Queue-Proportional Sampling: A Better Approach to Crossbar Scheduling for Input-Queued Switches

Long Gong[†] Paul Tune[‡] Liang Liu[†] Sen Yang[†] Jun (Jim) Xu[†]
Georgia Institute of Technology[†] School of Mathematical Sciences, University of Adelaide[‡]
{gonglong,lliu315,sen.yang}@gatech.edu paul.tune@adelaide.edu.au jx@cc.gatech.edu

ABSTRACT

Most present day switching systems, in Internet routers and data-center switches, employ a single input-queued crossbar to interconnect input ports with output ports. Such switches need to compute a matching, between input and output ports, for each switching cycle (time slot). The main challenge in designing such matching algorithms is to deal with the unfortunate tradeoff between the quality of the computed matching and the computational complexity of the algorithm. In this paper, we propose a general approach that can significantly boost the performance of both Serena and iSLIP, yet incurs only $O(1)$ additional computational complexity at each input/output port. Our approach is a novel proposing strategy, called *Queue-Proportional Sampling (QPS)*, that generates an excellent *starter matching.* We show, through rigorous simulations, that when starting with this starter matching, iSLIP and Serena can output much better final matching decisions, as measured by the resulting throughput and delay performance, than they otherwise can.

Keywords

Crossbar scheduling; input-queued switch; matching; queue-proportional sampling

1. INTRODUCTION

Most present day switching systems, in Internet routers and data-center switches, employ a single crossbar to interconnect input ports with output ports. A generic input-queued switch is shown in Figure 1, with N input and N output ports interconnected by a crossbar. Each input port has N Virtual Output Queues (VOQs). A VOQ j at input port i serves as a buffer for packets going from input port i to output port j. The use of VOQs solves the Head-of-Line (HOL) blocking issue [13], which severely limits the throughput of the switch system.

In an input-queued switch, each input port can be connected to only one output port, and vice versa, in each switching cycle, or time slot. Hence, input-queued switches

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

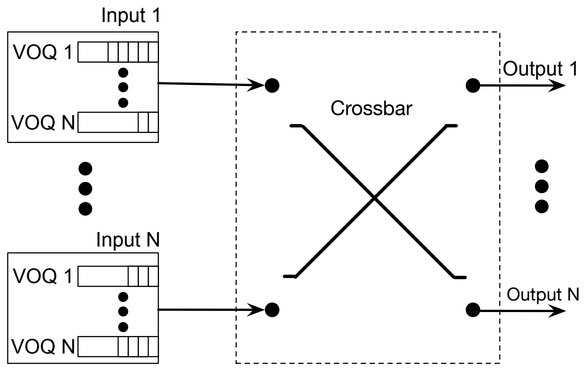


Figure 1: *Generic input-queued crossbar switch.*

need to compute, per time slot, a one-to-one *matching* between input and output ports. With the relentless growth in the volume of network traffic across the Internet and in data-centers, switches capable of connecting a large number of ports and operating at very high port/link speeds are badly needed. The primary research challenge when designing such large single-crossbar switch architectures is to develop algorithms that can compute "high quality" matchings – *i.e.,* those that result in high switch throughput (ideally 100%) and low queueing delays for packets – at high speeds.

Unfortunately, there appears to be a tradeoff between the quality of a matching and the time needed to compute it (*i.e.,* computational complexity). Maximum Weight Matching (MWM), with a suitable weight measure, is known to produce (empirically) optimal matchings in terms of queueing delay for a large variety of traffic patterns [34]. Each matching decision however takes $O(N^3)$ time to compute [7]. Researchers have been searching for alternatives that have complexity much lower than $O(N^3)$, but have performance (mostly in terms of delay) close enough to MWM.

Serena [10] is one such algorithm. It outputs excellent matching decisions resulting in 100% switch throughput and queueing delay close to that of MWM. Each matching decision takes $O(N)$ time to compute. Another example is iSLIP [17], a distributed iterative algorithm where input and output ports compute a matching in parallel through multiple iterations of message exchanges. iSLIP has a per-port computational complexity of $O(\log^2 N)$ ($O(\log N)$ iterations that each has $O(\log N)$ circuit depth) that is lower than Serena's overall complexity of $O(N)$. However, iSLIP computes a different type of matching called a Maximum-Size Matching (MSM), which is of lower quality than MWM. Hence iS-

LIP cannot achieve 100% throughput except under uniform traffic, and has much longer queueing delays than Serena under heavy nonuniform traffic.

1.1 Starter Matching and Its Importance

In Serena, a starter (partial) matching is first generated, via a *proposing process*, then populated into a full matching, and finally refined into the final matching. The proposing process works as follows. Each input port "proposes" to an output port that it would like to match with by sending the output port a message containing the length of the corresponding VOQ. An output port, upon receiving proposals from one or more input ports, accepts the one whose corresponding VOQ is the longest.

Serena's proposing strategy is the so-called *arrival graph*: each input port proposes to an output port corresponding to the destination of a packet that arrived in the previous time slot, if applicable. This is a sensible strategy because, in the steady state, an output port is proposed to with a probability proportional to the packet arrival rate of the corresponding VOQ.

However, this proposing strategy has a subtle shortcoming: it is oblivious to the current lengths of N VOQs at each input port, so not enough attention is devoted to reducing the lengths of longest VOQs. For example, a VOQ with many packets but without recent arrivals, which could happen under bursty traffic (see §6), will mostly be denied service until it has new arrivals.

1.2 Queue-Proportional Sampling (QPS)

In this paper, we propose a general approach that can significantly boost the performance of both Serena and iSLIP, yet incurs only $O(1)$ additional computational complexity at each input/output port. Our approach is a novel proposing strategy, called *Queue-Proportional Sampling (QPS)*, that generates an excellent *starter matching*, better than the arrival graph used by Serena. Scheduling algorithms that start from "scratch" (*i.e.,* an empty matching), such as iSLIP, may also benefit significantly from QPS, by instead starting from a QPS-generated starter matching.

Our proposing strategy, QPS, at any input port, is extremely simple to state: the input port proposes to an output port with a probability proportional to the length of the corresponding VOQ. QPS's name comes from the fact that the output port proposed to by any input port is sampled, out of all N output ports, using the queue-proportional distribution at the input port. We note that, although this general approach – of serving queues at rates/probabilities proportional to their lengths – to resource allocation is classical [8], QPS is a novel application of this approach to crossbar scheduling.

We will show in §4 that QPS is also extremely cheap to execute: we developed an $O(1)$ data structure and algorithm for generating such a sample at each input port. This may be surprising to readers, since even to "read" the lengths of all N VOQs at an input port takes $O(N)$ time. Due to its $O(1)$ (per port) computational complexity, any QPS-augmented algorithm has the same asymptotic complexity as the original algorithm.

In this work, we consider two QPS-augmented algorithms: *QPS-iSLIP* and *QPS-Serena*, which combine QPS with iSLIP [17] and Serena [10] respectively. Both QPS-augmented algorithms are shown to outperform the original algorithms, in both throughput and delay, under various load conditions and traffic patterns, by a wide margin in §6. As the QPS approach is very general, it can be used to augment other low-complexity switching algorithms in the future.

We make the following three major contributions in this work. First, we propose QPS, a simple yet effective approach to crossbar scheduling, and use it to augment both iSLIP and Serena. Second, we propose a data structure that carries out each QPS operation with only $O(1)$ computation per port. Third, for proving the stability of QPS-Serena, we derive a new and stronger theorem for proving the stability of a large family of switching algorithms.

The rest of this paper is organized as follows. In §2, we provide some background on input-queued crossbar scheduling. In §3, we describe the QPS proposing strategy and two QPS-augmented crossbar scheduling algorithms, namely QPS-iSLIP and QPS-Serena. In §4, we show how to carry out each QPS operation with only $O(1)$ computation. In §5, we prove that QPS-Serena can achieve 100% throughput. In §6, we evaluate the throughput and delay performance of QPS-iSLIP and QPS-Serena against other competing algorithms. In §7, we describe related work before concluding the paper in §8.

2. BACKGROUND

In this section, we provide an overview of the input-queued crossbar switch architecture and formulate the research problem of crossbar scheduling.

2.1 Input-Queued Crossbar Architecture

In an input-queued switch, packets arriving at an input port are queued first in their respective VOQs before being switched to their respective output ports by the crossbar. In this work, we adopt the standard assumption that all incoming variable-size packets are segmented into fixed-size packets (sometimes referred to as cells), which are then reassembled when leaving the switch. Hence we consider the switching of only fixed-size packets in the sequel, and each such fixed-size packet takes exactly one time slot to transmit. We also make the following standard homogeneity assumption that every input or output link/port has the same maximum transmission rate (normalized to 1), which is equal to that of a transmission line or crosspoint in the crossbar (also normalized to 1).

An $N \times N$ crossbar is generally modeled as a weighted complete bipartite graph, with the N input ports and the N output ports represented as the two disjoint vertex sets respectively. An edge between an input port i and an output port j corresponds to the VOQ j at the input port i, and its weight is the queue length (*i.e.,* the number of packets buffered) of the VOQ. A valid schedule, or *matching*, is a set of edges between the N input ports and the N output ports, in which no two distinct edges share a vertex. Since there can be at most N edges in any such matching, the crossbar can switch at most N packets to their respectively output ports during each time slot. Each matching can also be represented as an $N \times N$ sub-permutation matrix[1] $S = (s_{ij})$, in which $s_{ij} = 1$ if and only if the input port i is matched with the output port j.

[1]An $N \times N$ sub-permutation matrix is an $N \times N$ 0-1 matrix where at most one element in each row or column can take value 1.

2.2 Performance Metrics

The research objective of crossbar scheduling is to design scheduling algorithms that select a good matching, as measured by certain performance metrics, in each time slot, with a reasonable amount of computation. Typically, scheduling algorithms are evaluated on three performance metrics: *throughput*, *delay*, and *complexity*.

Throughput: Normalized throughput is defined as the average number of packets that exit an output port during each time slot. It is a value between 0 and 1 (*i.e.,* 100%). Throughout this work, we mean normalized throughput whenever we use the word "throughput".

We say a switch, employing a certain crossbar scheduling algorithm, is stable [19] – under a certain workload – if its total queue (VOQ) length $\|Q(t)\|_1$ satisfies $\sup_{0\le t<\infty} \mathbb{E}\big[\|Q(t)\|_1\big] < \infty$. A crossbar scheduling algorithm is said to achieve 100% *throughput*, if the switch is stable under any traffic arrival process that is admissible (defined next) and satisfies certain other mild conditions (see §5.1). For example, Serena can achieve 100% throughput under any such admissible arrival process, whereas iSLIP generally cannot.

Delay: We define delay as the number of time slots elapsed since the arrival of a packet to its eventual departure from the switch. An ideal scheduling algorithm has 100% throughput and low delay. Achieving 100% throughput is relatively easier than achieving low delay. For instance, in TASS [33], 100% throughput is achieved, at the cost of high delays, using a simple randomized adaptive algorithm that we will describe in §5.2.

Complexity: Another criterion for evaluating a scheduling algorithm is the time complexity of computating a matching. As mentioned earlier, folklore suggests a tradeoff between the quality of matching and the computational complexity. A key contribution of our QPS approach is to strike better performance-complexity tradeoffs than existing approaches such as iSLIP and Serena.

2.3 Admissible Traffic Patterns

Let λ_{ij} be the normalized (to the percentage of the rate of an input/output link) mean arrival rate of packets to the j^{th} VOQ (*i.e.,* those destined for output port j) at input port i. Then the traffic pattern, represented by an $N \times N$ traffic matrix $\Lambda = \{\lambda_{ij}\}_{N\times N}$, is called *admissible* if

$$\lambda_i^{(in)} \triangleq \sum_j \lambda_{ij} < 1 \quad \forall 1 \le i \le N \tag{1}$$

$$\lambda_j^{(out)} \triangleq \sum_i \lambda_{ij} < 1 \quad \forall 1 \le j \le N \tag{2}$$

Equivalently, we say Λ is admissible, if and only if $\rho < 1$, where ρ, defined as

$$\rho \triangleq \max\big\{\max_{1\le i\le N}\{\lambda_i^{(in)}\}, \max_{1\le j\le N}\{\lambda_j^{(out)}\}\big\} \tag{3}$$

is the maximum normalized load imposed on any input or output port. Clearly, $\rho < 1$ is a necessary condition for any crossbar scheduling algorithm to ensure the stability of a switch.

Now we state a well-known fact that has been used, usually without a proof, in almost every switch stability proof in the literature.

FACT 1. *For each $N \times N$ admissible traffic matrix Λ, whose maximum per input/output load is ρ (defined in (3)), there exist $N \times N$ matching (sub-permutation) matrices M_n, $n = 1, 2, \ldots, K$ such that*

$$\Lambda = \sum_{n=1}^{K} \alpha_n M_n \tag{4}$$

where $K \le N^2 - 2N + 2$, $\alpha_n > 0$ and $\sum_{n=1}^{K} \alpha_n \le \rho$.

This fact follows from the fact that Λ/ρ is a sub-stochastic matrix, which can be expressed as a linear combination of sub-permutation matrices with positive coefficients summing up to a value no larger than 1, known as the Birkhoff–von Neumann decomposition [6, 21, 23].

3. QUEUE-PROPORTIONAL SAMPLING

In this section, we first describe the QPS proposing strategy in details. Then we explain how to augment iSLIP and Serena using QPS. We next compare QPS with ShakeUp [11], another "add-on" technique that can be used to augment iterative switching algorithms such as iSLIP and iLQF [16]. In Appendix A, we discuss a QPS variant called FQPS, which samples a VOQ with a probability proportional to a function of the VOQ length.

3.1 The QPS Proposing Strategy

In all QPS-augmented crossbar scheduling algorithms, the first step is for input ports and output ports to perform one iteration of message exchanges to generate a starter matching. This iteration consists of two phases, namely, a QPS-proposing phase and an accepting phase.

Proposing phase. In this phase, each input port proposes to exactly one output port – decided by the QPS strategy – unless it has no packet to transmit. Procedure 1 shows the pseudocode of the QPS proposing strategy at an input port 1; that at any other input port is identical. Denote as m_1, m_2, $\cdots$, m_N the respective lengths of N VOQs at input port 1, and as m their total (i.e., $m \triangleq \sum_{k=1}^{N} m_k$). Input port 1 simply samples an output port j with probability $\frac{m_j}{m}$ (line 2), *i.e.,* proportional to the length of the corresponding VOQ; it then proposes the value m_j to output port j (line 3).

Accepting phase. We adopt the same accepting strategy as in Serena: "Longest VOQ first". The pseudocode of the accepting phase, at output port 1, is shown in Procedure 2; that at any other output port is identical. The action of output port 1 depends on the number of proposals it receives. If it receives exactly one proposal from an input port, it will accept the proposal and (tentatively) match with the input port. However, if it receives proposals from multiple input ports, it will accept the proposal accompanied with the highest VOQ length, with ties broken uniformly at random.

The computational complexity of this accepting strategy is $O(1)$ in practice although in theory an output port could receive up to N proposals and have to compare their accompanying VOQ lengths. This is because the probability for an output port to receive proposals from more than several (say 5) input ports is tiny, and even if this rare event happens, the output port can ignore/drop all proposals beyond the first several (say 5) without affecting the quality of the final matching much. In our evaluations, we indeed set this threshold to 5.

```
1 Procedure QPS-Propose()
2     Sample an output port $j$ with probability $\frac{m_j}{m}$
3     Send $m_j$ (length of VOQ $j$) to output port $j$
```

Procedure 1: Proposing phase at input port 1.

```
1 Procedure Accept()
2     if one or more proposals are received then
3         Accept the one with largest VOQ length
```

Procedure 2: Accepting phase at output port 1.

