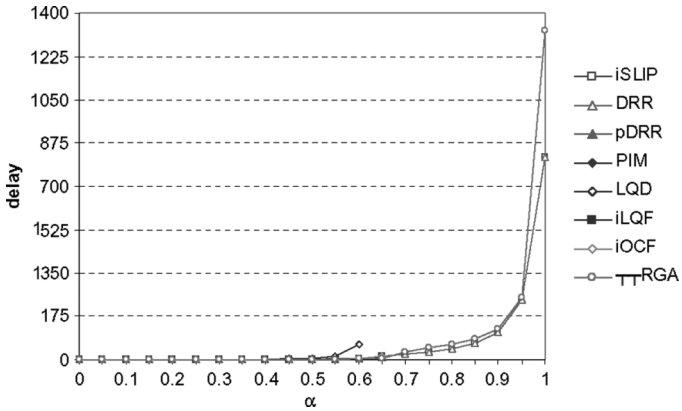

Fig. 5. Delay with uniform balanced traffic.

participating outputs from 1 to n such that $r_{i1} = \rho r_{i2} = \rho^2 r_{i3} = \ldots = \rho^{n-1} r_{in}$, where $0 < \rho \leq 1$. The same is true for every output j. When $\rho = 1$, this is a uniform unbalanced traffic. We used $\rho = 1/2$ and $n = N/2 = 8$ and the same flows above.

- Non-admissible traffic: This traffic is used to explore the issue of starvation that π-RGA might cause when the switch is overloaded (see last paragraph of Section IV). It is exactly the Non-uniform Balanced traffic with one added modification: the switch is idle in every other time step. Therefore, if $\alpha > 1/2$, the switch is overloaded. For instance, if $\alpha = 1$, the best throughput that can be achieved for such traffic is 50%.

For a given value of α, we compute **throughput** as the percentage of served packets, and **delay** as the average delay of served packets (the delay of a packet is the difference between its arrival time step and departure time step, i.e. zero if it departs immediately). The percentage of served packets is a good indication of throughput. Ideally speaking, if we let μ to be the percentage of served packets when $\alpha = 1$, then μ should be the throughput of the switch. In all of our simulations, we observed that when $\alpha > \mu$, the percentage of served packets drops considerably below 100% and the average delay increases substantially, implying that μ is in deed a good measure of the throughput.

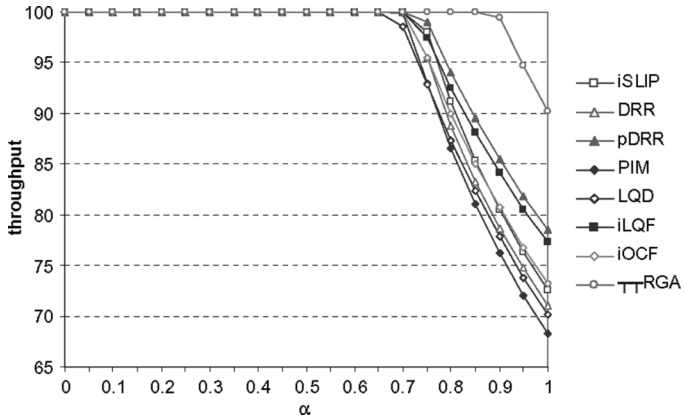

Fig. 6. Throughput with non-uniform balanced traffic.

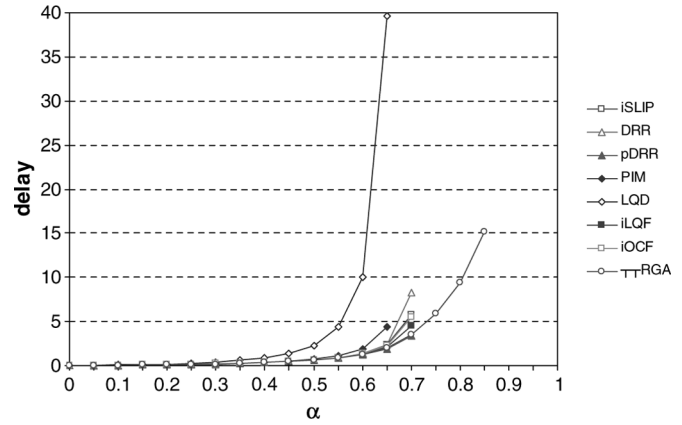

Fig. 7. Delay with non-uniform balanced traffic.

