Sliding-Window QPS (SW-QPS): A Perfect Parallel Iterative Switching Algorithm for Input-Queued Switches

Jingfan Meng
Georgia Tech
jmeng40@gatech.edu

Long Gong
Georgia Tech
gonglong@gatech.edu

Jun (Jim) Xu
Georgia Tech
jx@cc.gatech.edu

ABSTRACT

In this work, we first propose a parallel batch switching algorithm called Small-Batch Queue-Proportional Sampling (SB-QPS). Compared to other batch switching algorithms, SB-QPS significantly reduces the batch size without sacrificing the throughput performance and hence has much lower delay when traffic load is light to moderate. It also achieves the lowest possible time complexity of $O(1)$ per matching computation per port, via parallelization. We then propose another algorithm called Sliding-Window QPS (SW-QPS). SW-QPS retains and enhances all benefits of SB-QPS, and reduces the batching delay to zero via a novel switching framework called sliding-window switching. In addition, SW-QPS computes matchings of much higher qualities, as measured by the resulting throughput and delay performances, than QPS-1, the state-of-the-art regular switching algorithm that builds upon the same underlying bipartite matching algorithm.

Keywords

Switching, input-queued switch, bipartite matching

1. INTRODUCTION

Many present day switching systems in Internet routers and data-center switches employ an input-queued crossbar to interconnect their input ports and output ports. In an $N \times N$ input-queued crossbar switch, each input port has N Virtual Output Queues (VOQs). A VOQ j at input port i serves as a buffer for the packets going into input port i destined for output port j.

In an $N \times N$ input-queued crossbar switch, each input port can be connected to only one output port and vice versa in each switching cycle or time slot. Hence, in every time slot, the switch needs to compute a one-to-one *matching* (*i.e.*, the crossbar schedule) between input and output ports . A major research challenge of designing high-link-rate switches with a large number of ports (called *high-radix* [3]) is to develop switching algorithms that can compute "high quality" matchings – those that result in high switch throughput and low queueing delays for packets – in a short time slot.

1.1 Existing Approaches

While many switching algorithms have been proposed for input-queued switches, they either have a (relatively) high time complexity that prevents a matching computation from being completed in a short time slot, or cannot produce high-quality matchings that translate into excellent throughput and delay performances. For example, the widely-used iSLIP algorithm [13] can empirically achieve over 80% throughputs under most of traffic patterns, as will be shown in §6.2. However, even with a parallel iterative implementation, its time complexity per port is $O(\log^2 N)$, which is still too high when the switch size N is large and the time slot is short (say a few nanoseconds long).

It is possible to improve the quality of the matching without increasing the time complexity of the switching algorithm using a strategy called batching [1, 16, 18]. Unlike in a regular switching algorithm, where a matching decision is computed for every time slot, in a batch switching algorithm, multiple (say T) consecutive time slots are grouped as a batch and these T matching decisions are batch-computed. Each matching under computation has a period of T time slots to find opportunities to improve the quality of the matching by the underlying bipartite matching algorithm, whereas in a regular switching algorithm, it only has one time slot for such opportunities. Hence, a batch switching algorithm can usually produce matchings of higher qualities (and the quality is better when T is larger) than a regular switching algorithm using the same underlying bipartite matching algorithm, because the opportunities do not all present themselves in a single designated time slot.

However, existing batch switching algorithms are not without shortcomings. They all suffer from at least one of the following two problems. First, all existing batch switching algorithms except [18] are serial algorithms and it is not known whether any of them can be parallelized. As a result, they all have a time complexity of at least $O(N)$ per matching computation, since it takes $O(N)$ time just to "print out" the computed result. This $O(N)$ time complexity is clearly too high for high-radix high-line-rate switches as just explained. Second, most existing switching algorithms require a large batch size T to produce high-quality matchings that can lead to high throughputs, as will be elaborated in §5. A large batch size T is certain to lead to poor delay performance: Regardless of the offered load condition, the average packet delay for any batch switching algorithm due to batching is at least $T/2$, since any packet belonging to the current batch has to wait till at least the beginning of the next batch to be switched.