We have also considered and experimented with another accepting strategy: accepting each competing proposal with a probability proportional to the length of the corresponding VOQ, which we refer to as *Proportional Accepting* (PA). The advantage of PA over "longest VOQ first" above is that when the switch is severely overloaded (*i.e.,* with offered load $>$ 100%), PA could provide better fairness to competing input ports and help prevent certain starvation situations. For example, consider the pathological scenario in which, for a fairly long period of time (say 1 minute), packets destined for an output j would arrive at input ports i_1 and i_2 with rates 1 and 0.1 respectively. Under "longest VOQ first", the output port j would keep accepting proposals from input port i_1 (because its VOQ length is longer) and hence starve input port i_2, whereas under PA, the output port j would accept proposals from input port i_2 with roughly 1/11 probability.

However, we prefer "longest VOQ first" over PA because, as we will show in Appendix E.3, the former generally has better average delay performance, albeit slightly, and guarantees almost the same fairness and lack of starvation, under all *admissible* workloads. We believe the primary mission of a crossbar scheduling algorithm is to deliver excellent performance under *admissible* workloads; such "grace under fire" (proportional fairness and lack of starvation even when severely overloaded) is a secondary consideration and can be better achieved through other "knobs or levers" orthogonal to switching such as congestion control, packet scheduling, or traffic policing/shaping. This said, we prove in Appendix D that QPS-Serena with PA can also achieve 100% throughput just like QPS-Serena with "longest VOQ first", in case the former is preferred in certain application scenarios.

Message Complexity. The message complexity of each "propose-accept" iteration is $O(1)$ messages per input or output port, because each input/output port transmits no more than one message during the propose/accept phase.

3.2 Augmenting iSLIP and Serena

Now we describe, in QPS-iSLIP and QPS-Serena respectively, how iSLIP and Serena are augmented using QPS. We also describe iLQF [16] in this section, because it is closely related to iSLIP, and its performance will be compared against QPS-iSLIP in §6.

3.2.1 iSLIP, QPS-iSLIP, and iLQF

The iSLIP algorithm computes an approximate MSM (Maximum Size Matching) via multiple iterations of message exchanges between the input and output ports. Each iteration consists of three stages: request, grant, and accept. In the request stage, each input port sends requests to all output ports whose corresponding VOQs are not empty. In the grant stage, each output port, upon receiving requests from multiple input ports, grants to one in a round-robin order. This round-robin order is enforced through a *grant pointer* that records the identifier of the input port – to whom a grant was accepted in the first iteration – during the most recent time slot when this situation occurred. Finally, in the accept stage, each input port, upon receiving accepts from multiple output ports, accepts one in a round-robin order, enforced similarly through an *accept pointer*.

QPS-iSLIP can be viewed as adding a "0^{th} iteration" to iSLIP. In this 0^{th} iteration, QPS is executed to generate a starter matching. Then iSLIP is called to match only those input/output ports not matched in the 0^{th} iteration, through multiple request-grant-accept iterations. We specify that in QPS-iSLIP, it is those ports matched in the 1^{st} iteration (by iSLIP), not those matched in the 0^{th} iteration (by QPS), who update the values of their grant or accept pointers. The rationale is that the aforementioned objective of enforcing the round-robin order is not accomplished in the QPS iteration.

iLQF [16] operates in the same way as iSLIP, except that (1) it is aware of the edge weights (*i.e.,* lengths of VOQs), and (2) it favors the request or grants with the heaviest weight (*i.e.,* greedy) in the grant or accept stage respectively. Hence, iLQF can be viewed as a greedy approach to approximately compute the MWM. iLQF generally performs better than iSLIP, but has a higher computational complexity of $O(N)$ per port (compared to $O(\log^2 N)$ for iSLIP). We show in §6 that our QPS-iSLIP algorithm has a similar performance as iLQF, but the same per-port complexity as iSLIP.

3.2.2 Serena and QPS-Serena

As described earlier, Serena derives a starter matching from the arrival graph. This starter matching, which is typically partial, is then populated into a full matching by pairing the unmatched nodes in the bipartite graph uniformly at random. Serena then combines, using a MERGE procedure, this full matching with the matching used in the previous time slot, to arrive at a new matching that is at least as heavy as both matchings. This new matching will then be used for the current time slot. We omit the details of this MERGE procedure, since it is not related to how QPS augments Serena. Finally, to precisely specify QPS-Serena, it suffices to note that the only difference between QPS-Serena and Serena is that QPS-Serena uses a QPS-generated starter matching, instead of one derived from the arrival graph.

3.3 QPS vs. ShakeUp

As we have shown, QPS is used mainly as an "add-on" to certain switching algorithms. In the literature, the only other add-on technique that we are aware of is ShakeUp [11]. ShakeUp is a set of randomized algorithms designed to boost the performance of certain *iterative* switching algorithms, such as iSLIP and iLQF. It does so by preventing these iterative algorithms from getting stuck at (locally) maximal matchings during their iterative executions. ShakeUp is typically used as follows: a ShakeUp-augmented switching algorithm alternates between an iteration of the underlying switching algorithm (*e.g.,* iSLIP) and a ShakeUp iteration.

There are two types of ShakeUp algorithms: unweighted and weighted [11]. The unweighted ShakeUp is designed to augment switching algorithms that do not consider VOQ lengths in their decision-making, such as Parallel Iterative Matching (PIM) [2] and iSLIP [17]. In each unweighed ShakeUp iteration, unmatched input ports are first permuted in a random order. From this (random) order, each unmatched input port sends a request to an output port *uniformly at random* (*i.e.,* unweighted) chosen from the set of output ports to which the corresponding VOQs are nonempty. An output port, upon receiving such a request, must now pair with this input port, even if it was already paired with another input port. If an output port receives multiple requests during the same ShakeUp iteration, it selects one of them uniformly at random. The iSLIP scheme augmented this way was called SLIP-SHAKE in [11]. In §6, we will compare the its performance (renamed to iSLIP-ShakeUp) with that of QPS-iSLIP.

The weighted ShakeUp [11] is designed to augment switching algorithms that incorporate VOQ lengths in their decision-making, such as iLQF [16]. In each weighed ShakeUp iteration, each unmatched input port, one after another in the above-mentioned randomly order, sends a request to an output port with a probability proportional to the length of the corresponding VOQ.

Admittedly, weighted ShakeUp's proposing strategy sounds very similar to our QPS strategy. However, there are four key differences: how they are used, how widely applicable they are, their intended purpose, and how they are implemented. First, in ShakeUp, only unmatched input ports execute this strategy to "shake up" an existing suboptimal matching, whereas in QPS, all input ports execute the strategy at the very beginning to generate a starter matching for other switching algorithms to build on. In a sense, ShakeUp is designed for "post-processing" whereas QPS is designed for "pre-processing". Second, while our QPS scheme can easily augment a non-iterative algorithm such as Serena, it is not known whether ShakeUp, weighted or unweighted, can do the same. Third, it was never suggested in [11] that this (weighted) strategy might be suitable for "weight-oblivious" switching algorithms such as PIM or iSLIP; only the unweighed ShakeUp was "prescribed" for PIM or iSLIP. Last, unlike in our work, there was no mention of how the queue-proportional proposing strategy could be carried out in $O(1)$ time (per port), and no data structure was proposed for doing so [11].

4. QPS IMPLEMENTATION

In this section, we describe the data structure and algorithm that allows an input port to sample a VOQ in the queue-proportional manner (*i.e.,* line 2 of Procedure 1), and, if needed, to remove the Head-of-Line (HOL) packet of any VOQ (for receiving switching service), both with $O(1)$ (per port) computational complexity. This data structure is extremely simple, although we have so far not been able to find anything sufficiently similar in the literature.

The memory overhead of the QPS data structure is no more than 20 bytes per packet; the detailed "accounting" is shown in Appendix B. Assuming an average packet size of 500 bytes, the amount of memory consumed by the QPS data structure is no more than 4% of what is needed for storing the actual packets. This is a modest space overhead ratio to pay, for the significant improvements in switching performance.

4.1 Overview of the Sampling Algorithm

We first provide a high-level overview of the sampling algorithm. It consists of two steps. In the first step, we sample a packet, out of all packets currently queued at the input port, uniformly at random. Specifically, if there are a total of m packets across all N VOQs at the input port, each packet is sampled with probability $1/m$. With such uniform sampling, the j^{th} VOQ, which has length m_j, will have one of packets sampled with probability m_j/m. This is precisely the QPS behavior called for in line 2 of Procedure 1.

Suppose a packet is thus sampled. A part of the second step is to find out which VOQ this packet belongs to so that the input port can propose to the corresponding output port with its queue length (see line 3 of Procedure 1). However, more effort is still required. Since all switching algorithms serve packets in a VOQ strictly in the FIFO order, if this proposal is successful (*i.e.,* accepted by the output port), and the input and output port pair is eventually a part of the final matching, the HOL packet of this VOQ, which may or may not be the sampled packet, needs to be located and serviced. Hence, the other part of the second step is to locate the HOL packet of this VOQ.

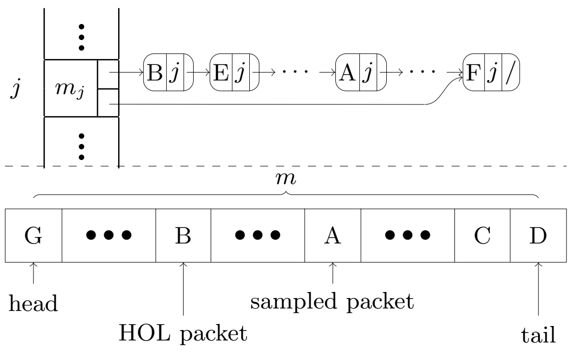
Before going into the details, we list two other basic operations that this data structure needs to also support. The first operation is that any new incoming packet must be recorded in the data structure so that it is logically "added to the end of the VOQ that it belongs to". The second operation is that, when the scheduling algorithm eventually decides to pair the input port with a different output port than was proposed to, which could happen due to either the proposal being rejected or the initially accepted proposal being overridden by the scheduling algorithm (*e.g.,* during Serena's MERGE operation in the case of QPS-Serena), the HOL packet of the (new) corresponding VOQ needs to be located and removed for receiving the switching service. Both operations can be supported with $O(1)$ complexity, as will be shown next.

4.2 The Detailed Data Structure

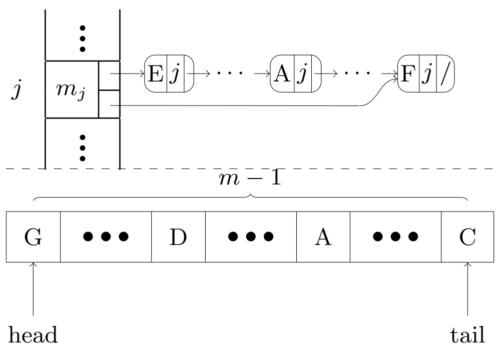
We show that the two steps of the QPS proposing strategy can be performed in $O(1)$ time, at any input port, via a main and an auxiliary data structures, that are the same for all input ports. Figure 2(a) and (b) present the data structures, *at a single input port*, before and after the HOL packet of its j^{th} VOQ is chosen for (switching) service. The top half and bottom half of the figures show the main and the auxiliary data structures respectively.

The main data structure. The main data structure is an array of N records, corresponding to the N VOQs at the input port. Each record j (*i.e.,* array entry j) is associated with a linked list, which corresponds to (pointers to) packets queued at a VOQ in the order they arrived, starting with the HOL packet. Each node in the linked list contains two pointers encoded as "$\langle letter \rangle$" (*e.g.,* A); one points to the actual packet (*e.g.,* packet A) in the packet buffer (not shown in the figure) and the other to the corresponding entry (*e.g.,* entry A) in the auxiliary data structure, which we refer to as a *back pointer*.

For simplicity, Figure 2 shows only record j (corresponding to VOQ j). Each record contains a head and a tail pointers that point to the head node and the tail node of the linked list respectively. The head pointer is needed for locat-



(a) Before scheduling



(b) After scheduling

Figure 2: *Illustrating the action of the QPS data structures on a single input port.*

ing and for removing the head node (*i.e.,* the HOL packet) in $O(1)$ time; it is also needed for locating and replacing the array entry that corresponds to the HOL packet in the auxiliary data structure. The tail pointer is needed for inserting a newly arrived packet to the "end of the VOQ" (*i.e.,* the first basic operation) in $O(1)$ time.

The auxiliary data structure. The bottom half of Figure 2 shows the auxiliary data structure used for performing the sampling. Suppose there are a total of m packets queued across all N VOQs at the input port. The auxiliary data structure is simply an array of m entries, each of which is a pointer that points to a distinct (packet) node (*e.g.,* node A) in one of the N linked lists in the main data structure.

Despite arrivals and departures of packets over time, the auxiliary data structure always occupies a contiguous block of array entries, the boundaries of which are identified by a head and a tail pointer as shown in the bottom half of Figure 2. This contiguity allows any array entry (packet) to be sampled uniformly at random in $O(1)$ time, an aforementioned key step of QPS. Hence this contiguity needs to be maintained in the event of packet arrivals and departures. The case of a packet arrival is easier: the entry corresponds to the new packet is inserted after the current tail position, and the tail pointer updated. The case of a packet departure is only slightly trickier: if the departing packet leaves a "hole" in the block, the tail entry is moved to fill this hole, and the tail pointer updated.

In the case of a packet departure, the (packet) node in the main data structure that is pointed to by the former tail entry (now moved to "fill the hole") needs to have its *back pointer* updated to the offset of the former hole, where the former tail entry now is. This is clearly an $O(1)$ procedure. A similar procedure can be used to support the second basic operation in $O(1)$ time.

An illustrative example. To see how the main and the auxiliary data structures work together to facilitate QPS, consider the example shown in Figure 2. In Figure 2(a), the packet A was sampled out of m packets in the auxiliary data structure. However, it is not the HOL packet, so its destination (output) port (*i.e.,* VOQ identifier) is checked, which turns out to be j. By accessing the j^{th} record in the main data structure, which corresponds to VOQ j, the HOL packet is packet B. Now, the input port proposes to match with output port j. In Figure 2(b), if the proposal is accepted by, and the input port is eventually matched to, output port j, packet B will depart (for output port j) in the current time slot. The head pointer in the j^{th} record of the main data structure is updated to (point to) E, the new HOL packet. These operations, *i.e.,* the search for the HOL packet, and the updates to both data structures, all take $O(1)$ time.

5. STABILITY PROOF OF QPS-SERENA

In this section, we prove that the QPS-Serena algorithm is stable (*i.e.,* can achieve 100% throughput) under any arrival processes that are admissible and satisfy certain mild conditions. In §5.1, we introduce some background information and notations that we need in the stability proofs. In §5.2, we describe a theorem used in [33] to prove the stability of the TASS algorithm. Unfortunately, this theorem is not applicable to QPS-Serena, because QPS-Serena in general does not satisfy the so-called *Property P*, a condition required by the theorem. In §5.3, we state a stronger theorem that requires only a weaker condition than *Property P*, which is satisfied by QPS-Serena.

5.1 Background and Notations

We first define three $N \times N$ matrices $Q(t)$, $A(t)$, and $S(t)$. Let $Q(t) = \big(q_{ij}(t)\big)$ be the queue length matrix where $q_{ij}(t)$ is the length of the j^{th} VOQ at input port i during time slot t. Let $A(t) = \big(a_{ij}(t)\big)$ be the traffic arrival matrix where $a_{ij}(t)$ is the number of packets arriving at the input port i destined for output port j during time slot t, which can be viewed as the counting process associated with underlying traffic arrival process. Let $S(t) = \big(s_{ij}(t)\big)$ be the schedule (matching) matrix for time slot t output by the crossbar scheduling algorithm. As we explained earlier, each $S(t)$ is a 0-1 matrix in which $s_{ij}(t) = 1$ if and only if input port i is matched with output j during time slot t. Then, the queue length matrix Q evolves over time as follows. For $\forall 1 \le i, j \le N$,

$$q_{ij}(t+1) = [q_{ij}(t) + a_{ij}(t) - s_{ij}(t)]^+ \qquad (5)$$

where $[\,\cdot\,]^+$ is defined as $\max\{\,\cdot\,, 0\}$. With a slight abuse of the notation, we rewrite (5), into the matrix form, as $Q(t+1) = [Q(t) + A(t) - S(t)]^+$.

