# Providing 100% Throughput in a Buffered Crossbar Switch

Yanming Shen, Shivendra S. Panwar, H. Jonathan Chao
Department of Electrical and Computer Engineering
Polytechnic University

*Abstract*— **Buffered crossbar switches have received great attention recently because they have become technologically feasible, have simpler scheduling algorithms, and achieve better performance than a bufferless crossbar switch. Buffered crossbar switches have a buffer placed at each crosspoint. A cell is first delivered to a crosspoint buffer and then transferred to the output port. With a speedup of two, a buffered crossbar switch has previously been proved to provide 100% throughput. We propose what we believe is the first feasible scheduling scheme that can achieve 100% throughput without speedup and a finite crosspoint buffer. The proposed scheme is called SQUISH: a Stable Queue Input-output Scheduler with Hamiltonian walk. With SQUISH, each input/output first makes decisions based on the information from the virtual output queues and crosspoint buffers. Then it is compared with a Hamiltonian walk schedule to avoid possible "bad" states. We then prove that SQUISH can achieve 100% throughput with a speedup of one. Our simulation results also show good delay performance for SQUISH.**

## I. Introduction

With the growing demand of Internet traffic, there is an increasing interest in designing high performance packet switches. Due to the memory speed constraint, input queueing or combined input and output queueing (CIOQ) is used, with bufferless crossbar switching fabrics. With input-queueing, at each input port, there is a separate queue corresponding to each output, known as virtual output queues (VOQs), to avoid head-of-line (HOL) blocking [1]. A bufferless crossbar switching fabric is used to transfer cells from inputs to outputs. However, such switches usually require complex scheduling algorithms to achieve good performance; these include maximum weight matching [2], [3], maximal [4] and maximum size matching [5], or iterative schedulers [6], [7]. While some schedulers have lower complexity ($O(\log N)$), where $N$ is the number of ports in the switch, they still suffer from delays that grow with $N$ [8]–[10].

To provide good performance, while addressing the complexity issue of scheduling algorithms, one approach is to add limited buffers at each crosspoint inside the crossbar. With today's ASIC technology, this can be implemented in a single chip. This makes buffered crossbar switches an attractive solution compared to the traditional input-queued switch because the crosspoint buffers allow for simpler scheduling algorithms and better delay performance.

This work is supported in part by the National Science Foundation under Grant CNS-0435303, and also in part by the New York State Center for Advanced Technology in Telecommunications (CATT).

With a speedup of 2, the authors in [11] showed that a buffered crossbar can provide 100% throughput. In [12], the results were extended to variable size packets. The author in [13] proved that the speedup requirement can be reduced to $2 - 1/N$. However, without speedup, previous throughput results are only limited to uniform traffic loads. Under uniform traffic, it has been shown that longest-queue-first at the input port and round-robin at the output port (LQF-RR) guaranteed 100% throughput [14]. In [15], the authors proved that a simple round-robin scheduler at both input and output ports can provide 100% throughput under uniform traffic

It is of general interest to know the maximum throughput that a buffered crossbar switch, without speedup, can achieve under admissible traffic. In [16], the authors proposed a distributed scheduling algorithm and derived a relationship between throughput and the size of crosspoint buffers. However, to achieve 100% throughput, it needed an infinite buffer. To our knowledge, there is no scheme that can achieve 100% throughput for a *finite* crosspoint buffer *without speedup*. In this paper, we propose a Stable QUeue Input-output Scheduler with Hamiltonian walk (SQUISH) for buffered crossbar switches. We assume a buffered crossbar switch with finite crosspoint buffers and no speedup. With SQUISH, each input first makes decisions on which crosspoint to send a cell based on the buffer occupancy information. Each output simply chooses the longest available queue for service. Then a Hamiltonian walk is applied to escape from potentially bad states. The SQUISH algorithm, which has a complexity of $O(\log N)$, can be shown to achieve 100% throughput for any admissible traffic satisfying the strong law of large numbers (SLLN). We also include an alternative proof using the Lyapunov function technique; this is however a weaker result since it holds only for Bernoulli arrivals.

The rest of the paper is organized as follows. In section II, we briefly describe the buffered crossbar switch architecture. In section III, we introduce the fluid model of a buffered crossbar and prove that an approximate maximum weight scheduler is rate stable. In section IV, we propose the SQUISH scheduling algorithm and prove its stability. In Section V, we present simulation results. Finally, section VI concludes the paper.