1.2 Our Contributions

The first contribution of this work is a novel batch switching algorithm, called SB-QPS (Small-Batch QPS), that addresses both weaknesses of existing batch switching algorithms. First, it can attain a high throughout of over 85%, under various traffic load patterns, using only a small batch size of $T = 16$ time slots. This much smaller batch size translates into much better delay performances than those of existing batch switching algorithms, as will be shown in §6.3. Second, SB-QPS is a fully distributed algorithm so that the matching computation load can be efficiently divided evenly across the $2N$ input and output ports. As a result, its time complexity is the lowest possible: $O(1)$ per matching computation per port.

The design of the SB-QPS algorithm is extremely simple. Only T rounds of request-accept message exchanges by the input and the output ports are required for computing the T matchings used (as the crossbar configurations) in a batch of T time slots. In each round, each input port i sends a pairing request to an output port that is sampled (by input port i) in a random queue-proportional fashion: Each output port j is sampled with a probability proportional to the length of the corresponding VOQ. For this reason, we call this algorithm small-batch QPS (queue-proportional sampling). Since each QPS operation can be performed in $O(1)$ time using a simple data structure as shown in [9], the time complexity of SB-QPS is $O(1)$ per matching computation per port. As will be explained in §3.2, the way QPS is used in this work (SB-QPS) is very different than that in [9]. For one thing, whereas in [9] QPS is used as an auxiliary component to other switching algorithms such as iSLIP [13] and SERENA [7], in this work, QPS serves the primary building block for SB-QPS.

Even though SB-QPS has a much smaller batching delay than other batch switching algorithms due to its much smaller T, the batching delay accounts for the bulk of the total packet delay under light to moderate traffic loads, when all other delays are comparatively much smaller. The second contribution of the work is a novel switching algorithm called SW-QPS (SW for sliding window) that inherits and enhances all the good features of SB-QPS yet pays zero batching delay. More precisely, it has the same $O(1)$ time complexity as and achieves strictly better throughput and delay performances than SB-QPS.

SW-QPS does so by solving the switching problem under a novel framework called *sliding-window switching*. A sliding-window switching algorithm is different than a batch algorithm only in the following aspect. In a batch switching algorithm, a batch of T matchings are produced every T time slots. In contrast, in a sliding-window switching algorithm, each window is still of size T but a single matching is produced every time slot just like in a regular switching algorithm. More precisely, at the beginning of time slot t, the sliding window contains matchings-under-computation for the T time slots t, $t+1$, $\cdots$, $t+T-1$. The "leading edge of the window", corresponding to the matching for the time slot t (the "senior class"), "graduates" and is used as the crossbar configuration for the current time slot t. Then at the end of time slot t, a new and currently empty matching is added to the "tail end of the window" as the "freshman class". This matching will be computed in the next T time slots and hopefully becomes a high-quality matching by the time $t+T$, when it "graduates". SW-QPS completely removes the batching delay because "it graduates a class every year" and furthermore always schedules an incoming packet to "graduate" at the earliest "year" possible.

We consider SW-QPS to be the only research outcome of this work, since it strictly outperforms SB-QPS. However, we describe both SB-QPS and SW-QPS in detail for two reasons. First, the incremental contributions of SB-QPS over existing batch switching algorithms and that of SW-QPS over SB-QPS are orthogonal to each other: The former is to significantly reduce the batch size without sacrificing the throughput performance much and to reduce the time complexity to $O(1)$ via parallelization, whereas the latter is to retain the full benefits of batching without paying the batching delay. Second, thanks to this orthogonality, explaining the differences between SB-QPS and existing batch switching algorithms and that between SW-QPS and SB-QPS separately and incrementally makes the presentation much easier, as will become apparent in §3 and §4.

The rest of this paper is organized as follows. In §2, we state assumptions and the problem model used in this work. §3 and §4 detail the SB-QPS and SW-QPS algorithms respectively. In §5, we survey the related works. Then, we evaluate the performances of SB-QPS and SW-QPS in §6 and in §7, we conclude this paper.