Like in [34], we assume that, for each $1 \le i, j \le N$, $\{a_{ij}(t)\}_{t=0}^{\infty}$ is a sequence of *i.i.d.* random variables, and the second moment of their common distribution ($= \mathbb{E}\big[a_{ij}^2(0)\big]$) is finite. Note that, the same or even stronger assumptions (*e.g.,* Bernoulli *i.i.d.* arrivals) were made for proving the stabilities of TASS [33] and Serena [10] respectively. For ease of presentation, we refer to such an $A(t)$ as an *i.i.d.* arrival (counting) process in the sequel.

Now we flatten the $N \times N$ matrices Q, A, and S into N^2-dimensional vectors in the row-major order, *i.e.,* the first row of the matrix becomes the first N scalars in the vector, the second row becomes the next N scalars, and so on. Now that Q, A, and S are vectors, we can take their inner products, denoted as $\langle\cdot,\cdot\rangle$, in the following derivations. For example, $\langle S(t), Q(t)\rangle$ is the weight of the schedule (matching) $S(t)$, *w.r.t.* the queue length vector $Q(t)$, at time slot t.

5.2 TASS, Serena, and Their Stability

5.2.1 The Adaptive and Non-Degenerative Family

The idea of TASS [33], shown below, is very simple: generate a "fresh" (*i.e.,* independent of all other random vectors) random matching $R(t)$, compare its weight with that of $S(t-1)$, the matching used in the previous time slot, and use the winner as the matching for the current time slot (*i.e.,* $S(t)$). Here $R(t)$ is a random vector whose distribution is parameterized only by the current VOQ length vector $Q(t)$. Amazingly, such a simple adaptive algorithm can achieve 100% throughput, albeit at the cost of higher delays.

$$S(t) = \begin{cases} R(t) & \text{if } \langle Q(t), R(t)\rangle \geq \langle Q(t), S(t-1)\rangle \\ S(t-1) & \text{otherwise} \end{cases} \tag{6}$$

Note that the TASS algorithm is also by definition (*i.e.,* (6)) non-degenerative, defined next.

Definition 1. *A scheduling algorithm is non-degenerative if it guarantees that for any time slot $t \geq 1$, we have*

$$\langle S(t), Q(t)\rangle \geq \langle S(t-1), Q(t)\rangle$$

5.2.2 Generalized Algorithm Family $\widetilde{\Pi}$

Denote Π as the family of adaptive algorithms defined by (6). For the TASS' stability proof and theorem to apply also to Serena, we need to generalize the family of Π to $\widetilde{\Pi}$ that is defined by

$$S(t) = \mathcal{F}\big(R(t), S(t-1), Q(t)\big) \tag{7}$$

where $\mathcal{F}$ is an operator, the resulting $S(t)$ satisfies the *non-degenerative* property defined above, and $R(t)$ is a random schedule whose probability distribution is a function only of $Q(t)$. To ease proving our result, we also force $S(t) = R(t)$ *when all queues (VOQs) are empty* at time slot t, *i.e.,* to "forget the previous schedule $S(t-1)$" and reset to the "default random schedule" $R(t)$.

In TASS, this $\mathcal{F}$ is clearly the "MAX operator", that is, choosing the heavier schedule *w.r.t.* $Q(t)$, between $R(t)$ and $S(t-1)$. In Serena, this $\mathcal{F}$ is the MERGE operator, that is, $S(t) = MERGE\big(R(t), S(t-1), Q(t)\big)$. As we explained in §3.2.2, the MERGE operator combines two matchings into one that is at least as heavy, *w.r.t.* $Q(t)$, as either, so the Serena algorithm, like TASS, is also *non-degenerative*. Hence, Serena also belongs to this extended family $\widetilde{\Pi}$. *Now it is clear that QPS-Serena also belongs to* $\widetilde{\Pi}$ because it differs from Serena only in how the random schedule $R(t)$ is computed, and in QPS-Serena this $R(t)$ is generated in the "$Q(t)$-proportional" manner (so its probability distribution is a function only of $Q(t)$).

We claim that, given any switching algorithm $\pi \in \widetilde{\Pi}$, the joint queueing and scheduling process $\big\{\big(Q(t), S(t)\big)\big\}_{t=0}^{\infty}$, resulting from π and any *i.i.d.* arrival process $A(t)$ (not necessarily admissible), is a Markov chain. This property is clear from the following two facts. First, by (7), $S(t)$ is a function of only $Q(t)$ and $S(t-1)$ (note $R(t)$ is a function only of $Q(t)$). Second, by (5), $Q(t)$ is a function of only $Q(t-1)$, $S(t-1)$, and the random packet arrival vector $A(t)$ that is independent of all other random vectors.

5.2.3 Stability Theorem for Family $\widetilde{\Pi}$

The following theorem, concerning the stability of the family of switching algorithms $\widetilde{\Pi}$, was proven in [33].

Theorem 1. *For any (randomized) algorithm $\pi \in \widetilde{\Pi}$ that satisfies Property P, defined next, and under any admissible i.i.d. arrival process $A(t)$ (defined in §5.1), the joint queueing and scheduling process $\big\{\big(Q(t), S(t)\big)\big\}_{t=0}^{\infty}$ is an ergodic Markov chain, and as a consequence, the queueing process $\{Q(t)\}_{t=0}^{\infty}$ converges in distribution to a random vector $\widehat{Q}$. Furthermore,*

$$\mathbb{E}[\|\widehat{Q}\|_1] < \infty$$

where $\|\cdot\|_1$ is the 1-norm.

Fix a randomized switching algorithm π. Let $W(t) \triangleq \langle S(t), Q(t)\rangle$ be the weight of the schedule output by π at time slot t. Denote as W_Q the weight of the MWM *w.r.t.* a queue length vector Q, *i.e.,* $W_Q \triangleq \max_S\{\langle S, Q\rangle\}$. Let S_Q be one of the schedules that attain this maximum weight (i.e., $\langle S_Q, Q\rangle = W_Q$).

Definition 2 (Property P [33]). *A switching algorithm π satisfies Property P if at any time slot t,*

$$\mathbb{P}\big[W(t) = W_{Q(t)}\big] \geq \delta$$

where $\delta > 0$ is a constant independent of the time slot t and the queue length vector $Q(t)$.

In other words, π satisfies *Property P* if, at any time slot t, the schedule $S(t)$ output by π is a MWM with at least a constant probability δ. Both TASS and Serena satisfy *Property P* because there is a constant (*w.r.t.* $Q(t)$) probability for $R(t)$ to be a MWM in both cases, and when this happens, $S(t)$ remains a MWM after a "MAX" or "MERGE" operation. Since both TASS and Serena also belong to family $\pi \in \widetilde{\Pi}$, Theorem 1 implies that both can achieve 100% throughput.

5.3 Stability of QPS-Serena

Although QPS-Serena also belongs to family $\widetilde{\Pi}$, Theorem 1 is not applicable to QPS-Serena, because it can be shown that QPS-Serena does not satisfy *Property P*. We establish a stronger theorem that allows us to prove that QPS-Serena can achieve 100% throughput. More specifically, we first show in Lemma 1 that QPS-Serena satisfies a weaker condition called (ϵ,δ)-*MWM*, defined next[2]. Then we show

[2]Note that, the definition of (ϵ,δ)-*MWM* is quite different than that of the **1-APRX** (to MWM) defined in [28].

in Theorem 2 that this weaker condition, combined with the $\widetilde{\Pi}$ family membership, is sufficient for a switching algorithm to achieve 100% throughput.

Definition 3. *A switching algorithm π is called (ϵ, δ)-MWM, if $\forall \epsilon > 0$, there exists a constant $0 < \delta \leq 1$ s.t.*

$$\mathbb{P}\big[W(t) \geq (1-\epsilon)W_{Q(t)}\big] \geq \delta$$

where δ is a constant independent of the time slot t and the queue length vector $Q(t)$. Note this δ can depend on ϵ and other (constant) system parameters such as N. Here, $W(t)$ and $W_{Q(t)}$ are similarly defined as before.

In other words, an algorithm π is called (ϵ, δ)-*MWM* if, at any time slot t, the schedule $S(t)$ output by π is within $(1-\epsilon)$ of the optimal (*i.e.,* MWM) with at least a constant probability δ. This condition is clearly weaker than *Property P*, which requires $S(t)$ to be optimal (*i.e.,* MWM) with at least a constant probability.

The following Lemma shows that QPS alone is (ϵ, δ)-*MWM*. Since at any time slot t, QPS-Serena merges $S(t-1)$ with the schedule $R(t)$ output by QPS, resulting in a schedule $S(t)$ that is at least as heavy as $R(t)$, QPS-Serena is also (ϵ, δ)-*MWM*. Therefore, by Theorem 2 below, we conclude that QPS-Serena can achieve 100% throughput.

Lemma 1. *QPS is (ϵ, δ)-MWM.*

We defer its proof of to Appendix D in the interest of space.

Theorem 2. *For every algorithm $\pi \in \widetilde{\Pi}$ that is (ϵ, δ)-MWM, the conclusion of Theorem 1 (i.e., convergence to a stationary distribution with finite first moment) continues to hold, under admissible i.i.d. arrivals.*

We defer its proof to Appendix C in the interest of space. Remark: Like Theorem 2 above, Theorem 1 in [20] also establishes stability with conditions weaker than that are needed in Theorem 1. However, they weaken different parts of the assumptions made in Theorem 1, and hence their proofs are very different. Theorem 2 above weakens *Property P* in Theorem 1 above to $(\epsilon, \delta) - MWM$. In contrast, Theorem 1 in [20] requires *Property P*, but weakens the non-degenerative requirement (see Definition 1) in Theorem 1 above, by allowing it to be violated with a tiny probability.

6. EVALUATION

In this section, we compare the performance of two QPS-augmented algorithms, QPS-iSLIP and QPS-Serena, against the iterative Longest Queue First (iLQF) [16], iSLIP-ShakeUp (iSLIP augmented by ShakeUp [11]), and the two original algorithms, iSLIP [17] and Serena [10]. We evaluate, through simulations, their throughputs and delays under various load conditions and traffic patterns. Maximum Weight Matching (MWM) is also simulated to provide a benchmark for these comparisons.

The evaluation results show conclusively that QPS-iSLIP and QPS-Serena outperform iSLIP and Serena respectively in both throughput and delay. They also show that QPS-iSLIP brings about the same amount of performance improvement to iSLIP as iLQF, even though QPS-iSLIP is far less computationally expensive ($O(\log^2 N)$ per port) than iLQF ($O(N)$ per port), thus giving the "same bang for less buck". Furthermore, they show QPS-iSLIP overall performs better than iSLIP-ShakeUp.

6.1 Simulation setup

In all our simulations, we set the number of input/output ports $N = 32$. Note that we have also investigated how the mean delay performance of various switching algorithms scales with respect to N; these results are shown in Appendix E.2. For the accurate measurement of throughput and delay, each VOQ is assumed to have infinite buffer, so that there is no packet drop at any input port. Every simulation run lasts $6,000 \times N^2$ $(= 6.144 \times 10^6)$ time slots. This duration is chosen so that every simulation run enters the steady state after a tiny fraction of this duration and stays there for the rest. The throughput and delay measurements are taken after the simulation run enters the steady state.

We initially assume Bernoulli *i.i.d.* traffic arrivals: the distributions of arrivals to different input ports are *i.i.d.*, and in each time slot, there is a probability $\rho \in (0, 1)$ that a packet will arrive. We will then look at bursty traffic arrivals further below. The following 4 standard types of load matrices (*i.e.,* traffic patterns) are used for generating the switch's workloads:

1. *Uniform*: packets arriving at any input port go to each output port with probability $\frac{1}{N}$.
2. *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 $\frac{1}{2(N-1)}$.
3. *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).
4. *Diagonal*: packets arriving at input port i go to output port $j = i$ with probability $\frac{2}{3}$, or go to output port $(j \bmod N) + 1$ with probability $\frac{1}{3}$.

The load matrices 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.

In both iSLIP and iLQF, the total number of iterations in a time slot is usually set to $\log_2 N$. However, to achieve a fair comparison between iSLIP, iLQF, and QPS-iSLIP, in simulating these algorithms, the total number of iterations in a time slot is set to $1 + \log_2 N$. For instance, with QPS-iSLIP, this means that we ran 1 iteration of QPS followed by $\log_2 N$ iterations of iSLIP. In doing so, we emphasize that the outperformance of QPS-iSLIP does not come from an extra iteration. Note that, with $1 + \log_2 N$ iterations, the complexity of both iSLIP and QPS-iSLIP remains $O(\log^2 N)$ per port and that of iLQF remains $O(N)$ per port.

For iSLIP-ShakeUp, we alternate between an iSLIP iteration and a ShakeUp iteration *also* for a total of $\log_2 N + 1$ iterations (*i.e.,* $\frac{\log_2 N+1}{2}$ iterations for each). This algorithmic setting and parameter setting both follow the guidelines provided in [11] for iSLIP-ShakeUp, and the throughput numbers we have obtained (shown in Table 1) match those reported in [11].

We consider two performance metrics: throughput and delay. We measure two types of delays: the mean delay and the 95^{th} percentile delay. The 95^{th} percentile delay is the delay value exceeded by exactly 5% of the packets. This 95^{th} percentile delay gauges whether a crossbar scheduling algorithm sacrifices the delay performance of packets in the

Traffic	Uniform	Quasi-diag	Log-diag	Diag
iSLIP	100.00%	81.70%	83.85%	83.47%
QPS-iSLIP	100.00%	99.38%	96.46%	88.36%
iSLIP-ShakeUp	99.98%	91.08%	92.73%	92.41%
iLQF	100.00%	99.41%	96.47%	89.32%

Table 1: *Maximum throughput.*

longest VOQs when evacuating other VOQs. In our simulations, the 95th percentile delay is measured by using the high dynamic range (HDR) histograms [1].

6.2 QPS Throughput Results

We have measured the maximum achievable throughput of iSLIP, QPS-iSLIP, iSLIP-ShakeUp and iLQF, under the 4 different load matrices and an offered load close to 100%. The results are presented in Table 1. We do not include the throughputs of MWM, Serena and QPS-Serena in Table 1 because they provably achieve 100% throughput.

There are three important observations from Table 1. First, for non-uniform traffic patterns, where iSLIP does poorly, QPS-iSLIP significantly boosts the throughput performance of iSLIP, increasing it by an additive term of 0.1768, 0.1261, and 0.0489 for the quasi-diagonal, log-diagonal, and diagonal load matrices respectively. Moreover, for non-uniform traffic, the throughput of QPS-iSLIP are very close to those of iLQF, which is much more expensive computationally. Second, the throughput of QPS-iSLIP is higher than that of iSLIP-ShakeUp under all load matrices except diagonal. Third, just like iSLIP, QPS-iSLIP can achieve 100% throughput under uniform traffic.

We highlight a subtle fact that may sound counterintuitive to some readers: That a switch (running a scheduling algorithm) has a throughput of $\mu < 1$ when the offered load is 100% does not imply that the switch is stable under any offered load (say ρ) smaller than μ. This is because the extra $1-\rho$ "switching resource" freed up by the reduced offered load may not all be efficiently utilized by the scheduling algorithm to clear up the longest queues. For example, iSLIP-ShakeUp is not stable under Quasi-diagonal traffic when the offered load is 90% (see the corresponding missing point in Figure 3 (1st row, 2nd from left)), even though its throughput under 100% offered load is 91.08%. In the sequel, we use the terms "load", "normalized load", "offered load", "traffic load" and "load factor" interchangeably.