The results for each of the traffic patterns described above are based on an average of **100** simulations (10^5 time steps each). We produce throughput and delay plots for values of α varying from 0 to 1 (21 points). Delay plots are truncated at the point before a 10 fold increase, hence possibly reflecting a throughput lower than the actual throughput.

A. Uniform Balanced Traffic

With a uniform balanced traffic, *iSLIP* and *DRR* perform the best. Note that this traffic represents the theoretical condition for the two algorithms to achieve 100% throughput. The simulation shows that this is true in deed (98.76%). π-RGA has a comparable performance to *iSLIP* and *DRR*, showing about 97.78% throughput. Both throughputs converge to 100% when increasing the simulation time indicating that π-RGA also achieves 100% throughput under a uniform traffic. The rest of the algorithms achieve less than 70% throughput. Refer to Figs. 4 and 5.

B. Non-Uniform Balanced Traffic

With a non-uniform balanced traffic, the throughput of *iSLIP* and *DRR* drops (expectedly) to below 75%. π-RGA achieves slightly above 90% throughput. The rest of the algorithms achieve less than 80% throughput. Refer to Figs. 6 and 7.

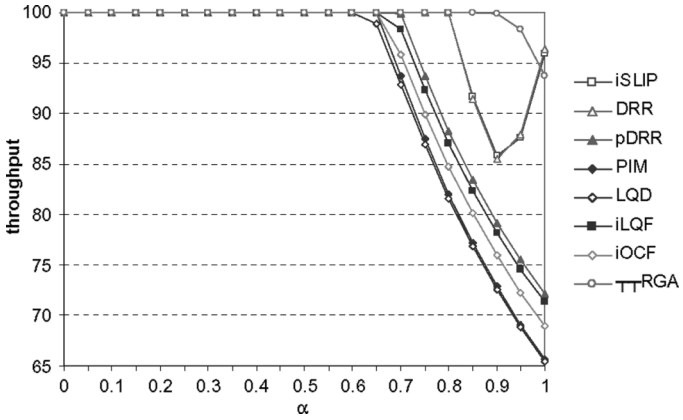

Fig. 8. Throughput with uniform unbalanced traffic.

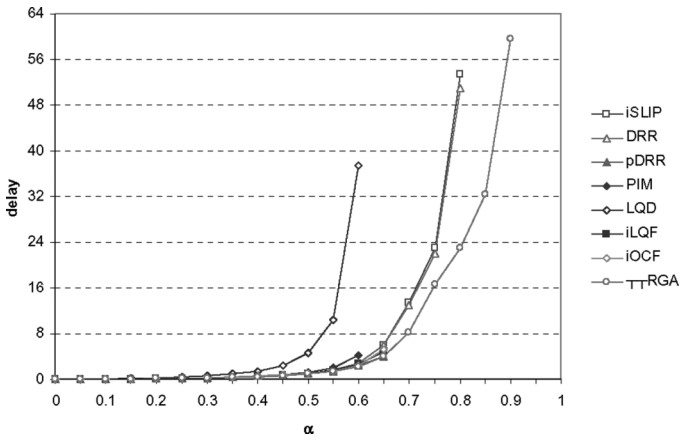

Fig. 9. Delay with uniform unbalanced traffic.

C. Uniform Unbalanced Traffic

With a uniform unbalanced traffic, the throughput of π-RGA is about 93.79%. Except for *iSLIP* and *DRR*, the performance of π-RGA at high loading is better than all other algorithms (achieving less than 75% throughput). Refer to Figs. 8 and 9.

The reported throughput is the average of a 100 simulated throughput values. This average shows that both *iSLIP* and *DRR* achieve higher throughput ($\approx 96.5\%$) than that of π-RGA when α is close to 1. For instance, all active VOQs receive packets at the same rate (see description of a uniform unbalanced traffic above); therefore, since α is close to 1, no VOQ is likely to become empty if all VOQs are served equally. Moreover, if all VOQs remain non-empty, when N VOQs are served in one matching, the round robin pointers may lock into a periodic behavior where N VOQs are served in every matching. Therefore, the traffic acts like a uniform traffic for *iSLIP* and *DRR*, and all VOQs will be served equally. The throughput in that case is theoretically 100%.