2. ASSUMPTIONS AND PROBLEM MODEL

In this work, we make the following two assumptions that are widely adopted in the literature (*e.g.*, [10, 13]). First, we assume that all incoming variable-length packets are first segmented into fixed-length packets, which are then reassembled before leaving the switch. Hence, we consider the switching of only fixed-length packets in the sequel, and each such fixed-length packet takes exactly one time slot to transmit. Second, we assume that input ports, output ports and the crossbar operate at the same speed.

An $N \times N$ input-queued crossbar can be modeled as a weighted bipartite graph, of which the two disjoint vertex sets are the N input ports and the N output ports respectively. We note that the edge set in this bipartite graph might change from a time slot to another. In this bipartite graph during a certain time slot t, there is an edge between input port i and output port j, if and only if the j^{th} VOQ at input port i, the corresponding VOQ, is nonempty (at t). The weight of this edge is defined as the length of (*i.e.*, the number of packets buffered at) this VOQ. A set of such edges constitutes a *valid crossbar schedule*, or a *matching*, if any two of them do not share a common vertex.

3. SMALL-BATCH QPS

3.1 Batch Switching Algorithms

	O_1	O_2	$\cdots$	O_N
1	I_3	I_7	$\cdots$	I_1
2	I_5	–	$\cdots$	I_3
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
T	–	I_5	$\cdots$	I_2

Figure 1: A joint calendar. "–" means unmatched.

Since Small-Batch QPS (SB-QPS) is a batch switching algorithm [1, 16, 18], we first provide some background on batch switching. In a batch switching algorithm, the T matchings for a batch of T future time slots are batch-computed. These T matchings form a joint calendar (schedule) of the N output ports that can be encoded as a $T \times N$ table with TN cells in it, as illustrated by an example shown in Figure 1. Each column corresponds to the calendar of an output port and each row a time slot. The content of the cell at the intersection of the t^{th} row and the j^{th} column is the input port that O_j is to pair with during the t^{th} time slot in this batch. Hence, each cell also corresponds to an edge (between the input and the output port pair) and each row also corresponds to a matching (under computation for the corresponding time slot). In the example shown in Figure 1, output port O_1 is to pair with I_3 during the 1^{st} time slot (in this batch), I_5 during the 2^{nd} time slot, and is unmatched during the T^{th} time slot.

At each input port, all packets that were in queue before a cutoff time (for the current batch), including those that belong to either the current batch, or previous batches but could not be served then, are waiting to be inserted into the respective calendars (*i.e.,* columns of cells) of the corresponding output ports. The design objective of a batch switching algorithm is to pack as many such packets across the N input ports as possible into the TN cells in this joint calendar. After the computation of the current joint calendar is completed, the T matchings in it will be used as the crossbar configurations for a batch of T future time slots. In the meantime, the switch is switching packets according to the T matchings specified in a past joint calendar that was computed earlier.

3.2 The SB-QPS Algorithm

In this section, we describe in detail SB-QPS, a batch switching algorithm that uses a small constant batch size T that is independent of N. SB-QPS is a parallel iterative algorithm: The input and output ports run T QPS-like iterations (request-accept message exchanges) to collaboratively pack the joint calendar. The operation of each iteration is extremely simple: Input ports request for cells in the joint calendar, and output ports accept or reject the requests. More precisely, each iteration of SB-QPS, like that of QPS [9], consists of two phases: a proposing phase and an accepting phase.

Proposing Phase. We adopt the same proposing strategy as in QPS [9]. In this phase, each input port, unless it has no packet to transmit, proposes to *exactly one* output port that is decided by the QPS strategy. Here, we will only describe the operations at input port 1; those at any other input port are identical. Like in [9], we denote by $m_1, m_2, \cdots, m_N$ the respective queue lengths of the N VOQs at input port 1, and by m their sum (*i.e.,* $m \triangleq \sum_{k=1}^{N} m_k$). At first, input port 1 simply samples an output port j with probability m_j/m, *i.e.,* proportional to m_j, the length of the corresponding VOQ; it then sends a proposal to output port j. The content of the proposal in SB-QPS is slightly different than that in QPS. In QPS, the proposal contains only the VOQ length information (*i.e.,* the value m_j), whereas in SB-QPS, it contains also the following availability information (of input port 1): Out of the T time slots in the batch, what (time slots) are still available for input port 1 to pair with an output port? The time complexity of this QPS operation, carried out using the data structure proposed in [9], is $O(1)$ per input port.