6.3 QPS Delay Performance Results

6.3.1 Bernoulli arrivals

Figure 3 (the 1st row) presents the mean delays of iSLIP, QPS-iSLIP, iSLIP-ShakeUp, iLQF, and MWM under the 4 different load matrices. Since iSLIP, QPS-iSLIP, iSLIP-ShakeUp, and iLQF generally cannot achieve 100% throughput, we only measure their delay performance under the offered loads that make them stable; in all figures in the sequel, each "missing point" on a curve indicates that the corresponding scheduling algorithm is not stable under the corresponding offered load.

Figure 3 (the 1st row) clearly shows that QPS-iSLIP has much lower mean delays than iSLIP under all load matrices, especially when the load factor is high (*e.g.,* 80%); we note that the differences between the curves unfortunately look smaller on a log scale (on the y-axis) than they actually are.In addition, the mean delays of QPS-iSLIP are very close to those of iLQF, the more expensive algorithm computationally, under all load matrices and factors.

Figure 3 (the 1st row) also shows that QPS-iSLIP has either similar or slightly higher mean delays than iSLIP-ShakeUp under all load matrices, when the traffic load is low to moderate. However, when the traffic load is high (say $> 80\%$), the iSLIP-ShakeUp either becomes unstable or has higher mean delays than QPS-iSLIP, under all load matrices.

Figure 3 (the 2nd row) presents the mean delays of Serena, QPS-Serena, and MWM under the 4 different load matrices. We can see that QPS-Serena outperforms Serena under all load matrices for all load factors. More specifically, QPS-Serena outperforms Serena by a wide margin, under uniform and diagonal load matrices for all load factors; it does so also under quasi-diagonal and log-diagonal load matrices for load factors that are not too high (≤ 0.8).

Figure 3 (the 2nd row) also shows that the relative difference of the mean delay between QPS-Serena and Serena generally becomes larger as the traffic load becomes lighter. This phenomena is due to the choice of the starter matching. In Serena, the starter matching is the arrival graph, and when the load is light, the arrival graph does not provide enough "cue" for the scheduling algorithm to select the longest VOQs. QPS-Serena, on the other hand, has a better starter matching that accounts for the VOQ lengths under any load conditions, and thus beats Serena in mean delay. The outperformance of QPS-Serena over Serena reinforces our message about the importance of choosing a good starter matching.

Figure 4 (the 1st row) shows the 95th percentile delays of iSLIP, QPS-iSLIP, iSLIP-ShakeUp, iLQF, and MWM under the 4 different load matrices. Due to the presence of delay values that are very close to 0, which would severely "deform" all the curves if they were plotted in a log scale on the y-axis, Figure 4 is plotted in the linear scale on the y-axis. Figure 4 (the 1st row) shows that QPS-iSLIP and iLQF achieve much lower 95th percentile delays than iSLIP and iSLIP-ShakeUp, especially under heavy loads.

Figure 4 (the 2nd row) shows the 95th percentile delays of QPS-Serena, Serena, and MWM under the 4 different load matrices. Again QPS-Serena outperforms Serena by a wide margin under all four load matrices for almost all load factors.

6.3.2 Bursty arrivals

In real networks, packet arrivals are likely to be bursty. In this section, we evaluate the performance of these scheduling algorithms under bursty traffic, generated by a two-state ON-OFF arrival process described in [10]. The durations of each ON (burst) stage and OFF (no burst) stage are geometrically distributed: the probabilities the ON and OFF state lasts for $t \geq 0$ time slots are given by

$$P_{ON}(t) = p(1-p)^t \text{ and } P_{OFF}(t) = q(1-q)^t,$$

with the parameters $p, q \in (0,1)$ respectively. As such, the average duration of the ON and OFF states are $(1-p)/p$ and $(1-q)/q$ time slots respectively.

In an OFF state, an incoming packet's destination (*i.e.,* output port) is generated according to the corresponding load matrix. In an ON state, all incoming packet arrivals to an input port would be destined to the same output port,

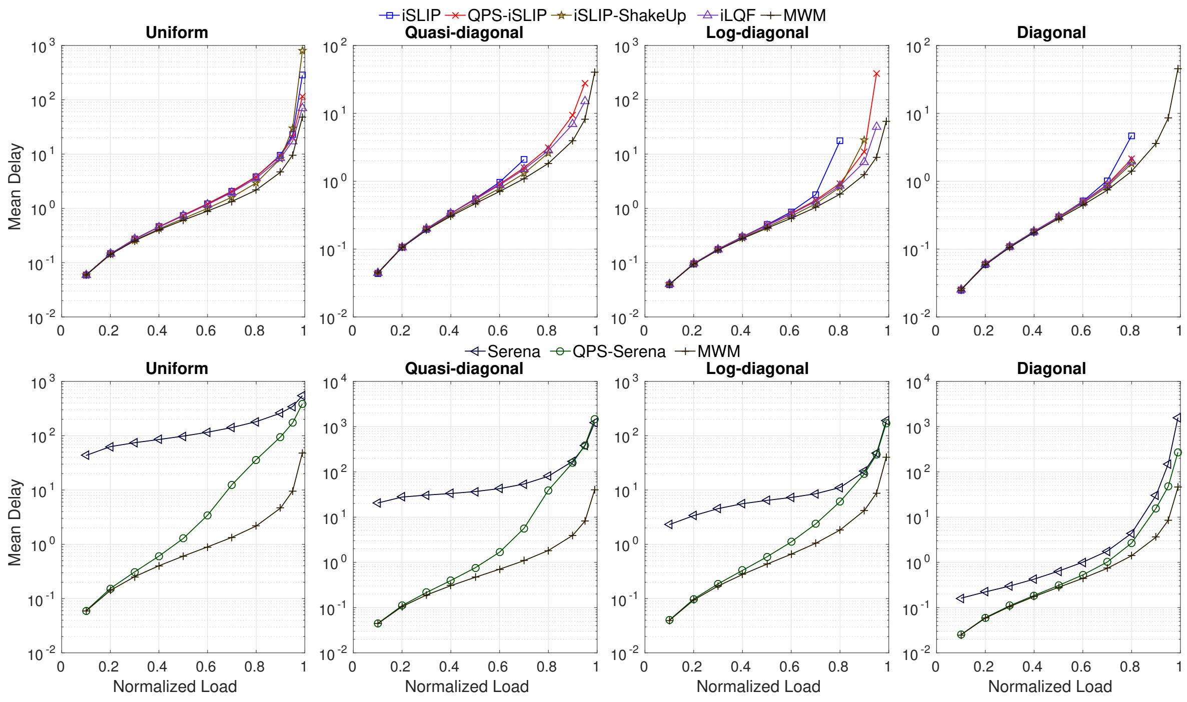


Figure 3: *Mean delays under Bernoulli i.i.d. traffic arrivals with the 4 load matrices.*

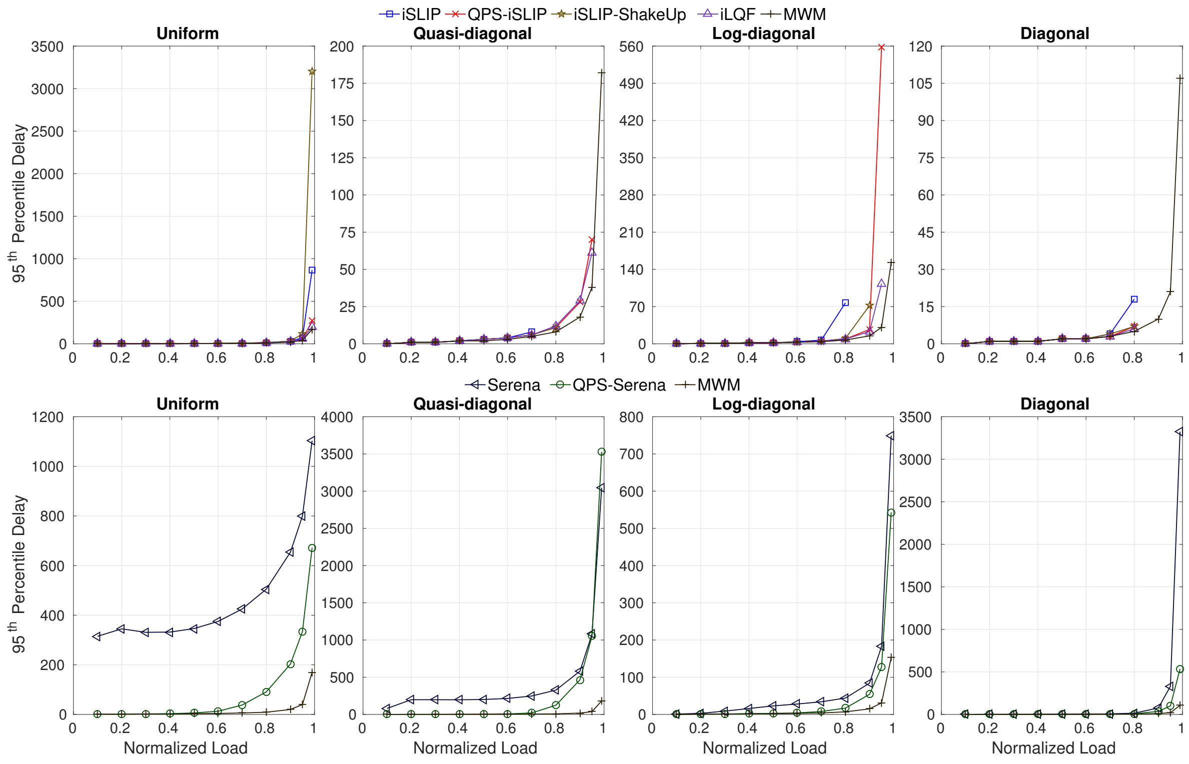


Figure 4: 95th *percentile delay under Bernoulli i.i.d. traffic arrivals with the 4 load matrices.*

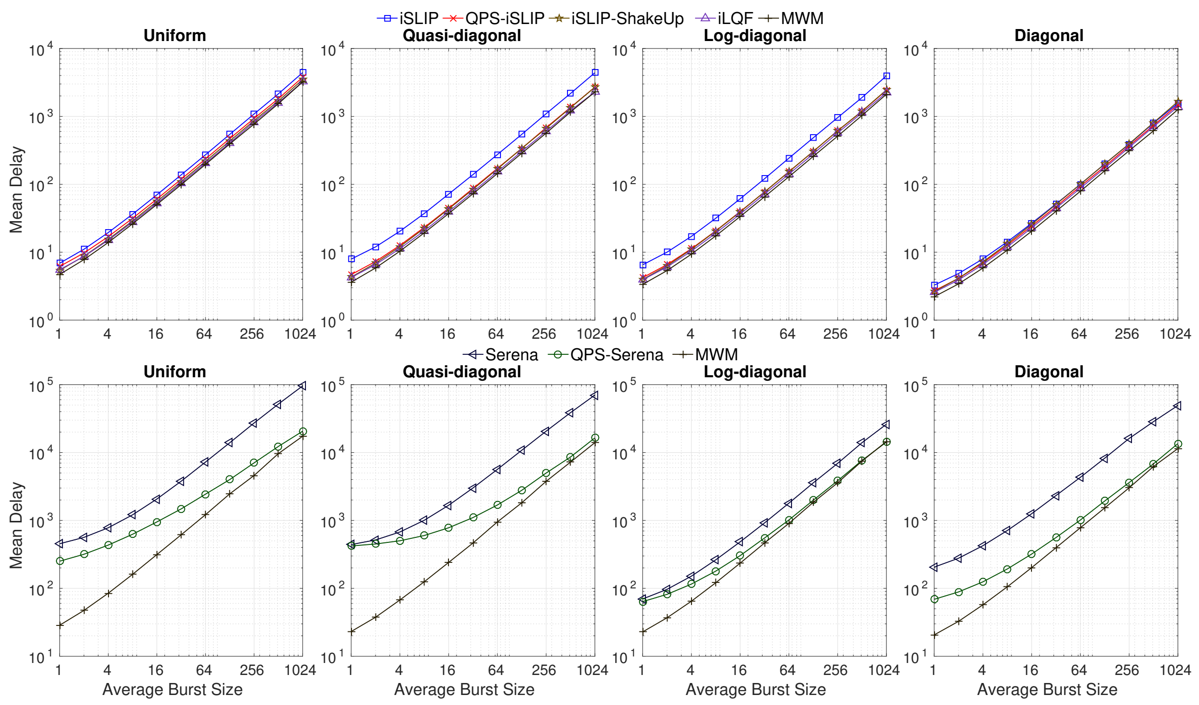


Figure 5: *Mean delays under bursty traffic with the 4 load matrices.*

thus simulating a burst of packet arrivals. By adjusting p, we can control the desired average burst size while by adjusting q, we can control the load of the traffic.

We first compare QPS-iSLIP against iSLIP, iSLIP-ShakeUp, iLQF, and MWM, with average burst sizes ranging from 1 to 1024 packets, on an offered load of 0.75. We use this load factor because iSLIP is not stable under certain load matrices when the offered load is larger than or equal to 0.8.

The simulation results are shown in Figure 5 (the 1^{st} row). We can see that QPS-iSLIP beats iSLIP, and is on par with iLQF and QPS-ShakeUp, under all load matrices for all burst sizes. Furthermore, QPS-iSLIP beats iSLIP by a wide margin, under quasi-diagonal and log-diagonal load matrices. In fact, the starter matching generated by QPS for iSLIP is so superior that QPS-iSLIP is only slightly worse than MWM in the mean delay performance under all load matrices for all burst sizes.

We then evaluate QPS-Serena's mean delay performance against Serena's and MWM's. Figure 5 (the 2^{nd} row) presents the results with average burst sizes ranging from 1 to 1024 packets under an offered load of 0.95, under the 4 load matrices respectively. Performance under other heavy loads, such as at 0.9, is similar to this case.

We can see from Figure 5 (the 2^{nd} row) that the mean delay increases for all scheduling algorithms when the burst size increases, under all 4 load matrices, which is not surprising. However, Figure 5 (the 2^{nd} row) also clearly shows that QPS-Serena handles highly bursty traffic much better than Serena, as we will elaborate next.

We make the following two observations from Figure 5 (the 2^{nd} row). First, QPS-Serena outperforms Serena by an increasingly wider margin, in both absolute and relative terms, as the burst size becomes larger. Second, the gap between QPS-Serena and MWM shrinks rapidly as the burst size becomes larger. Our explanation for the first observation is that, because QPS-Serena obtains information directly from the *current* lengths of the VOQs, rather than indirectly from the *current* arrivals, QPS-Serena reacts to the rapid build-up of packets in a VOQ from a *past* traffic burst much more promptly than Serena. For the second observation, the reason is as follows. When the burst size increases, the longest one or two VOQs at every input port account for an increasingly higher percentage of all packets queued at the input port, and hence have an increasingly higher chances of being sampled by QPS, so the resulting starter matching becomes increasingly closer to an MWM.

7. RELATED WORK

In this section, we first provide a brief survey of crossbar scheduling algorithms or policies, besides those we have already described earlier (including MWM [34], iSLIP [17], iLQF [16], Serena [10], and ShakeUp [11]), focusing on those directly related to our work. Then in §7.2, we compare our QPS strategy with other queue-proportional resource allocation policies.

7.1 Crossbar Scheduling Algorithms

We order the presentations of these algorithms/policies roughly by their (total) computational complexities.