However, approximately 13% of the simulations (a total of 1000 simulations) show that the throughput of *iSLIP* and *DRR* can be as low as 75% under a uniform unbalanced traffic pattern ($0.13 * 75\% + 0.87 * 100\% = 96.75\%$), while the throughput of π-RGA never drops below 93%. Fig. 10 supports this observation and shows the standard deviation of a 100 simulated

	iSLIP	*DRR*	*pDRR*	*PIM*	*LQD*	*iLQF*	*iOCF*	*π-RGA*
Traffic A	0.034	0.028	0.024	0.026	0.029	0.027	0.026	0.036
Traffic B	0.026	0.027	0.021	0.029	0.027	0.020	0.025	0.673
Traffic C	**8.026**	**7.767**	0.025	0.023	0.029	0.023	0.027	**0.231**
Traffic D	0.026	0.029	0.022	0.027	0.027	0.023	0.023	0.718

Fig. 10. Standard deviations of throughputs.

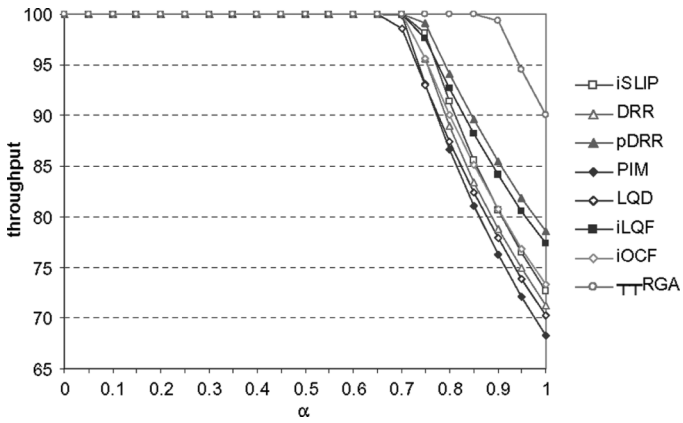

Fig. 11. Throughput with non-uniform unbalanced traffic.

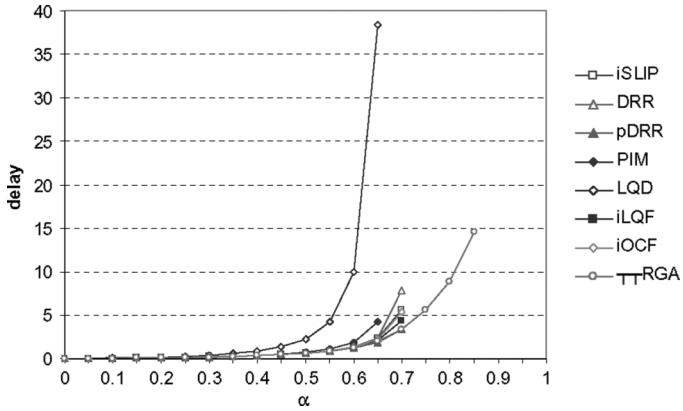

Fig. 12. Delay with non-uniform unbalanced traffic.

throughput values for each pair of traffic and switching algorithm (standard deviations in bold are based on 1000 simulations).

D. Non-Uniform Unbalanced Traffic

With a non-uniform unbalanced traffic, π-RGA achieves slightly above 90% throughput. The rest of the algorithms achieve below 80% throughput. Refer to Figs. 11 and 12.

E. Non-Admissible Traffic

As mentioned at the end of Section IV, π-RGA avoids starvation (every VOQ is inactive infinitely many times) with an admissible traffic satisfying $\sum_k r_{ik} \leq 1$ and $\sum_k r_{kj} \leq 1$. However, if the traffic is not admissible, i.e. the switch is overloaded, π-RGA can lead to starvation. For instance, if the traffic is non-uniform such that a given VOQ receives most of the packets, once that VOQ starts getting service, it will continue to be served by π-RGA forever because it will never be emptied. This implies that other VOQs at the same input (and output)

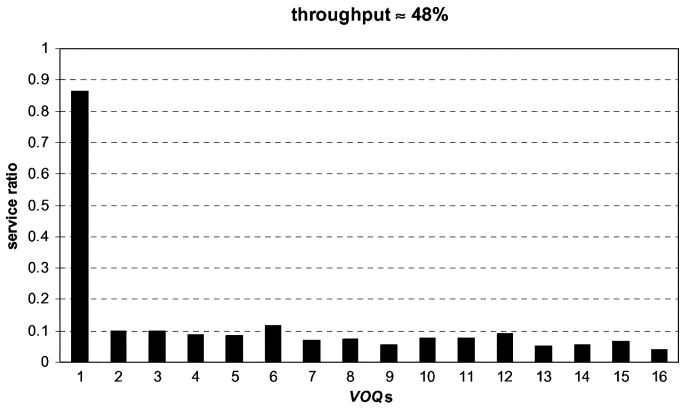

Fig. 13. Service ratios with π-RGA under non-admissible traffic.