Accepting Phase. In SB-QPS, the accepting phase at an output port is quite different than that in QPS [9]. Whereas the latter allows at most one proposal to be accepted at any output port (as QPS is a part of a regular switching algorithm that is concerned with only a single time slot at a time), the former allows an output port to accept multiple (up to T) proposals (as each output port has up to T cells in its calendar to be filled). Here, we describe the accepting phase at output port 1; that at any other output port is identical. The operations at output port 1 depend on the number of proposals it receives. If output port 1 receives exactly one proposal from an input port (say input port i), it tries to accommodate this proposal using an accepting strategy we call *First Fit Accepting* (FFA). The FFA strategy is to match in this case input port i and output port 1 at the earliest time slot (in the batch of T time slots) during which both are still available (for pairing); if they have "schedule conflicts" over all T time slots, this proposal is rejected. If output port 1 receives proposals from multiple input ports, then it first sorts (with ties broken arbitrarily) these proposals in a descending order according to their corresponding VOQ lengths, and then tries to accept each of them using the FFA strategy.

In SB-QPS, opportunities – in the form of proposals from input ports – can arise, throughout the time window (up to T time slots long) for computing the join calendar, to fill any of its TN cells. As explained earlier, this "capturing every opportunity" to fill the joint calendar allows a batch switching algorithm to produce matchings of much higher qualities than a regular switching algorithm that is based on the same underlying bipartite matching algorithm can. Indeed, SB-QPS significantly outperforms QPS-1, the regular switching algorithm that is based on the same QPS bipartite matching primitive, as we will show in §6.

Time Complexity. The time complexity for the accepting phase at an output port is $O(1)$ even in the worst case, because like in [9], we can let the output port drop ("knock out") all proposals except the earliest few (say 3) to arrive. In this work, we indeed set this threshold to 3 and find that it has a negligible effect on the quality of resulting matchings.

We now explain how to carry out an FFA operation in $O(1)$ time. In SB-QPS, we encode the availability information of an input port i as a T-bit-long bitmap, one bit for one time slot. Since the batch size T in SB-QPS is a small constant (say T=16), the availability bitmap can fit into a single CPU word. FFA consists of only two instructions: a bitwise-AND (suppose a 1 bit means being available), and a "finding the first 1" instruction that is supported by most modern CPUs. Each instruction only takes $O(1)$ time.

To summarize, the worst-case time complexity of SB-QPS is $O(T)$ per input or output port for the joint calendar consisting of T matchings ($O(1)$ for each matching), since SB-QPS runs T iterations and each iteration has $O(1)$ worst-case time complexity per input or output port.

Message Complexity. The message complexity of each "propose-accept" iteration is $O(1)$ messages per input or output port, because each input port sends at most one proposing message per iteration and each output port sends out at most 3 acceptance messages (where 3 is the "knock-out" threshold explained above). Each proposing message is $T+\lceil \log_2 W \rceil$ bits long (T bits for encoding the availability information and $\lceil \log_2 W \rceil$ bits for encoding the corresponding

VOQ length), where W is the longest possible VOQ length. Each acceptance message is $\lceil \log_2 T \rceil$ bits long (for encoding the time slot the pairing is to be made).

4. SLIDING-WINDOW QPS

In this section, we present in detail the Sliding-Window QPS (SW-QPS) algorithm, the final and only research product of this work. Before we do so, we describe next the sliding-window framework that SW-QPS builds on.

4.1 Sliding-Window Switching

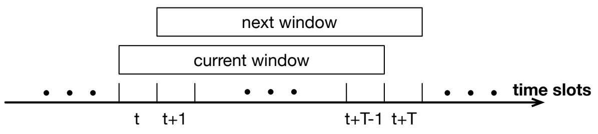

Figure 2: Sliding-window switching.