7.1.1 Belief Propagation Algorithms

As explained earlier, although MWM is an ideal algorithm in terms of performance, its most efficient implementation [7] has a prohibitively high computational complexity of $O(N^3)$. Note that MWM-α [14] and MWM-0^+ [31] are variants that only explore the MWM policy space by adopting different edge weight functions; they contain no algorithmic innovations that would reduce the $O(N^3)$ complexity of MWM.

Another family of approximate MWM is the family of distributed iterative algorithms [4,5] based on belief-propagation (BP). In this family, the input ports engage in multiple iterations of message exchanges with the output ports to learn enough information about the lengths of all N^2 VOQs so that each input port can decide on a distinct output port to match with. The resulting matching either is, or is close to, the MWM. Note that the BP-based algorithms are simply parallel algorithms to compute the MWM: the total amount of computation, or the total number of messages needed to be exchanged, is still $O(N^3)$, but is distributed evenly across the input and the output ports (*i.e.,* $O(N^2)$ work for each input/output port).

A technique called BP-assisted scheduling was proposed in a recent work [3], in which BP is used to boost the performance of certain distributed iterative algorithms (called "carrier" algorithms) that are not BP-based such as iLQF [16]. Its idea is to replace the contents of the messages exchanged between input and output ports by those that would be exchanged in a BP-based algorithm. The "BP assistance" part alone has a total computational complexity of $O(N^2)$, so it is best suited for a carrier algorithm that has the same asymptotical complexity, such as iLQF.

7.1.2 MVM and LHPF

Another approach to reducing the complexity to $O(N^{2.5})$ while achieving performance similar to MWM is the family of Maximum Vertex-weighted Matching (MVM) policies [18]. The MVM family was later extended to a larger family called Lazy Heaviest Port First (LHPF) [12] that also has $O(N^{2.5})$ complexity. In a standard MVM policy, each input or output port, denoted as a vertex, is assigned a weight that is equal to the total number of packets (across all N VOQs) queued at the vertex. The weight of an edge (i, j) is the sum of the weights of its two vertices i and j, if there is at least one packet in the corresponding VOQ (*i.e.,* q_{ij}), and is 0 otherwise. An MVM policy dictates that the heaviest (vertex-weighted) matching be used for crossbar scheduling. MVM can achieve 100% throughput, and has a delay performance quite close to that of MWM.

7.1.3 Lower-Complexity Randomized Algorithms

Several randomized algorithms, starting with TASS [33] and culminating in Serena [10, 27] were proposed to push the total complexity further down to $O(N)$ (*i.e.,* linear complexity). We have described in earlier sections both TASS and Serena in details.

A randomized scheduling algorithm specialized for switching variable-size packets was proposed in [38] that has $O(1)$ total computational complexity (per switch). It belongs to a family of randomized algorithms (*e.g.,* [9, 22, 24, 29]) primarily designed for computing a collision-free transmission schedule, which corresponds to an independent set in the interference graph, in a wireless network. These algorithms all build upon a Markov Chain Monte-Carlo (MCMC) technique called Glauber dynamics [36] for computing independent sets (convertible to bipartite matchings in the switching context).

The algorithm in [38] for computing, at each time slot t, the matching for the next time slot $S(t+1)$, works follows. It samples one of the N^2 VOQs (edges) uniformly at random. Suppose the sampled VOQ (edge) is the j^{th} VOQ at input port i (*i.e.,* edge (i, j)). Then, with probability $e^w/(e^w + 1)$, it adds the edge (i, j) (*i.e.,* pairing input port i with output port j) to or keeps the edge in $S(t + 1)$, if neither i nor j is currently matched (in $S(t)$) or (i, j) already belongs to the $S(t)$. Here the weight w is set to the celebrated slowly varying weight function $\ln(\ln(e+x))$ proposed in [24], where x is the weight of the edge (i, j) (*i.e.,* the length of the corresponding VOQ). Clearly, the algorithm makes at most one change (hence $O(1)$ total complexity), from any time slot t to the next, to the configuration of the crossbar (*i.e.,* the matching).

It was proven in [24] that all such algorithms that use this weight function, including the algorithm in [38], can achieve 100% throughput. However, our simulation results (presented in Appendix E.4) show that, when used for switching fixed-size packets, the algorithm in [38] has very poor delay performance and the total queue length does not stabilize (*i.e.,* keeps increasing) until after a very large number of time slots. These simulation results are not surprising: all algorithms that adopt this $\ln\ln(e + \cdot)$ weight function have similar poor delay performance, because as explained in [9], the $\ln\ln(e + \cdot)$ weight function, aimed at achieving 100% throughput [24], reacts very slowly to changes in queue lengths and hence allows long queues to build up.

7.2 Queue-Proportional Resource Allocation

Serving queues at rates or probabilities proportional to their (queue) lengths is an intuitively appealing resource allocation approach that has been used in various computer and communications systems for many years. For example, in [8], a simple queue-proportional scheduler was proposed for scheduling transmissions in wireless broadcast channels, and a geometric programming based formulation of this problem specialized to the Gaussian broadcast channel was later established in [25, 26]. However, unlike our QPS strategy, in which an input port proposes to an output port with a *probability* proportional to the length of the corresponding VOQ, the scheduler in [8, 25, 26] dictates that each link receives an service *rate* proportional to its current queue length during each time slot. As a result, it has to solve a convex optimization problem that has a much higher computational complexity.

In [15], B. Li et al. proposed a generalized version of the above queue-proportional scheduler called Queue-Proportional Rate Allocation (QPRA), with the objective of achieving maximum throughput in a multi-hop wireless network. As the QPRA algorithm is generally hard to implement in practice, they further proposed a low-complexity version called LC-QPRA to make their scheme more practical. The LC-QPRA algorithm resembles the proposing step in our QPS scheme in that, during each time slot, a sender proposes (attempts to transmit) to each receiver with a probability proportional to the length of the corresponding "VOQ".

There are three key differences between QPRA and QPS however. First, in QPRA, during any time slot, the probability with which each sender proposes (to any receiver) is also proportional to its total queue length, whereas in QPS, this probability is 1 for any sender unless its total queue length is 0. Second, in QPRA, if two senders propose (transmit) to the same receiver during a time slot, both transmissions are corrupted, whereas in QPS, only one is allowed to eventually transmit a packet to the receiver. Third, in QPRA, the outcomes (successful or corrupted) of these proposals (attempted transmissions) define the final matching, whereas

QPS only generates a starter matching that will be further refined into a full or more complete matching.

Finally, another policy was proposed in [37] for scheduling packets in a single-hop network, where crossbar scheduling is a special case. However, this policy is closely related to MWM-0^+ [31], and is unrelated to QPRA or QPS.

8. CONCLUSION

In this paper, we propose a new proposing strategy, called queue-proportional sampling (QPS), that generates superior starter matchings than all other known strategies. We use QPS to augment two existing crossbar scheduling algorithms, namely Serena and iSLIP. We show that the augmented algorithms, namely QPS-Serena and QPS-iSLIP, outperform the original algorithms by a wide margin, under various load conditions and traffic patterns. These performance enhancements come at virtually no additional computational cost due to QPS being an $O(1)$ algorithm (per port). Finally, to prove that QPS-Serena can achieve 100% throughput, we have proved a new and stronger stability theorem.

Acknowledgements

This work is supported in part by US NSF grants CNS-1423182 and CNS-1302197 and Australian Research Council grant DP110103505.

9. REFERENCES

[1] Hdrhistogram: A high dynamic range (hdr) histogram. https://github.com/HdrHistogram/HdrHistogram.

[2] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High-speed switch scheduling for local-area networks. *ACM Trans. Comput. Syst.*, 11(4):319–352, Nov. 1993.

[3] S. Atalla, D. Cuda, P. Giaccone, and M. Pretti. Belief-propagation-assisted scheduling in input-queued switches. *IEEE Transactions on Computers*, 62(10):2101–2107, Oct. 2013.

[4] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma. Iterative scheduling algorithms. In *Proceedings of the IEEE INFOCOM*, pages 445–453, Anchorage, AK, USA, May 2007.

[5] M. Bayati, D. Shah, and M. Sharma. Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251, Mar. 2008.

[6] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[7] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, Apr. 1972.

[8] A. Eryilmaz, R. Srikant, and J. R. Perkins. Throughput-optimal scheduling for broadcast channels. In *Proceedings of ITCom (Modeling and Design of Wireless Networks)*, Denver, CO, August 2001.

[9] J. Ghaderi and R. Srikant. On the design of efficient csma algorithms for wireless networks. In *49th IEEE Conference on Decision and Control (CDC)*, pages 954–959, Dec 2010.

[10] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE Journal on Selected Areas in Communications*, 21(4):546–559, May 2003.

[11] M. W. Goudreau, S. G. Kolliopoulos, and S. B. Rao. Scheduling algorithms for input-queued switches: randomized techniques and experimental evaluation. In *Proceedings of the IEEE INFOCOM*, pages 1634–1643 vol.3, Mar. 2000.

[12] G. R. Gupta, S. Sanghavi, and N. B. Shroff. Node weighted scheduling. In *Proceedings of the ACM SIGMETRICS*, pages 97–108, Seattle, WA, USA, Jun. 2009.

[13] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, Dec. 1987.

[14] I. Keslassy and N. McKeown. Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. In *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2001.

[15] B. Li and R. Srikant. Queue-proportional rate allocation with per-link information in multihop networks. In *Proceedings of the ACM SIGMETRICS*, pages 97–108, Portland, OR, USA, Jun. 2015.

[16] N. McKeown. *Scheduling Algorithms for Input-Queued Cell Switches*. PhD thesis, University of California at

Berkeley, May 1995.

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

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

[19] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings of the IEEE INFOCOM*, pages 792–799 vol.2, Mar. 1998.

[20] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, pages 27–38, New York, NY, USA, 2006. ACM.

[21] J. v. Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2:5–12, 1953.

[22] J. Ni, B. Tan, and R. Srikant. Q-csma: Queue-length-based csma/ca algorithms for achieving maximum throughput and low delay in wireless networks. *IEEE/ACM Transactions on Networking*, 20(3):825–836, June 2012.

[23] I. Olkin and A. W. Marshall. *Inequalities: theory of majorization and its applications*. Academic press, 2016.

[24] S. Rajagopalan, D. Shah, and J. Shin. Network adiabatic theorem: An efficient randomized protocol for contention resolution. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 133–144, New York, NY, USA, 2009. ACM.

[25] K. Seong, R. Narasimhan, and J. M. Cioffi. Queue proportional scheduling in gaussian broadcast channels. In *2006 IEEE International Conference on Communications*, volume 4, pages 1647–1652, June 2006.

[26] K. Seong, R. Narasimhan, and J. M. Cioffi. Queue proportional scheduling via geometric programming in fading broadcast channels. *IEEE Journal on Selected Areas in Communications*, 24(8):1593–1602, Aug 2006.

[27] D. Shah, P. Giaccone, and B. Prabhakar. Efficient randomized algorithms for input-queued switch scheduling. *IEEE Micro*, 22(1):10–18, Jan. 2002.

[28] D. Shah and M. Kopikare. Delay bounds for approximate maximum weight matching algorithms for input queued switches. In *Proceedings of the IEEE INFOCOM*, volume 2, pages 1024–1031 vol.2, 2002.

[29] D. Shah and J. Shin. Randomized scheduling algorithm for queueing networks. *The Annals of Applied Probability*, 22(1):128–171, 2012.

[30] D. Shah, N. Walton, and Y. Zhong. Optimal queue-size scaling in switched networks. In *Proceedings of the ACM SIGMETRICS*, pages 17–28, New York, NY, USA, 2012. ACM.

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

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

[33] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proceedings of the IEEE INFOCOM*, pages 533–539, San Francisco, CA, USA, Mar. 1998.

[34] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, Dec. 1992.

[35] R. Tweedie. The existence of moments for stationary markov chains. *Journal of Applied Probability*, pages 191–196, 1983.

[36] E. Vigoda. A note on the glauber dynamics for sampling independent sets. *Electronic Journal of Combinatorics*, 8(1):1–8, 2001.

[37] N. Walton. Concave switching in single and multihop networks. In *Proceedings of ACM SIGMETRICS*, pages 139–151, Austin, TX, USA, Jun. 2014.

[38] S. Ye, T. Shen, and S. Panwar. An $O(1)$ scheduling algorithm for variable-size packet switching systems. In *Proceedings of the 48th Annual Allerton Conference*, pages 1683–1690, Sept. 2010.

APPENDIX

A. QPS VARIANTS

The success we have with QPS leads us to wonder if we can obtain better switching performance by using other proportional sampling strategies. For example, instead of setting sampling probabilities proportional to the lengths of the VOQs, we may set them proportional to the squares of the lengths of the VOQs. More generally, we can set the sampling probabilities proportional to any arbitrary function $f(\cdot)$ of the lengths of the VOQs. We refer to this family of strategies as FQPS, where QPS is a special case (using a linear weight function $f(x) = x$).

To some readers, FQPS may sound similar to MWM-α [14]. They are, however, fundamentally different. The MWM-α work studies the performances of MWM when the weight of a VOQ queue of length x is set to x^α; it does not care how a MWM is computed. FQPS, on the other hand, is about how to better generate a starter matching that can result in a final matching that is as close to the MWM as possible, after a reasonable amount of further computation (*e.g.,* $O(N)$).

We have evaluated the performance of several FQPS variants through simulations. Simulation results (see Appendix E.1) show that the delay performance of QPS can be slightly improved with certain nonlinear weight functions (*e.g.,* with $f(x) = x^2$). However, whereas the computational complexity of QPS is $O(1)$ per packet, other FQPS variants all have a higher computational complexity of $O(\log N)$ per packet. Hence we conclude that QPS overall remains the best practical solution.

B. SPACE COMPLEXITY OF QPS

Each node (packet) in the main data structure contains

3 pointers (2 pointers encoded as "$\langle letter \rangle$" plus 1 for the linked list) and the index of the VOQ (the value j in every node in Figure 2), which needs $\log_2 N$ bits to store (typically less than 2 bytes). Each array entry (packet) in the auxiliary data structure is also a pointer. Note that, in the main data structure, we need an array entry (record) for each VOQ, not for each packet; since the maximum number of packets at an input port is typically much larger than N, the number of VOQs, the memory overhead of these array entries (record), is no more than 2 bytes per packet. Therefore, the memory overhead of the data structures is no more than 4 pointers (4 bytes each) plus 4 bytes, or 20 bytes, per packet.

C. PROOF OF Theorem 2

We have explained in § 5 that, given any switching algorithm $\pi \in \widetilde{\Pi}$, its joint queueing and scheduling process $\{(Q(t), S(t))\}_{t=0}^{\infty}$, under any *i.i.d.* arrival processes $A(t)$ (not necessarily admissible), is a Markov chain. We claim this Markov chain is irreducible and aperiodic, when π is furthermore (ϵ,δ)-*MWM* and $A(t)$ is furthermore admissible. Here we provide only a sketchy justification. To justify the irreducibility, we show that $Q(t)$, starting from any state (*i.e.,* queue lengths) it is currently in, will with a nonzero probability return to the "all-queues-empty" state in a finite number of time slots. To show this property, we claim that, for any integer $\tau > 0$, the switch could, with a nonzero probability, have no packet arrivals to any of its VOQs during $[t, t+\tau]$. This claim is true because, the arrival process $A(t)$ is *i.i.d.*, and for any $1 \leq i \leq N$ and $1 \leq j \leq N$, we have $\beta_{ij} \triangleq P[a_{ij}(t) = 0] > 0$ (otherwise the process $a_{ij}(t)$ is not admissible). Hence, when there are no packet arrivals during $[t, t+\tau]$, which happens with a nonzero probability, a "reasonably good" switching algorithm (being *non-degenerative* and (ϵ,δ)-*MWM*) can clear all the queues during $[t, t+\tau]$, with a sufficiently large τ, and return the $Q(t)$ part of the Markov chain to the "all-queues-empty" state. As to the $S(t)$ part of the Markov chain, the algorithm resets (*i.e.,* returns) $S(t)$ to the default random schedule $R(t)$ when all queues are empty, as explained earlier. Therefore, the Markov chain is irreducible. To justify the aperiodicity of the Markov chain, we note that there is a nonzero probability for the Markov chain to stay at "all-queues-empty" for at least two consecutive time slots.