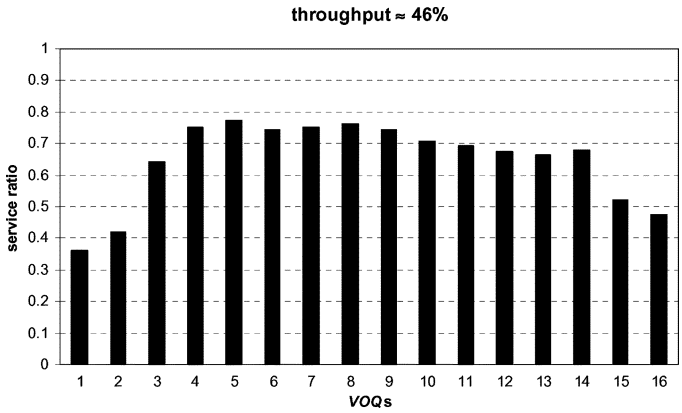

Fig. 14. Service ratios with modified π-RGA under non-admissible traffic.

never get a chance to be served, leading to starvation. Such behavior can be easily obtained with the non-admissible traffic described earlier. Fig. 13 illustrates the service ratio (the number of packets served divided by the total number of packets) for every VOQ at input 1 under such traffic with $\alpha = 1$.

It is obvious from Fig. 13 that the first VOQ starves all other VOQs at input 1. In fact, since the first VOQ receives approximately half the packets at input 1 (rates decrease geometrically with $\rho = 1/2$), the throughput ($\approx$48%) is almost entirely due to serving the first VOQ. Ideally, we would like the service ratio of each VOQ to be equal to the throughput, and the throughput to be as close as possible to 50% (which is the best we can achieve under this non-admissible traffic).

Fortunately, we can solve the problem of starvation under a non-admissible traffic pattern by slightly modifying the π-RGA algorithm. The modification is the following: if a VOQ is served for M successive matching phases, its activation time is set to the current time (i.e. it is assigned the lowest priority according to π_0). When this happens, other VOQs will get a chance to be served. The choice of M must not affect the theoretical guarantees of π-RGA. This is possible since under a speedup of $S \geq 2\alpha$, every VOQ remains active for at most $S(((2N-1)(\sigma+1)+K)/(S-2\alpha+r_0))$ matching phases (proof of Theorem 2); therefore, M can be set accordingly (if $S = 2\alpha$, however, the knowledge of the smallest rate r_0 is required). As a practical choice, we set $M = N^3$ (i.e. 4096 in our case) since this is the order of the bound mentioned above. Fig. 14 shows the service ratios for the VOQs of input 1 after this modification under the non-admissible traffic pattern. Results similar to those in Figs. 13 and 14 were observed for the VOQs at output 1.

	π-*RGA*	*WFA*
Traffic A	97.78%	97.09%
Traffic B	90.19%	81.41%
Traffic C	93.79%	92.87%
Traffic D	90.13%	81.27%

Fig. 15. Throughputs of π-RGA and *WFA* (average of 100 simulations).

Clearly, this modification to π-RGA solves the starvation problem for the non-admissible traffic pattern, although a slight reduction in throughput is observed (but it remains better than half the throughput achieved under a non-uniform balanced traffic, which is the admissible version of this traffic). We verified by simulation that this modification does not affect the results for the other traffic patterns.

VII. A Pipelining Argument to π-RGA

While π-RGA does not explicitly use pipelining, with the *Earliest Activation Time* priority scheme π_0, one can draw a conceptual connection to the process of pipelining a maximal matching, in a way similar to *WFA* (and its variant *WWFA*) [18] for instance. *WFA* does not belong to the RGA arbitration type switching algorithms; it consists of N^2 crosspoints arranged in an $N \times N$ matrix, where crosspoint (i, j) corresponds to VOQ_{ij}. The arbitration with *WFA* begins at the top priority crosspoint (1,1) and proceeds in a "wave front" that moves diagonally from the top left to the bottom right of the matrix. Therefore, a maximal matching is reached in $O(N)$ iterations using a fixed priority scheme (VOQ_{11} has the highest priority and VOQ_{NN} has the lowest priority) [18].

In the pipelined version of *WFA*, crosspoint (i, j) can start working on iteration $k + 1$ at the same time as crosspoints $(i, j+1)$ and $(i+1, j)$ start working on iteration k. Therefore, although the computation of the matching takes $O(N)$ iterations, the pipelined version of *WFA* produces a maximal matching per iteration. Moreover, if no VOQ transitions occur, this maximal matching will be the same over successive iterations (due to the fixed priority scheme). However, the highest priority crosspoint (i.e. VOQ) can rotate among the crosspoints periodically [18], hence disturbing the pipelining (and the maximal matching) for a number of iterations, but only to converge back to another maximal matching.