As mentioned earlier, the only difference between SW-QPS and SB-QPS is that SW-QPS changes the batch switching operation of SB-QPS to a sliding-window switching operation. Sliding-window switching combines regular switching with batch switching and gets the better of both worlds, as follows. On one hand, during each time slot, under a sliding-window switching operation, there are T matchings under computation, just like under a batch switching operation. Each such matching has had or will have a window of T time slots to find opportunities to have its quality improved by the underlying bipartite matching algorithm before it "graduates". Hence, each such matching, when it "graduates", can have a similar or even better quality than that computed by the batch switching algorithm that is based on the same underlying bipartite matching algorithm, as will be confirmed in §6.

On the other hand, under a sliding-window switching operation, the "windows of opportunities" of these T matchings are staggered so that one matching ("class") is output ("graduated") every time slot. This matching is to be used as the crossbar configuration for the current time slot. In this respect, it behaves like a regular switching algorithm and hence completely eliminates the batching delay of the batch switching. More specifically, at the beginning of time slot t, the most senior matching ("class") in the window was added ("enrolled") to the window at the end of time slot $t - T - 1$ and is to "graduate" at the beginning of time slot t, so its "window of opportunity" (to have its quality improved) is $[t - T, t - 1]$. The "window of opportunity" for the second most senior matching is $[t - T + 1, t]$ and so on. At the end of time slot t, a "freshman class" (an empty matching) is "enrolled" and scheduled to "graduate" at time slot $t + T + 1$ in the future.

Figure 2 shows how the sliding window evolves from time slot t to time slot $t+1$. In Figure 2, each interval along the timeline corresponds to a "class". As shown in Figure 2, at the beginning of time slot t, the current window contains "classes of years" (matchings-under-computation to be used as crossbar schedules for time slots) $t, t+1, \cdots,$ and $t+T-1$. Then, at the beginning of time slot $t+1$, the current window slides right by 1 (time slot), and the new window contains "classes of years" $t + 1$, $t + 2$, $\cdots$, and $t + T$, because the "class of year t" just graduated and the "class of year $t+T$" was just "enrolled".

In theory, almost any batch switching algorithm can be converted into a sliding-window switching algorithm by making the "windows of opportunity" for the batch of T matchings-under-computation staggered instead of aligned. This conversion would in general improve switching performance by eliminating the batching delay. Hence, this sliding-window switching framework is itself a separate contribution of this work.

4.2 The SW-QPS Algorithm

SW-QPS is exactly such a conversion of the batch switching algorithm SB-QPS into a sliding-window switching algorithm. SW-QPS is also a parallel iterative algorithm whose each iteration is identical to that of SB-QPS. Hence SW-QPS has the same $O(1)$ time and $O(1)$ message complexities (per port per matching computation) as SB-QPS. The only major difference is that, SW-QPS "graduates" a matching every time slot whereas SB-QPS "batch-graduates" T matchings every T time slots. This "graduating a class each year" allows SW-QPS to completely eliminate the batching delay. As explained earlier, in SW-QPS, at the beginning of time slot t, the joint calendar consists of the T matchings-under-computation that are to "graduate" in "years" (time slots) t, $t + 1$, $\cdots$, $t + T - 1$ respectively. Hence at time slot t, the T-bit-long availability bitmap of an input port i indicates the availabilities of i during $[t, t + T - 1]$.

Note that SW-QPS inherits the FFA (First Fit Accepting) strategy of SB-QPS that is to arrange for an input-output pairing – and hence the switching of a packet between the pair – at the earliest mutually available time slot. In other words, an incoming packet is always "advanced to the most senior class that it can fit in schedule-wise" so that it can "graduate" at the earliest "year" possible. This greedy strategy further reduces the queueing delay of a packet, as will be shown in §6.

5. RELATED WORK

In this section, we provide a brief survey of prior studies that are directly related to ours.

Regular Switching Algorithms. Using MWM (Maximum Weighted Matching) as crossbar schedules is known to result in 100% switch throughput and near-optimal queueing delays under various traffic patterns [14], but each MWM takes $O(N^{2.5} \log W)$ time to compute using the state-of-the-art algorithm [4], where W is the maximum possible length of a VOQ. Motivated by this, various parallel exact or approximate MWM algorithms (*e.g.*, [2,5]) have been proposed to reduce its time complexity. However, the time complexities of all these algorithms above are still too high to be used in high-line-rate high-radix switches.