Now that the Markov chain is irreducible and aperiodic, to prove Theorem 2, it remains to show that (1) the Markov chain is positive recurrent and hence converges to a stationary distribution, and (2) the stationary distribution has a finite first moment. We accomplish both by analyzing the following Lyapunov function $V(\cdot)$ of $Y(t) = (Q(t), S(t))$:

$$V(Y) = V_1(Y) + V_2(Y) \tag{8}$$

where $V_1(Y) = \|Q\|_2^2$, $V_2(Y) = \left([\langle \rho^* S_Q - S, Q\rangle]^+\right)^2$. Here, $\|\cdot\|_2$ is the 2-norm, S_Q is a schedule/matching achieving maximum weight *w.r.t.* Q, and $\rho^* = \frac{1}{2}(1+\rho)$, where $\rho < 1$ is the maximum normalized load imposed on any input or output port as defined in (3). It is clear that $\rho^* < 1$.

Note that, in [33], $V_1(Y)$ is defined in the same way as in this work, whereas $V_2(Y)$ is defined as $V_2(Y) \triangleq \left(\langle S_Q - S, Q\rangle\right)^2$, which is quite different than in this work. We must define $V_2(Y)$ differently here because if its definition in [33] were used instead, there would be an additional positive drift term $c_4 V_1(Y(t))$ on the RHS of (11) (in Lemma 4) which is asymptotically larger than the negative drift term $-\epsilon_1\sqrt{V_1(Y(t))}$ on the RHS of (10) (in Lemma 3), resulting in an overall positive drift on the RHS of (9) when Lemma 3 and Lemma 4 are combined to prove Lemma 2.

The proof of Theorem 2 relies on the following drift condition of $V(Y)$.

Lemma 2. *If the arrivals are admissible i.i.d., then there exists $B, \epsilon > 0$ such that, if $V(Y(t)) > B$, we have,*

$$\mathbb{E}\left[V(Y(t+1)) - V(Y(t)) \mid Y(t)\right] < -\epsilon \|Q(t)\|_2 \tag{9}$$

The proof of Lemma 2 in turn relies on the following two lemmas.

Lemma 3. *If the arrivals are admissible i.i.d., then the drift of the function V_1 satisfies the following inequality*

$$\begin{aligned}&\mathbb{E}\left[V_1(Y(t+1)) - V_1(Y(t)) \mid Y(t)\right]\\ \leq& -\epsilon_1\sqrt{V_1(Y(t))} + 2\sqrt{V_2(Y(t))} + c_1\end{aligned} \tag{10}$$

Here, $\epsilon_1 = \frac{1-\rho}{N}$, $c_1 = \mathbb{E}\left[\|A(t)+\mathbf{1}\|_2^2\right]$ and $\mathbf{1}$ is the vector with all its elements equal to 1.

Lemma 4. *If the arrivals are admissible i.i.d., then the drift of the function V_2 satisfies the following inequality*

$$\begin{aligned}&\mathbb{E}\left[V_2(Y(t+1)) - V_2(Y(t)) \mid Y(t)\right]\\ \leq& -\epsilon_2 V_2(Y(t)) + c_2\sqrt{V_2(Y(t))} + c_3\end{aligned} \tag{11}$$

Here, $\epsilon_2 > 0$ is a constant, $c_2 = 4(\rho+2)N$, $c_3 = 4\mathbb{E}\left[\left(\langle \mathbf{1}, A(t)\rangle + 2N\right)^2\right]$.

C.1 Proof of Lemma 3

By simple calculations and using (5), we have

$$\begin{aligned}&\mathbb{E}\left[V_1(Y(t+1)) - V_1(Y(t)) \mid Y(t)\right]\\ =&\mathbb{E}\left[\|Q(t+1)\|_2^2 - \|Q(t)\|_2^2 \mid Y(t)\right]\\ \leq&\mathbb{E}\left[\langle A(t) - S(t), 2Q(t)\rangle \mid Y(t)\right]\\ &+\mathbb{E}\left[\|A(t) - S(t)\|_2^2 \mid Y(t)\right]\end{aligned} \tag{12}$$

Here, we use the fact that $\|Q(t+1)\|_2^2 = \|[Q(t) + A(t) - S(t)]^+\|_2^2 \leq \|Q(t) + A(t) - S(t)\|_2^2$.

Focusing on the first term $\mathbb{E}\left[\langle A(t) - S(t), 2Q(t)\rangle \mid Y(t)\right]$ above, we have

$$\begin{aligned}&\mathbb{E}\left[\langle A(t) - S(t), 2Q(t)\rangle \mid Y(t)\right]\\ =&\langle \Lambda - S(t), 2Q(t)\rangle\\ =&2\langle \Lambda - \rho^* S_{Q(t)}, Q(t)\rangle + 2\langle \rho^* S_{Q(t)} - S(t), Q(t)\rangle\end{aligned} \tag{13}$$

According to Fact 1 (see (4)), we can decompose Λ as follows: $\Lambda = \sum_{n=1}^{K} \alpha_n M_n$, where $K \leq N^2 - 2N + 2$, $\alpha_n > 0$ for $n = 1, 2, ..., K$, and $\sum_{n=1}^{K} \alpha_n \leq \rho$.

Hence, we have

$$\begin{aligned}
&\langle \Lambda - \rho^* S_{Q(t)}, Q(t)\rangle \\
=&\langle \sum_{n=1}^{K} \alpha_n M_n - \rho^* S_{Q(t)}, Q(t)\rangle \\
=&\langle \sum_{n=1}^{K} \alpha_n M_n - \rho^* S_{Q(t)}, Q(t)\rangle - \sum_{n=1}^{K} \alpha_n W_{Q(t)} + \sum_{n=1}^{K} \alpha_n W_{Q(t)} \\
=&\sum_{n=1}^{K} \alpha_n \big(\langle M_n, Q(t)\rangle - W_{Q(t)}\big) + \big(\sum_{n=1}^{K} \alpha_n - \rho^*\big) W_{Q(t)} \\
\le&\big(\sum_{n=1}^{K} \alpha_n - \rho^*\big) W_{Q(t)} \qquad (14)\\
\le&\big(\rho - \frac{1}{2}(1+\rho)\big) W_{Q(t)} \qquad (15)\\
\le& -\frac{(1-\rho) W_{Q(t)}}{2} \qquad (16)
\end{aligned}$$

Inequality (14) holds because $\forall 1 \le n \le K$ we have $\alpha_n > 0$ and $\langle M_n, Q(t)\rangle - W_{Q(t)} \le 0$ (the weight of M_n is no more than $W_{Q(t)}$, the weight of the MWM *w.r.t.* $Q(t)$) and (15) is due to $\sum_{n=1}^{K} \alpha_n \le \rho$.

Since,

$$\begin{aligned}
W_{Q(t)} &\ge \max_{n=1,\cdots,N^2} q_n(t) \\
&\ge \sqrt{\frac{\|Q(t)\|_2^2}{N^2}} \\
&= \frac{1}{N}\sqrt{V_1\big(Y(t)\big)} \qquad (17)
\end{aligned}$$

From (12), (13), (16) and (17), we have

$$\begin{aligned}
&\mathbb{E}\big[V_1\big(Y(t+1)\big) - V_1\big(Y(t)\big) \mid Y(t)\big] \\
\le& -(1-\rho)\frac{1}{N}\sqrt{V_1\big(Y(t)\big)} + 2\langle \rho^* S_{Q(t)} - S(t), Q(t)\rangle \\
&+ \mathbb{E}\big[\|A(t) - S(t)\|_2^2 \mid Y(t)\big] \\
\le& -\epsilon_1\sqrt{V_1\big(Y(t)\big)} + 2\sqrt{V_2\big(Y(t)\big)} + c_1 \qquad (18)
\end{aligned}$$

Here $\epsilon_1 = \frac{1-\rho}{N}$, $c_1 = \mathbb{E}\big[\|A(t) + \mathbf{1}\|_2^2\big]$ and $\mathbf{1}$ is the vector with all its elements equal to 1.

C.2 Proof of Lemma 4

By simple calculations, we have

$$\begin{aligned}
&\mathbb{E}[V_2\big(Y(t+1)\big) \mid Y(t)] \\
=&\mathbb{P}[\mathcal{E}]\cdot 0 + \mathbb{P}[\mathcal{E}^c]\cdot \mathbb{E}[V_2\big(Y(t+1)\big) \mid Y(t), \mathcal{E}^c] \qquad (19)
\end{aligned}$$

Here, $\mathcal{E}$ is the event $\big\{\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle \le 0\big\}$, and $\mathcal{E}^c$ is the complementary event of $\mathcal{E}$.

Since algorithm π is (ϵ,δ)-*MWM* (see Definition 3), for $\epsilon_3 = 1-\rho^* > 0$, there exists $\delta > 0$, such that,

$$\begin{aligned}
\mathbb{P}[\mathcal{E}^c] &= 1 - \mathbb{P}\big[\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle \le 0\big] \\
&= 1 - \mathbb{P}\big[\rho^* W_{Q(t+1)} - W(t+1) \le 0\big] \\
&= 1 - \mathbb{P}\big[W(t+1) \ge \big(1-(1-\rho^*)\big) W_{Q(t+1)}\big] \\
&= 1 - \mathbb{P}\big[W(t+1) \ge \big(1-\epsilon_3\big) W_{Q(t+1)}\big] \\
&\le 1 - \delta \qquad (20)
\end{aligned}$$

Focusing on the second term in the RHS of (19), we have

$$\begin{aligned}
&\mathbb{E}\big[V_2\big(Y(t+1)\big) \mid Y(t), \mathcal{E}^c\big] \\
=&\mathbb{E}\Big[\big([\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle]^+\big)^2 \mid Y(t), \mathcal{E}^c\Big] \\
=&\mathbb{E}\big[\big(\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle\big)^2 \mid Y(t), \mathcal{E}^c\big] \\
\le&\mathbb{E}\big[\big(\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t) + A(t) - S(t)\rangle \\
&+ N\big)^2 \mid Y(t), \mathcal{E}^c\big] \qquad (21)\\
=&\mathbb{E}\big[\big(\langle \rho^* S_{Q(t+1)}, Q(t)\rangle - \langle S(t+1), Q(t)\rangle + N \\
&+ \langle \rho^* S_{Q(t+1)} - S(t+1), A(t) - S(t)\rangle\big)^2 \mid Y(t), \mathcal{E}^c\big] \qquad (22)
\end{aligned}$$

Here, the term N in (21) is because

$$\begin{aligned}
&\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle \\
=&\langle \rho^* S_{Q(t+1)} - S(t+1), [Q(t) + A(t) - S(t)]^+\rangle \\
=&\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t) + A(t) - S(t)\rangle \\
&+ \langle \rho^* S_{Q(t+1)} - S(t+1), \mathbb{1}_{\{Q(t)+A(t)-S(t)<0\}}\rangle \\
\le&\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t) + A(t) - S(t)\rangle \\
&+ \langle S_{Q(t+1)}, \mathbb{1}_{\{Q(t)+A(t)-S(t)<0\}}\rangle \\
\le&\langle \rho^* S_{Q(t+1)} - S(t+1), Q(t) + A(t) - S(t)\rangle + N
\end{aligned}$$

Here, $\mathbb{1}_{\{Q(t)+A(t)-S(t)<0\}}$ is a vector whose n^{th} element/scalar takes value 1 if $q_n(t) + a_n(t) - s_n(t) < 0$, which happens only when $q_n(t) = 0, a_n(t) = 0, s_n(t) = 1$ and value 0 otherwise. The last inequality is because $\langle S_{Q(t+1)}, \mathbf{1}\rangle \le N$, where $\mathbf{1}$ is the vector with all its elements equal to 1. In the following proof steps, we will use similar tricks to remove $[\,\cdot\,]^+$, which we may not elaborate again.

We now derive the following three inequalities that will be needed to complete our proof.

First, we have

$$\begin{aligned}
&\langle S(t+1), Q(t)\rangle \\
\ge&\langle S(t+1), Q(t+1) - A(t) + S(t)\rangle - N \qquad (23)\\
\ge&\langle S(t), Q(t+1)\rangle - \langle S(t+1), A(t) - S(t)\rangle - N \qquad (24)\\
=&\langle S(t), Q(t) + A(t) - S(t)\rangle + \langle S(t), \mathbb{1}_{\{Q(t)+A(t)-S(t)<0\}}\rangle \\
&- \langle S(t+1), A(t) - S(t)\rangle - N \\
\ge&\langle S(t), Q(t) + A(t) - S(t)\rangle - \langle S(t+1), A(t) - S(t)\rangle - N \\
=&\langle S(t), Q(t)\rangle - \langle S(t+1) - S(t), A(t) - S(t)\rangle - N \\
=&\langle S(t), Q(t)\rangle - \langle S(t+1) - S(t), A(t)\rangle \\
&+ \langle S(t+1) - S(t), S(t)\rangle - N \\
\ge&\langle S(t), Q(t)\rangle - \langle \mathbf{1}, A(t)\rangle - 2N \qquad (25)
\end{aligned}$$

Here, the constant term N in (23) is due to the removal of $[\,\cdot\,]^+$, and (24) is due to the fact that π is *non-degenerative*, *i.e.*, $\langle S(t+1), Q(t+1)\rangle \ge \langle S(t), Q(t+1)\rangle$. The derivation of (25) uses the following two simple facts: $0 \le \langle S(t+1), S(t)\rangle \le N$ and $0 \le \langle S(t), S(t)\rangle \le N$.

Second, we have

$$\langle S_{Q(t+1)}, Q(t)\rangle \le W_{Q(t)} = \langle S_{Q(t)}, Q(t)\rangle \qquad (26)$$

Third, we have

$$\begin{aligned}
&\langle \rho^* S_{Q(t+1)} - S(t+1), A(t) - S(t)\rangle \\
=&\langle \rho^* S_{Q(t+1)} - S(t+1), A(t)\rangle - \langle \rho^* S_{Q(t+1)} - S(t+1), S(t)\rangle \\
\le&\langle S_{Q(t+1)}, A(t)\rangle + \langle S(t+1), S(t)\rangle \\
\le&\langle \mathbf{1}, A(t)\rangle + N \qquad (27)
\end{aligned}$$

Now, according to (25), (26) and (27), we have, conditioned upon the event $\mathcal{E}^c$,

$$\begin{aligned}0 <& \langle \rho^* S_{Q(t+1)} - S(t+1), Q(t+1)\rangle \\ \leq& \langle \rho^* S_{Q(t+1)}, Q(t)\rangle - \langle S(t+1), Q(t)\rangle + N \\ &+ \langle \rho^* S_{Q(t+1)} - S(t+1), A(t) - S(t)\rangle \\ \leq& \langle \rho^* S_{Q(t)}, Q(t)\rangle - \big(\langle S(t), Q(t)\rangle - \langle \mathbf{1}, A(t)\rangle \\ &- 2N\big) + N + \big(\langle \mathbf{1}, A(t)\rangle + N\big) \\ \leq& \langle \rho^* S_{Q(t)} - S(t), Q(t)\rangle + 2\langle \mathbf{1}, A(t)\rangle + 4N \\ \leq& \sqrt{V_2\big(Y(t)\big)} + 2\big(\langle \mathbf{1}, A(t)\rangle + 2N\big) \end{aligned} \tag{28}$$

Therefore, we have

$$\begin{aligned}&\mathbb{E}\big[V_2\big(Y(t+1)\big) \mid Y(t), \mathcal{E}^c\big] \\ \leq& V_2\big(Y(t)\big) + 4\big(\mathbb{E}\big[\langle \mathbf{1}, A(t)\rangle \mid \mathcal{E}^c\big] + 2N\big)\sqrt{V_2\big(Y(t)\big)} \\ &+ 4\mathbb{E}\big[\big(\langle \mathbf{1}, A(t)\rangle + 2N\big)^2 \mid \mathcal{E}^c\big]\end{aligned} \tag{29}$$

Since $\langle \mathbf{1}, A(t)\rangle \geq 0$, we have,

$$\begin{aligned}&\mathbb{E}\big[\langle \mathbf{1}, A(t)\rangle \mid \mathcal{E}^c\big]\mathbb{P}[\mathcal{E}^c] \\ \leq& \mathbb{E}\big[\langle \mathbf{1}, A(t)\rangle \mid \mathcal{E}^c\big]\mathbb{P}[\mathcal{E}^c] + \mathbb{E}\big[\langle \mathbf{1}, A(t)\rangle \mid \mathcal{E}\big]\mathbb{P}[\mathcal{E}] \\ =& \mathbb{E}\big[\langle \mathbf{1}, A(t)\rangle\big] \\ \leq& \rho N\end{aligned} \tag{30}$$

Here, $\rho < 1$ is the maximum normalized load imposed on any input or output port as defined in (3).