## II. Buffered crossbar switch model

Figure 1 shows an $N \times N$ buffered crossbar switch. We shall assume fixed size packet (cell) switching. Variable size packet

1-4244-1206-4/07/$25.00 ©2007 IEEE

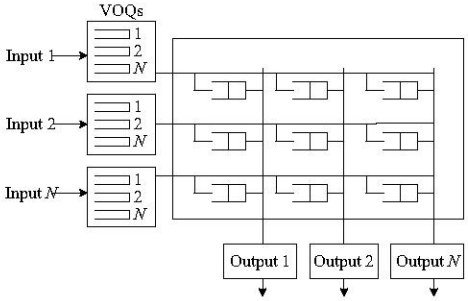


Fig. 1: Buffered crossbar switch.

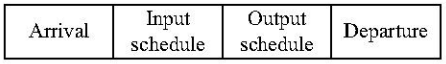


Fig. 2: The scheduling phases for buffered crossbar switch.

switching can be implemented by introducing packet segmentation and reassembly. To prevent head-of-line blocking, the inputs use virtual output queues as well. Each input maintains $N$ VOQs, one for each output. Let $Q_{ij}(n)$ denote the queue length of $VOQ_{ij}$ at time $n$, $n = 0, 1, 2, \ldots$. Each crosspoint contains a finite buffer of size $B$. The buffer between input $i$ and output $j$ is denoted as $CB_{ij}$; $B_{ij}(n)$ is the buffer occupancy of $CB_{ij}$ at time $n$, $B_{ij}(n) \leq B$.

The key role of crosspoint buffers is to separate the input contention from the output contention. This results in a two-stage scheduling scheme. As shown in Figure 2, in the input scheduling phase, each input determines which cell is transferred from a VOQ to the corresponding crosspoint buffer with available space. In the output scheduling phase, each output determines from which non-empty crosspoint buffer to serve a cell. When a crosspoint buffer is full, no more cells can be transferred to it. Note that if the crosspoint buffer size is unlimited, the buffered crossbar is equivalent to output queuing and input schedulers are not necessary, because packets can directly go to crosspoint buffers without buffering at inputs. For a practical single-chip implementation using current technology, however, the crosspoint buffers are constrained to a small number.

Let $A_{ij}(n)$ be the number of packets that have arrived at input $i$ destined for output $j$ up to time slot $n$. Assume the arrival process $\{A_{ij}(n), i, j = 1, \ldots, N\}$ satisfies the *strong law of large numbers* (SLLN), i.e., with probability one,

$$\lim_{n\to\infty} \frac{A_{ij}(n)}{n} = \lambda_{ij}, \quad i, j = 1, \ldots, N. \tag{1}$$

*Definition 1:* An arrival process is said to be *admissible* if

$$\sum_i \lambda_{ij} \leq 1, \ \sum_j \lambda_{ij} \leq 1. \tag{2}$$

Let $D_{ij}(n)$ be the number of departures from crosspoint buffer $CB_{ij}$ up to time slot $n$.

*Definition 2:* A switch operating under a matching algorithm is *rate stable* if, with probability one,

$$\lim_{n\to\infty} \frac{D_{ij}(n)}{n} = \lambda_{ij}, \quad i, j = 1, \ldots, N \tag{3}$$

for any arrival process satisfying condition (1).

Let $X_{ij}(n)$ be the total number of cells in $VOQ_{ij}$ and $CB_{ij}$ at time $n$, i.e., $X_{ij}(n) = Q_{ij}(n) + B_{ij}(n)$, $\mathbf{X}(n) = [X_{ij}(n)]$. Let $\mathbf{S}(n) = [\mathbf{S^I}(n); \mathbf{S^O}(n)]$ be the schedule at time $n$. $\mathbf{S^I}(n) = [S^I_{ij}(n)]$ is the *input schedule* and is subject to the following constraints:

$$\sum_j S^I_{ij}(n) \leq 1, S^I_{ij}(n) = 0 \text{ if } B_{ij}(n) = B; \tag{4}$$

$\mathbf{S^O}(n) = [S^O_{ij}(n)]$ is the *output schedule* and is subject to the following constraints:

$$\sum_i S^O_{ij}(n) \leq 1, S^O_{ij}(n) = 0 \text{ if } B_{ij}(n) = 0. \tag{5}$$

The set of all possible schedules is denoted by $\mathbf{\Pi}$. For each schedule $\mathbf{S} \in \mathbf{\Pi}$, define the weight $W_{\mathbf{S}}(n)$ of a schedule as

$$W_{\mathbf{S}}(n) = \left\langle \mathbf{S^O}(n), \mathbf{X}(n) \right\rangle,$$

where for two matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size, $\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{ij} A_{ij} B_{ij}$.

Note that although the weight is calculated with the output schedule only, in fact, the input schedule comes into the weight calculation implicitly. Indeed, from Figure 2, we can see that the output schedule is performed after the input schedule. A valid output schedule is determined by the state of crosspoint buffers. This takes place after the input scheduling phase, when the state of crosspoint buffers is updated.