The family of parallel iterative algorithms [10–13] generally has a low time complexity per port. However, their throughput and delay performances are generally much worse than those of MWM. We note that QPS-r [10], the state-of-the-art algorithm in this family, also builds on QPS [9]. It simply runs r (a small constant) iterations of QPS to arrive at a final matching. We will compare our SB-QPS and SW-QPS with it in §6.

Batch Switching Algorithms. Most of the existing batch switching algorithms [1, 16, 18] model the process of packing the joint calendar as an edge-coloring problem, but until now, most practical solutions to the latter problem are centralized and have high complexity. For example, the Fair-Frame algorithm [16] based on the Birkhoff von Neumann Decomposition (BvND) has a time complexity of $O(N^{1.5}\log N)$ per matching computation.

A recent work, based on parallel edge coloring, has been proposed in [18]. It pushes the per-port time complexity (per matching computation) down to $O(\log N)$. It requires a bath size of only $O(\log N)$, but as mentioned in §1, the constant factor hidden in the big-O is very large.

6. PERFORMANCE EVALUATION

In this section, we evaluate, through simulations, the throughput and delay performances of SB-QPS and SW-QPS under various load conditions and traffic patterns. Our algorithms are compared against iSLIP [13], which runs $\log_2 N$ request-grant-accept iterations and is hence much more expensive computationally. Our algorithms are also compared against QPS-1 (QPS-r with r=1 iteration) [10]. This is a fair comparison because QPS-1, like our algorithms, runs only a single iteration to compute a matching. The MWM algorithm, which delivers near-optimal delay performance [17], is also compared against as a benchmark.

6.1 Simulation Setup

In our simulations, we fix the number of input and output ports N to 64; we however will investigate in Appendix A.1 in [15], how the mean delay performances of these algorithms scale with respect to N. To accurately measure throughput and delay, we assume that each VOQ has an infinite buffer size, so no packet is dropped at any input port. Each simulation run follows the stopping rule in [6,8]: The number of time slots simulated is at least $500N^2$ and guarantees the difference between the estimated and the actual average delays to be within 0.01 time slots with at least 0.98 probability.

We assume in our simulations that each traffic arrival matrix $A(t)$ is *i.i.d.* Bernoulli with its traffic rate matrix equal to the product of the offered load and a traffic pattern matrix (defined next). Similar Bernoulli arrivals were studied in [7,9,13]. Later, in Appendix A.2 in [15], we will look at burst traffic arrivals. Note that only synthetic traffic (instead of that derived from packet traces) is used in our simulations because, to the best of our knowledge, there is no meaningful way to combine packet traces into switch-wide traffic workloads. The following four standard types of normalized (with each row or column sum equal to 1) traffic patterns are used: (I) *Uniform*: packets arriving at any input port go to each output port with probability $\frac{1}{N}$. (II) *Quasi-diagonal*: packets arriving at input port i go to output port $j=i$ with probability $\frac{1}{2}$ and go to any other output port with probability $\frac{1}{2(N-1)}$. (III) *Log-diagonal*: packets arriving at input port i go to output port $j=i$ with probability $\frac{2^{(N-1)}}{2^N-1}$ and go to any other output port j with probability equal $\frac{1}{2}$ of the probability of output port $j-1$ (note: output port 0 equals output port N). (IV) *Diagonal*: packets arriving at input port i go to output port $j=i$ with probability $\frac{2}{3}$, or go to output port $(i \bmod N)+1$ with probability $\frac{1}{3}$. These traffic patterns are listed in order of how skewed the volumes of traffic arrivals to different output ports are: from uniform being the least skewed, to diagonal being the most skewed.

When implementing SB-QPS and SW-QPS, we have to first decide on the value of batch (for SB-QPS) or window (for SW-QPS) size T. As explained earlier in §1, a larger T generally results in matchings of higher qualities and hence leads to better throughput performances. However, a larger T results in higher message complexities and longer batching delays (for SB-QPS only). Through simulations (results not shown here in the interest of space), we have found that $T = 16$ strikes a nice performance-cost tradeoff for SW-QPS: The proposal message size is small when $T = 16$, yet the throughput gains when increasing T beyond 16 (say to 32) are marginal. For SB-QPS, we also adopt $T = 16$ for consistency, which leads to a reasonably low batching delay and shows that SB-QPS clearly deserves its name (small-batch) since this tiny batch size of 16 is much smaller than that of any other batch switching algorithm.