Similarly, we have

$$\begin{aligned}&\mathbb{E}\big[\big(\langle \mathbf{1}, A(t)\rangle + 2N\big)^2 \mid \mathcal{E}^c\big]\mathbb{P}[\mathcal{E}^c] \\ \leq& \mathbb{E}\big[\big(\langle \mathbf{1}, A(t)\rangle + 2N\big)^2\big]\end{aligned} \tag{31}$$

Substituting (20), (29), (30) and (31) into (19), we have

$$\begin{aligned}&\mathbb{E}[V_2\big(Y(t+1)\big) \mid Y(t)] \\ \leq& (1-\delta)V_2\big(Y(t)\big) + c_2\sqrt{V_2\big(Y(t)\big)} + c_3\end{aligned} \tag{32}$$

where $c_2 = 4(\rho+2)N$, $c_3 = 4\mathbb{E}\big[\big(\langle \mathbf{1}, A(t)\rangle + 2N\big)^2\big]$.

Therefore, we have

$$\begin{aligned}&\mathbb{E}\big[V_2\big(Y(t+1)\big) - V_2\big(Y(t)\big) \mid Y(t)\big] \\ \leq& -\delta V_2\big(Y(t)\big) + c_2\sqrt{V_2\big(Y(t)\big)} + c_3\end{aligned} \tag{33}$$

Hence, Lemma 4 holds with $\epsilon_2 = \delta$.

C.3 Proof of Lemma 2

We now proceed to prove Lemma 2. Note that, the proof is the same as the proof of *Lemma 1* in [33]. We reproduce it with some minor revisions for this paper to be self-contained.

By Lemma 3 (concerning the drift of $V_1(Y)$) and Lemma 4 (concerning the drift of $V_2(Y)$), the drift of $V(Y)$ satisfies

$$\begin{aligned}&\mathbb{E}\big[V\big(Y(t+1)\big) - V\big(Y(t)\big) \mid Y(t)\big] \\ \leq& -\epsilon_1\sqrt{V_1\big(Y(t)\big)} + (2+c_2)\sqrt{V_2\big(Y(t)\big)} - \epsilon_2 V_2\big(Y(t)\big) \\ &+ c_1 + c_3\end{aligned} \tag{34}$$

When $V\big(Y(t)\big) \geq B$, we have $V_1\big(Y(t)\big) \geq B - V_2\big(Y(t)\big)$, and hence

$$-\epsilon_1\sqrt{V_1\big(Y(t)\big)} \leq -\frac{\epsilon_1}{2}\sqrt{V_1\big(Y(t)\big)} - \frac{\epsilon_1}{2}\sqrt{B - V_2\big(Y(t)\big)} \tag{35}$$

Substituting the first term in the RHS of (34) by the RHS of (35), we obtain

$$\begin{aligned}&\mathbb{E}\big[V\big(Y(t+1)\big) - V\big(Y(t)\big) \mid Y(t)\big] \\ \leq& -\frac{\epsilon_1}{2}\sqrt{V_1\big(Y(t)\big)} - \frac{\epsilon_1}{2}\sqrt{B - V_2\big(Y(t)\big)} + (2+c_2)\sqrt{V_2\big(Y(t)\big)} \\ &- \epsilon_2 V_2\big(Y(t)\big) + c_1 + c_3\end{aligned} \tag{36}$$

It is clear that when B is large enough, we have,

$$\begin{aligned}&-\frac{\epsilon_1}{2}\sqrt{B - V_2\big(Y(t)\big)} + (2+c_2)\sqrt{V_2\big(Y(t)\big)} \\ &- \epsilon_2 V_2\big(Y(t)\big) + c_1 + c_3 < 0\end{aligned}$$

Hence,

$$\begin{aligned}&\mathbb{E}\big[V\big(Y(t+1)\big) - V\big(Y(t)\big) \mid Y(t)\big] \\ <& -\frac{\epsilon_1}{2}\sqrt{V_1\big(Y(t)\big)} \\ =& -\frac{\epsilon_1}{2}\|Q(t)\|_2\end{aligned} \tag{37}$$

Hence Lemma 2 holds with $\epsilon = \frac{\epsilon_1}{2}$. Here $\epsilon_1 = \frac{1-\rho}{N}$ as specified in Lemma 3 (see (10)).

C.4 Proof of Theorem 2

To prove Theorem 2, we need a theorem due to Tweedie [35], stated as follows.

THEOREM 3 (TWEEDIE [35]). *Suppose that $\{Y_n\}_{n=0}^{\infty}$ is an aperiodic and irreducible Markov chain with countable state space $\mathcal{Y}$. Let $f(Y), g(Y)$ be real nonnegative functions such that $g(Y) \geq f(Y), Y \in D^c$, where D is a finite subset of $\mathcal{Y}$. If*

$$\mathbb{E}\big[g(Y_1) \mid Y_0 = Y\big] < \infty, \quad Y \in D \tag{38}$$

and

$$\mathbb{E}\big[g(Y_1) \mid Y_0 = Y\big] < g(Y) - f(Y), \quad Y \in D^c \tag{39}$$

then the Markov chain is ergodic and

$$\mathbb{E}\big[f(\widehat{Y})\big] < \infty$$

where the random variable $\widehat{Y}$ has the steady state distribution of the Markov chain $\{Y_n\}_{n=0}^{\infty}$.

Remarks: In the above theorem, Y_0, Y_1 can be replaced by Y_n, Y_{n+1}, respectively, for any integer $n \geq 0$, since $\{Y_n\}_{n=0}^{\infty}$ is a Markov chain.

Now, we can proceed to prove Theorem 2. Note that, the proof of Theorem 2 here, using Lemma 2 and Theorem 3, is mostly the same as in [33].

Let $Y_t = Y(t) = \big(Q(t), S(t)\big)$. Then Y_t is an irreducible and aperiodic Markov chain (explained in Appendix C). Define $f, g : \mathcal{Y} \to \mathbb{R}^+$ be such that

$$g(Y) = V(Y), f(Y) = \frac{\epsilon_1}{2}\|Q\|_2$$

where $Y = (Q, S)$ and $\epsilon_1 = \frac{1-\rho}{N}$ which is the same as in (37). Let $D^c = \big\{Y : V(Y) > B\big\}$, for B specified in the proof of Lemma 2. It is clear that (38) holds from the definition of D^c. By Lemma 2 (note $\epsilon = \frac{\epsilon_1}{2}$ in Lemma 2), Inequality (39) also holds (by replacing Y_t and Y_{t+1} in (9) by Y_0 and Y_1 respectively). By Theorem 3, we have that the Markov chain $Y(t) = \big(Q(t), S(t)\big)$ converges in distribution to $\widehat{Y} = (\widehat{Q}, \widehat{S})$, and that $\mathbb{E}\big[f(\widehat{Y})\big] < \infty$. Therefore, $\mathbb{E}\big[\|\widehat{Q}\|_2\big] = \frac{2}{\epsilon_1}\mathbb{E}\big[f(\widehat{Y})\big] < \infty$.

Given any outcome ω, the (deterministic) N^2-dimensional vector satisfies

$$\|\widehat{Q}(\omega)\|_1 \leq N\|\widehat{Q}(\omega)\|_2$$

by the *Cauchy-Schwarz inequality.*

Therefore, by the Dominated Convergence Theorem, we have

$$\mathbb{E}\big[\|\widehat{Q}\|_1\big] \leq N\mathbb{E}\big[\|\widehat{Q}\|_2\big] < \infty$$

This completes the proof of Theorem 2.

D. PROOF OF Lemma 1

Let Q be the VOQ length vector at the current time t; we do not use the notation $Q(t)$ here because the proof does not involve the term t. Let S_Q be a maximum weight matching *w.r.t.* Q, and let W_Q denote its weight. Given any $\epsilon > 0$, we derive another matching $S' \subseteq S_Q$ from S_Q as follows: remove every edge (*i.e.,* VOQ) from S_Q whose weight (*i.e.,* VOQ length) is less than $\frac{\epsilon}{N}W_Q$. Since there can be at most N edges in any matching, the weight of S' satisfies $\langle S', Q\rangle \geq W_Q - N\cdot\frac{\epsilon}{N}W_Q > (1-\epsilon)W_Q$.

Recall that in the proposing phase, QPS samples a set of edges (not necessarily a matching), which we denote as U. Next, we prove that, U contains all edges in S' (*i.e.,* $S' \subseteq U$) with at least a constant (*i.e.,* not as a function of Q) probability $\delta = \big(\frac{\epsilon}{N^2}\big)^N$. Given any edge $e = (i,j) \in S'$ (*i.e.,* j_{th} VOQ at input port i), its weight is at least $\frac{\epsilon}{N}W_Q$ since all edges lighter than that would have been removed earlier. Since the weight of any edge can be at most W_Q, the total weight of all edges (VOQs) incident on vertex (input port) i is at most NW_Q. Hence the probability that this edge $e = (i,j)$ (*i.e.,* output port j) is sampled by input port i in the QPS proposing phase is at least $(\frac{\epsilon}{N}W_Q)/(NW_Q) = \frac{\epsilon}{N^2}$. Since every input port makes the sampling decision independently, the probability that all edges in S' are sampled during the QPS proposing phase is at least $\big(\frac{\epsilon}{N^2}\big)^{|S'|} \geq \big(\frac{\epsilon}{N^2}\big)^N$, where $|S'|$ is the number of edges in S'.

Now suppose the event $S' \subseteq U$ happens during the QPS proposing phase. We show that the final matching accepted by the output ports, during the QPS accepting phase, is at least as heavy as S'. This is however clear from the following two facts. First, given any edge $e = (i,j) \in S'$, it is either accepted by output port j or beaten by another edge (*i.e.,* proposal) e' to output port j that has a heavier (or equal) weight (VOQ length). Second, when the latter happens, since S' is a matching, e' will not compete with (and beat) any edge in S' other than e.

Remark. Lemma 1 continues to hold if the "longest VOQ first" accepting strategy is replaced by the aforementioned proportional accepting (PA) strategy (see §3.1). Let $\mathcal{E}$ be the event that S' is contained in the final matching. To prove this remark, it suffices to show that there is a constant (*i.e.,* not as a function of Q) probability for $\mathcal{E}$ to happen, conditioned upon the happening of the event $S' \subseteq U$. Using the same argument as above for proving that the event $S' \subseteq U$ happens with a probability that is at least $\big(\frac{\epsilon}{N^2}\big)^N$, we can prove that $\mathcal{E}$ happens conditionally with a probability that is at least $\big(\frac{\epsilon}{N^2}\big)^N$.

E. MORE SIMULATION RESULTS

E.1 Mean Delay Performance for FQPS

Here, we consider several alternative functions $f(\cdot)$ of the queue lengths for FQPS, besides the VOQ lengths (*i.e.,* $f(x) = x$) used in QPS, to see if they can deliver better mean delay performance than QPS. We present the simulation results for two types of functions:

1. $f(x) = x^\alpha$ for $\alpha = 2, 3, 4, \infty$: inspired by the functions considered in MWM-α [14], and
2. $f(x) = \log(x+1)$: inspired by the log-weights used in MWM-0^+ [32].

The case $\alpha = \infty$ is an extreme case in which each input port samples the longest VOQ (with ties broken uniformly randomly) and proposes to the corresponding output port.

Figure 6 presents the mean delays of FQPS-augmented scheduling algorithms under the 4 load matrices and a range of normalized loads. By selecting a proper α, we can indeed achieve marginal improvements (*e.g.,* when $\alpha = 2, 3, 4$). However, when $\alpha \to \infty$, the mean delay increases dramatically when the load is high. This is not surprising because at high loads, a high α strategy severely penalizes short VOQs by blocking them from being serviced until they themselves become long enough, resulting in poor delay performance. Furthermore, the mean delays of the scheduling algorithms are similar when the load is light, but as the load increases, the performance gaps between the FQPS-augmented algorithms with different α values increase (though the differences remain small). Surprisingly, unlike MWM-α where mean the delay increases as α increases [14], for FQPS, the relationship between the mean delay and α is not so straightforward. On one hand, the mean delay performance is generally slightly better in cases $\alpha = 2, 3, 4$ than that in QPS (*i.e.,* $\alpha = 1$). On the other hand, in the case $\alpha = \infty$, the mean delay performance becomes much worse than that in QPS.

We also see that, unlike in MWM-0^+ [32], using $f(x) = \log(x+1)$ for FQPS actually increases the mean delay, as compared to QPS. The reason for this is that the use of the $\log(\cdot)$ weight function results in the probabilities of sampling the longer VOQs being very close to those of sampling the shorter queues. Such an almost weight-oblivious way of sampling intuitively does not yield good performance.

While there is slight improvement in mean delay for properly selected α under all load matrices, from Figure 6, we see that the difference between QPS-Serena (QPS-iSLIP) and the FQPS-Serena (FQPS-iSLIP) is, at best, marginal. Implementing FQPS, however, requires more complex data structures (and more space), such as a binary search tree. Such an implementation requires $O(\log N)$ (per packet) time complexity for the operations (insertion, deletion, *etc.*). In contrast, the $O(1)$ complexity of QPS makes it a far more attractive and practical solution. To summarize, all factors considered, QPS offers the best tradeoff between performance and computational/implementation complexities within the FQPS family.

E.2 How Mean Delay Scales with N

In the section, we investigate how the mean delays for QPS-augmented scheduling algorithms scale with the number of input/output ports N. We have simulated four different N values: $N = 16, 32, 64, 128$.