This is conceptually similar to what π-RGA does with the *Earliest Activation Time* priority scheme π_0 and, therefore, gives a strong argument to the π-RGA approach: with π-RGA and π_0, the matching converges to a maximal matching as long as no VOQ becomes inactive. The matching is disturbed upon such an event (this is when priorities change), only to converge back to another maximal matching. Conceptually, the difference between the two is that π-RGA assigns priorities based on the activation times rather than using a rotating priority scheme as in *WFA*. Fig. 15 shows the comparison in throughput for both algorithms. We allow *WFA* to compute a maximal matching every iteration and we rotate the priority every one iteration.

VIII. Conclusion

Theoretically speaking, with a particular priority scheme π_0 being the *Earliest Activation Time*, the π-RGA algorithm achieves bounded VOQ length (throughput) and a delay guarantee with a speedup $S \geq 2$, under a constant burst traffic. The π-RGA algorithm requires $O(\log N)$ computational complexity to update the priorities and select the highest priority request, with the use of appropriate parallelism at the ports. Since only one iteration is needed in each matching phase (or more generally, a constant number of iterations), the computational complexity of the π-RGA switching algorithm is $O(\log N)$.

Practically speaking, the π-RGA algorithm achieves high throughput with no speedup, only one iteration, and the *Earliest Activation Time* priority scheme π_0. We assess the relative performance of π-RGA to other iterative switching algorithms through simulation, under the one iteration limit, and with different traffic patterns: π-RGA consistently achieves higher throughput and lower delay than the other algorithms.

References

[1] T. E. Anderson, S. Owicki, J. Saxes, 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] J. Blanton, H. Badt, G. Damm, and P. Golla, "Iterative scheduling algorithms for optical packet switches," presented at the IEEE ICC 2001, Helsinki, Finland, Jun. 2001.

[3] J. Blanton, H. Badt, G. Damm, and P. Golla, "Impact of polarized traffic on scheduling algorithms for high speed optical switches," presented at the ITCom 2001, Denver, CO, Aug. 2001.

[4] A. Charny, P. Krishna, N. Patel, and R. Simcoe, "Algorithms for providing bandwidth and delay guarantees in input-buffered crossbars with speedup," in *Proc. 6th Int. Workshop on Quality of Service (IWQOS'98)*, May 1998, pp. 235–244.

[5] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. IEEE INFOCOM*, 2000, pp. 556–564.

[6] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American Mathematical Monthly*, vol. 69, pp. 9–15, 1962.

[7] G. Damm, J. Blanton, P. Golla, D. Verchere, and M. Yang, "Fast scheduler solutions to the problem of priorities for polarized data traffic," presented at the Int. Symp. Telecommunications (IST 2001), Tehran, Iran, 2001.

[8] P. Giaccone, B. Prabhakar, and D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth switches," in *Proc. IEEE INFOCOM*, 2002, pp. 1160–1169.

[9] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, Jan.-Feb. 1999.

[10] I. Keslassy, M. Kodialam, T. V. Lakshman, and D. Stiliadis, "On guaranteed smooth scheduling for input-queued switches," in *Proc. IEEE INFOCOM*, 2003, pp. 1384–1394.

[11] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *CACM*, vol. 21, no. 7, pp. 558–565, 1978.

[12] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Trans. Networking*, vol. 9, no. 1, pp. 104–118, Feb. 2001.

[13] Y. Li, S. Panwar, and H. J. Chao, "Saturn: a terabit packet switch using dual round robin," *IEEE Commun. Mag.*, vol. 38, no. 12, pp. 78–84, Dec. 2000.

[14] N. McKeown, "Scheduling algorithms for input queued cell switches," Ph.D. dissertation, Univ. California, Berkeley, May 1995.

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

[16] S. Mneimneh, V. Sharma, and K.-Y. Siu, "Switching using parallel input-output queued switches with no speedup," *IEEE/ACM Trans. Networking*, vol. 10, no. 5, pp. 653–665, Oct. 2002.

[17] S. Mneimneh and K. Y. Siu, "On achieving throughput in an input-queued switch," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 858–867, Oct. 2003.

[18] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, pp. 13–27, Jan. 1993.

[19] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proc. IEEE INFOCOM*, 1998, vol. 2, pp. 533–539.

Saad Mneimneh, photograph and biography not available at the time of publication.