For SW-QPS, T does not have to grow with N (to deliver similar throughput and delay performances), as we will show in Appendix A.1 in [15] that the delay performance of SW-QPS (with $T = 16$) does not degrade when N grows larger.

6.2 Throughput Performance Results

Table 1: Maximum achievable throughput.

Traffic	Uniform	Quasi-diag	Log-diag	Diag
SB-QPS	86.88%	87.10%	87.31%	86.47%
SW-QPS	92.56%	91.71%	91.40%	87.74%
iSLIP	99.56%	80.43%	83.16%	82.96%
QPS-1	63.54%	66.60%	68.78%	75.16%

Table 1 presents the maximum achievable throughput of SB-QPS, SW-QPS, iSLIP, and QPS-1, under the aforementioned four standard traffic patterns and an offered load close to 1 (more precisely, 0.9999). We do not include the throughout of MWM in Table 1, because it can provably attain 100% throughput. We make three observations from Table 1. First, SW-QPS significantly improves the throughput performance of QPS-1, increasing it by an additive term of 0.2902, 0.2511, 0.2262, and 0.1258 for the uniform, quasi-diagonal, log-diagonal, and diagonal traffic patterns respectively. Second, the throughput of SW-QPS is consistently higher than that of SB-QPS under the four traffic patterns. Third, under all traffic patterns except uniform, SW-QPS significantly outperforms iSLIP, which is much more expensive computationally as it runs $\log_2 N$ iterations for each matching computation.

6.3 Delay Performance Results

Figure 3 shows the mean delays of SB-QPS, SW-QPS, iSLIP, QPS-1, and MWM under the aforementioned four traffic patterns. As we have shown in §6.2, SB-QPS, SW-QPS, iSLIP and QPS-1 generally cannot attain 100% throughput, so we only measure their delay performances for the offered loads under which they are stable; in all figures in the sequel, each "missing point" on a plot indicates that the corresponding algorithm is not stable under the corresponding traffic pattern and offered load. Figure 3 shows that, when the offered load is not very high (say < 0.6), SB-QPS has a much higher mean overall delay than others thanks to its batching delay that is still relatively quite high (despite a small batch size of $T = 16$); in comparison, SW-QPS completely eliminates this batching delay. Figure 3 also shows that SW-QPS outperforms QPS-1 everywhere and outperforms

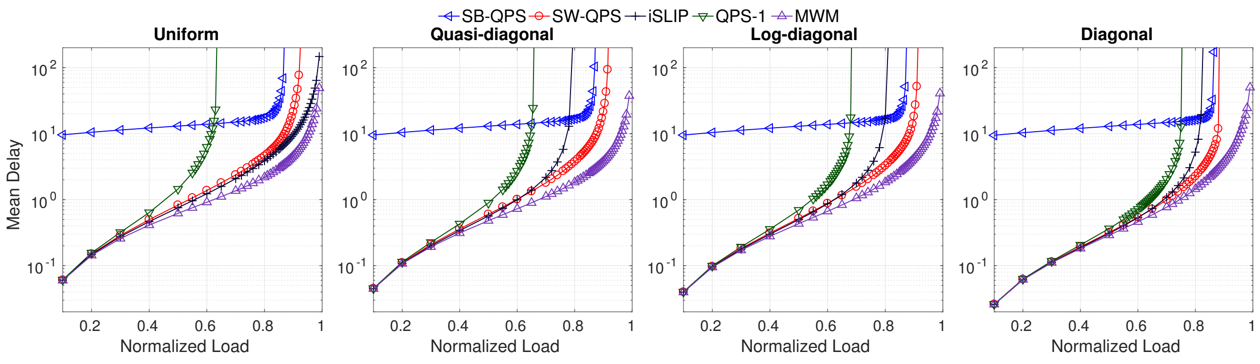

Figure 3: Mean delays of SB-QPS, SW-QPS, iSLIP, QPS-1, and MWM under the 4 traffic patterns.

iSLIP under all traffic patterns except uniform. Since as shown in Table 1 and Figure 3, the throughput and the delay performances of SW-QPS are strictly better than those of SB-QPS, we will show the performance results of only SW-QPS in Appendix A in [15], in which we present more evaluation results.