Figure 7 (the 1^{st} row) shows the mean delays for QPS-iSLIP, iSLIP, iSLIP-ShakeUp, iLQF, and MWM under the

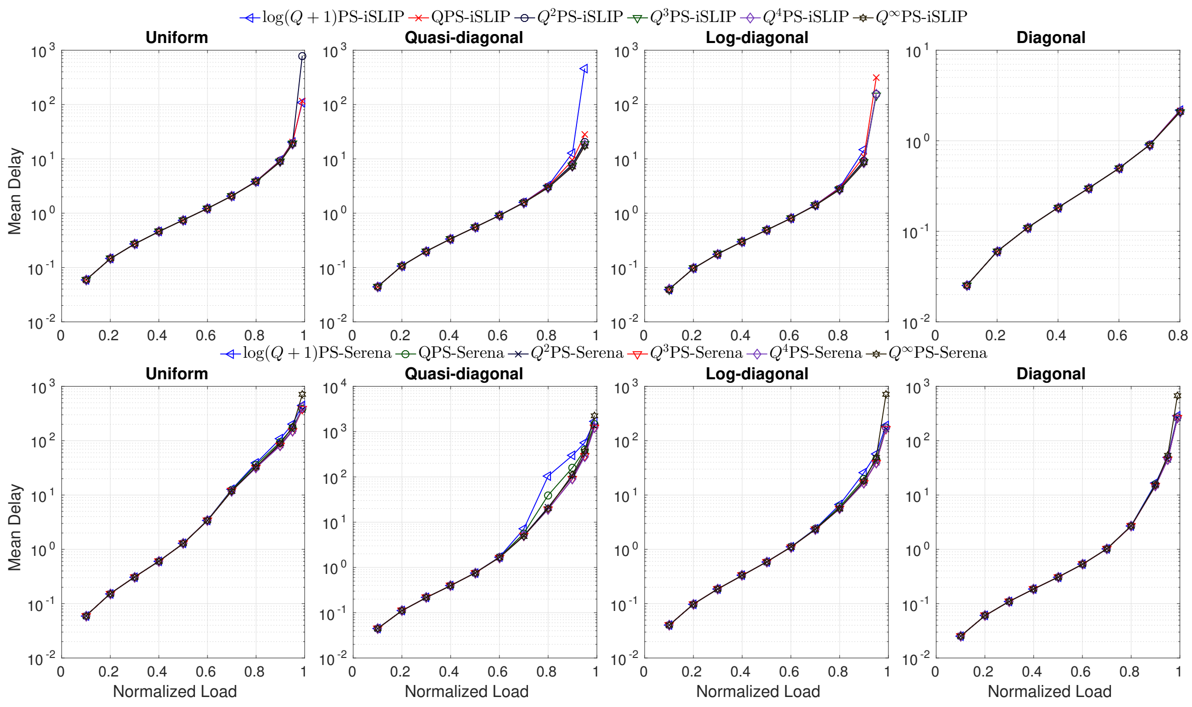


Figure 6: *Mean delays for different FQPS-iSLIP and FQPS-Serena under Bernoulli i.i.d. traffic arrivals with the 4 load matrices.*

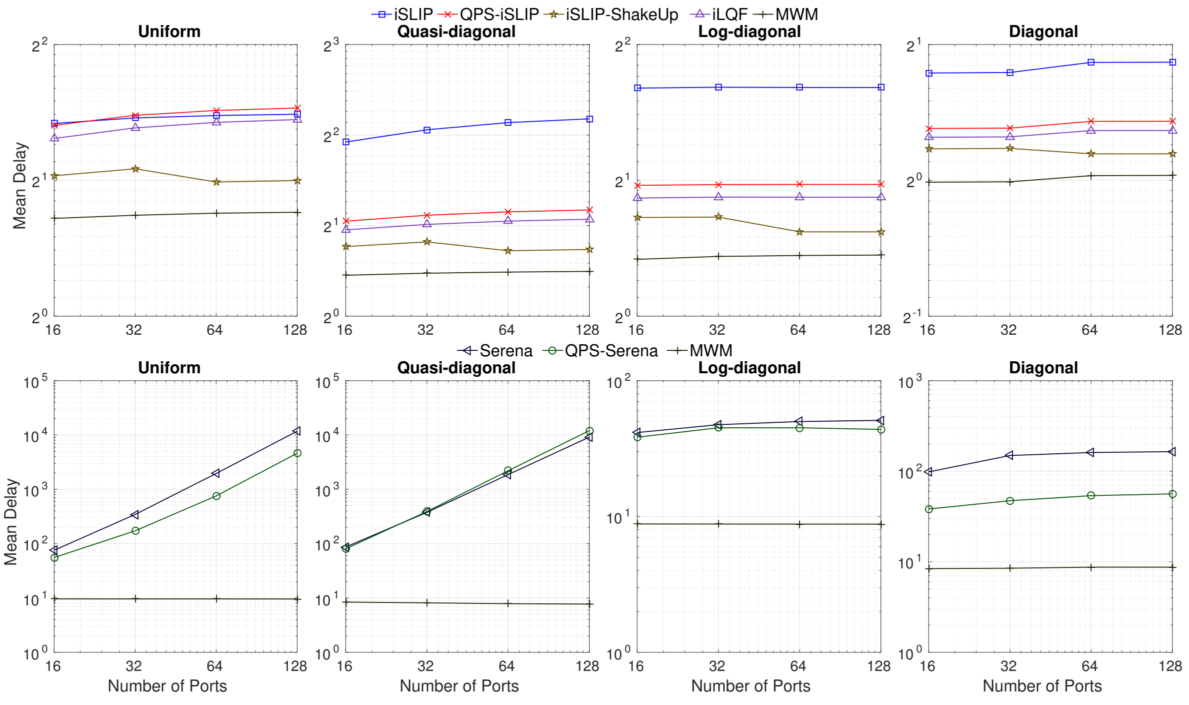


Figure 7: *Mean delays versus number of ports for different scheduling algorithms under Bernoulli i.i.d. traffic arrivals with the 4 load matrices.*

normalized load of 0.75 (some algorithms are not stable under load factor 0.8) and the 4 different load matrices. From Figure 7, we can see that all four scheduling algorithms scale quite well under all 4 load matrices: In every case, the mean delay nearly remains constant when N increases.

In Figure 7 (both rows), the mean delay of MWM (under 0.75 load in the 1^{st} row and 0.95 load in the 2^{nd} row) is nearly a constant *w.r.t.* N. This scaling behavior of MWM to a certain degree confirms a theoretical result proved in [30]. It states that the average total queue length (across all N input ports) under an optimal algorithm (*e.g.,* MWM) scales linearly with N as $\frac{N}{1-\rho}$, where $\rho \in (0,1)$ is the load factor. Suppose this total average queue length is furthermore nearly evenly distributed across the N input ports by an optimal algorithm, the mean delay (proportional to the average per-port queue length in the steady state) is expected to be nearly constant when N increases.

Figure 7 (the 2^{nd} row) shows the mean delays for QPS-Serena against Serena and MWM under the normalized load of 0.95 and the 4 different load matrices. As we can see, QPS-Serena outperforms Serena and the gap increases when N increases, under all load matrices except the quasi-diagonal. In addition, under the log-diagonal and the diagonal load matrices, both QPS-Serena and Serena achieve near-optimal scaling (*i.e.,* nearly constant as a function of N) of mean delay, whereas under the uniform and the quasi-diagonal load matrices, the mean delay grows roughly quadratically with N (*i.e.,* $O(N^2)$ scaling).

E.3 "Longest VOQ First" vs. Proportional Accepting

In this section, we compare the performance between the two different accepting strategies we proposed in §3.1: "longest VOQ first" and proportional accepting (PA). Figure 8 compares QPS-iSLIP with the 2 different accepting strategies (the 1^{st} row) and QPS-Serena with the 2 different accepting strategies (the 2^{nd} row), in terms of mean delay, under Bernoulli *i.i.d.* traffic arrivals with the 4 different load matrices. Similarly, in Figure 9, the 1^{st} row compares QPS-iSLIP with the 2 different accepting strategies under bursty traffic arrivals with an offered load of 0.75, and the 2^{nd} row compares QPS-Serena with the 2 different accepting strategies under bursty traffic with an offered load of 0.95. Figure 8 and Figure 9 show that PA results in either slightly worse or similar mean delay performances than "longest VOQ first" in all these scenarios.

E.4 QPS vs. O(1) Algorithm in [38]

Figure 10 compares QPS-iSLIP and QPS-Serena against the $O(1)$ scheduling algorithm in [38], in terms of mean delay, under Bernoulli *i.i.d.* traffic arrivals with the 4 different load matrices. Figure 10 clearly shows that the mean delays of the $O(1)$ algorithm in [38] are between 3 and 4 orders of magnitudes larger than those of QPS-iSLIP and QPS-Serena under all workload conditions. Note that in Figure 10, only mean delays under offered loads ≤ 0.8 (and ≤ 0.6 for quasi-diagonal load matrices) are reported for the $O(1)$ algorithm in [38], because its simulation could not reach the steady state within a reasonable amount of time, when the offered load is higher than that. As explained earlier, this phenomenon is expected, because such Glauber Dynamics based scheduling algorithms converge to the steady state very slowly when the number of ports (or wireless nodes) N is large and the traffic load is high [9]. Indeed, for the $O(1)$ algorithm to converge under an offered load of just 0.8, we had to increase the number of time slots in the simulation to at least $20000 \times N^2$, which is more than three times that was necessary for any other simulation run.

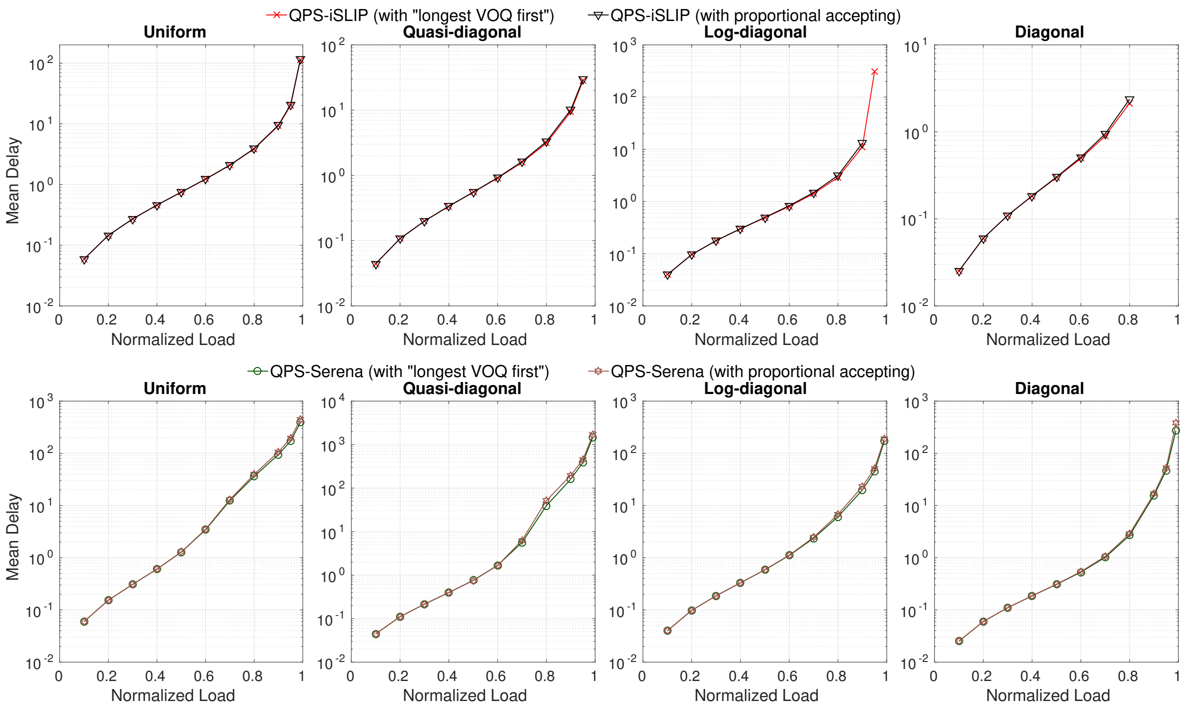


Figure 8: *Mean delays for QPS-iSLIP and QPS-Serena with the 2 different accepting strategies under Bernoulli i.i.d. traffic arrivals with the 4 load matrices.*

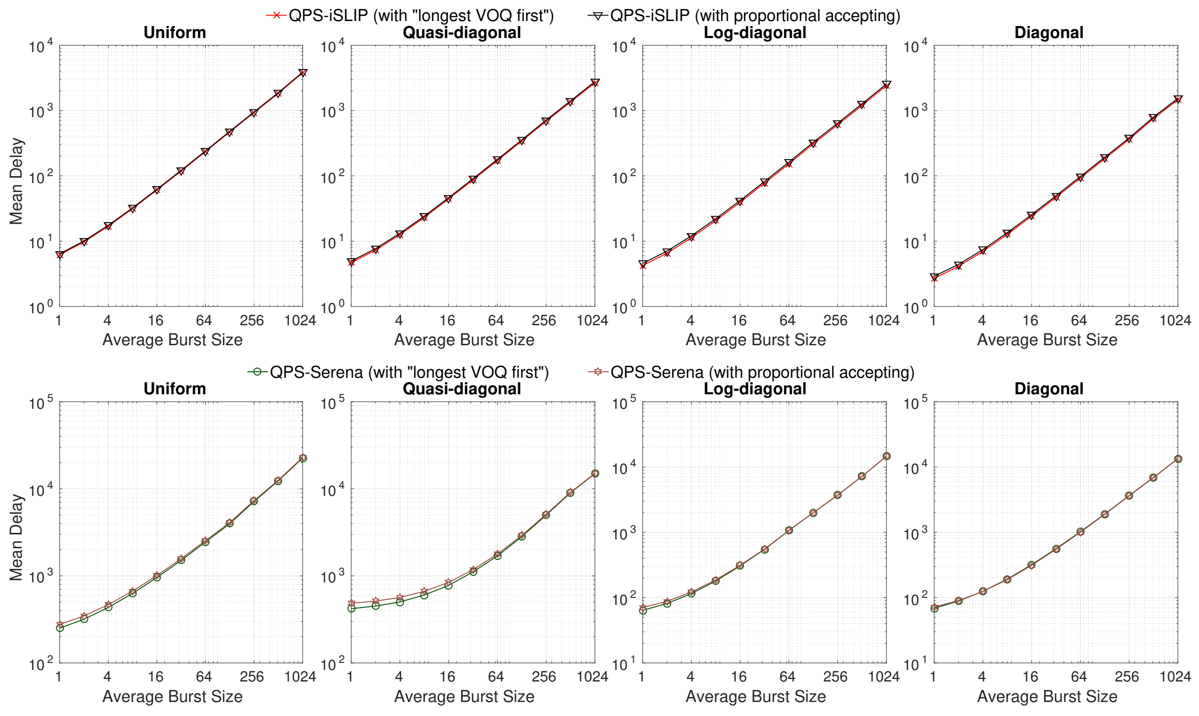


Figure 9: *Mean delays for QPS-iSLIP (offered load: 0.75) and QPS-Serena (offered load: 0.95) with the 2 different accepting strategies under bursty traffic arrivals with the 4 load matrices.*

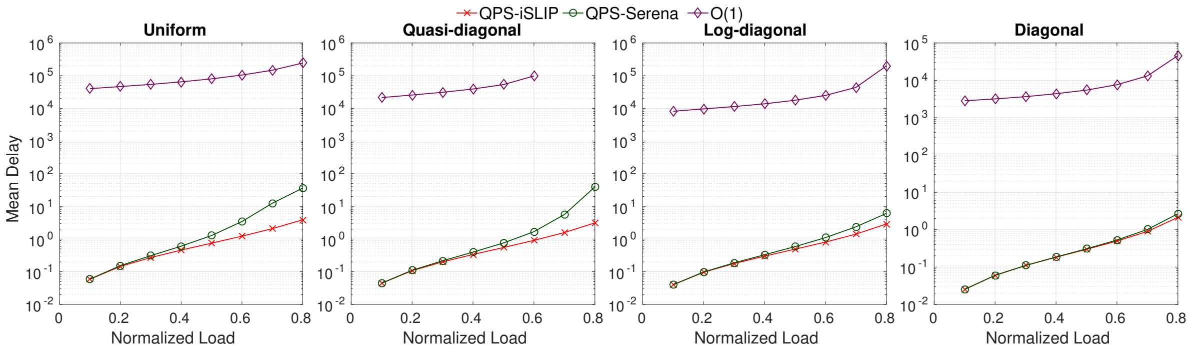


Figure 10: *Mean delay for QPS-iSLIP/QPS-Serena against* $O(1)$ *algorithm in [38] under Bernoulli i.i.d. traffic arrivals with the* 4 *load matrices.*