7. CONCLUSION

In this work, we first propose a batch switching algorithm called SB-QPS that significantly reduces the batch size without sacrificing the throughput performance much, and achieves a time complexity of $O(1)$ per matching computation per port via parallelization. We then propose a regular switching algorithm called SW-QPS that improves on SB-QPS using a novel sliding-window switching framework. SW-QPS inherits and enhances all benefits of SB-QPS and reduces the batching delay to zero. We show, through simulations, that the throughput and delay performances of SW-QPS are much better than those of QPS-1, the state-of-the-art regular switching algorithm based on the same underlying bipartite matching algorithm.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1909048 and CNS-2007006.

8. REFERENCES

[1] G. Aggarwal, R. Motwani, D. Shah, and A. Zhu. Switch scheduling via randomized edge coloring. In *Proc. of the IEEE FOCS*, pages 502–512, Oct 2003.

[2] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma. Iterative scheduling algorithms. In *Proc. of the IEEE INFOCOM*, pages 445–453, May 2007.

[3] C. Cakir, R. Ho, J. Lexau, and K. Mai. Scalable high-radix modular crossbar switches. In *Proc. of the HOTI*, pages 37–44, Aug 2016.

[4] R. Duan and H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the ACM-SIAM SODA*, pages 1413–1424, 2012.

[5] M. Fayyazi, D. Kaeli, and W. Meleis. Parallel maximum weight bipartite matching algorithms for scheduling in input-queued switches. In *Proc. of the IEEE IPDPS*, pages 4–11, Apr. 2004.

[6] J. Flegal, G. Jones, et al. Batch means and spectral variance estimators in markov chain monte carlo. *Ann. Stat.*, 38(2):1034–1070, 2010.

[7] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE J. Sel. Areas Commun.*, 21(4):546–559, 2003.

[8] P. Glynn, W. Whitt, et al. The asymptotic validity of sequential stopping rules for stochastic simulations. *Ann. Appl. Probab.*, 2(1):180–198, 1992.

[9] L. Gong, P. Tune, L. Liu, S. Yang, and J. Xu. Queue-proportional sampling: A better approach to crossbar scheduling for input-queued switches. *Proc. of the ACM SIGMETRICS*, 1(1):3:1–3:33, June 2017.

[10] L. Gong, J. Xu, L. Liu, and S. T. Maguluri. QPS-r: A cost-effective crossbar scheduling algorithm and its stability and delay analysis. In *Proc. of the EAI VALUETOOLS*, 2020.

[11] B. Hu, F. Fan, K. L. Yeung, and S. Jamin. Highest rank first: A new class of single-iteration scheduling algorithms for input-queued switches. *IEEE Access*, 6:11046–11062, 2018.

[12] B. Hu, K. L. Yeung, Q. Zhou, and C. He. On iterative scheduling for input-queued switches with a speedup of $2 - 1/n$. *IEEE/ACM Trans. Netw.*, 24(6):3565–3577, December 2016.

[13] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2):188–201, Apr. 1999.

[14] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Commun.*, 47(8):1260–1267, Aug. 1999.

[15] J. Meng, L. Gong, and J. Xu. Sliding-window QPS (SW-QPS): A perfect parallel iterative switching algorithm for input-queued switches, e-prints arXiv:2010.08620, Oct. 2020.

[16] M. J. Neely, E. Modiano, and Y. S. Cheng. Logarithmic delay for n × n packet switches under the crossbar constraint. *IEEE/ACM Trans. Netw.*, 15(3):657–668, June 2007.

[17] D. Shah and D. Wischik. Optimal scheduling algorithms for input-queued switches. In *Proc. of the IEEE INFOCOM*, pages 1–11, Apr. 2006.

[18] L. Wang, T. Ye, T. Lee, and W. Hu. A parallel complex coloring algorithm for scheduling of input-queued switches. *IEEE Trans. Parallel Distrib. Syst.*, 29(7):1456–1468, 2018.