## III. Fluid model results

Next, by using the fluid model introduced in [4], we describe a corresponding fluid model for a buffered crossbar switch. At each time slot, the switch employs a schedule $\mathbf{S}$. For a $\mathbf{S} \in \mathbf{\Pi}$, let $T_{\mathbf{S}}(n)$ be the cumulative amount of time that schedule $\mathbf{S}$ has been used by time slot $n$. Then we have the following equations governing the switch:

$$\begin{aligned} X_{ij}(n) &= X_{ij}(0) + A_{ij}(n) - D_{ij}(n), \\ D_{ij}(n) &= \sum_{\mathbf{S}\in\mathbf{\Pi}} \sum_{k=1}^{n} S^O_{ij}(T_{\mathbf{S}}(k) - T_{\mathbf{S}}(k-1)), \\ \sum_{\mathbf{S}\in\mathbf{\Pi}} T_{\mathbf{S}}(n) &= n. \end{aligned}$$

The first equation states that the number of cells in $X_{ij}$ backlogged in the switch equals the total number of arrivals minus departures. The second equation expresses the number of departures from $CB_{ij}$ in terms of the schedule that serves the buffer $CB_{ij}$. The third equation simply states that at each time slot, exactly one of the possible schedules is used.

In a manner similar to [4], the fluid model of a switch is governed by the following set of equations:

$$\begin{aligned} \bar{X}_{ij}(t) &= \bar{X}_{ij}(0) + \lambda_{ij} t - \bar{D}_{ij}(t), \\ \dot{\bar{D}}_{ij}(t) &= \sum_{\mathbf{S}\in\mathbf{\Pi}} S^O_{ij} \dot{\bar{T}}_{\mathbf{S}}(t), \\ \sum_{\mathbf{S}\in\mathbf{\Pi}} \bar{T}_{\mathbf{S}}(t) &= t. \end{aligned} \tag{6}$$

where each solution $(\bar{D}, \bar{T}, \bar{X})$ is called the fluid limit. The fluid limit can be obtained through a limiting procedure described next. First, extend the definition of $X_{ij}(t)$ for any $t \in (n, n+1)$,

$$X_{ij}(t) = X_{ij}(n) + (t-n)(X_{ij}(n+1) - X_{ij}(n)).$$

Define

$$\bar{X}^r_{ij}(t) = \frac{1}{r} X_{ij}(rt), r > 0.$$

Then, for each sequence $\{r_n\}$, there exists a subsequence $\{r_{n_k}\}$ and a continuous function $\bar{X}_{ij}(t)$ such that, for any $t > 0$,

$$\lim_{k\to\infty} \sup_{0\le t'\le t} \left| \bar{X}^{r_{n_k}}_{ij}(t') - \bar{X}_{ij}(t') \right| = 0.$$

That is,

$$\bar{X}_{ij}(t) = \lim_{r\to\infty} \frac{X_{ij}(rt)}{r}.$$

Any function $\bar{X}(t)$ obtained through the limiting procedure is said to be a *fluid limit* of the switch, and each fluid limit $(\bar{D}, \bar{T}, \bar{X})$ satisfies the fluid model equation (6).

For a buffered crossbar switch, we define the Input-Queued Switch Maximum-weight-matching Emulation (IQSME) schedule $\mathbf{S}^*$ as the solution of the following optimization problem:

$$\begin{aligned} & max_{\mathbf{S}} \langle \mathbf{S}(n), \mathbf{X}(n) \rangle \\ & s.t. \sum_{i=1}^{N} S_{ij} \le 1, \sum_{j=1}^{N} S_{ij} \le 1. \end{aligned}$$

$S^*_{ij} = 1$ means $(i)$ if $B_{ij} < B$, then input $i$ transfer a cell to crosspoint buffer $CB_{ij}$ and output $j$ serves a cell from crosspoint buffer $CB_{ij}$; $(ii)$ if $B_{ij} = B$, then input $i$ stays idle and output $j$ serves a cell from crosspoint buffer $CB_{ij}$. $\mathbf{S}^*$ is essentially an emulation of the maximum weight matching schedule in an input-queued switch with no crosspoint buffers. Define the weight $W^*$ of $\mathbf{S}^*$ as

$$W^* = \langle \mathbf{S}^*(n), \mathbf{X}(n) \rangle .$$

We will now introduce the following lemma, which is a fluid model equivalent of Lemma 1 in [9]:

*Lemma 1:* For any admissible traffic satisfying (1), if the weight of a scheduling algorithm at each time slot is within a bounded constant value $C$ from the maximum weight, i.e.,

$$W_{\mathbf{S}}(n) \ge W^*(n) - C, \tag{7}$$

then this algorithm is rate stable.

*Proof:* Let $\mathbf{\Pi}'$ be the set of schedules such that $W_{\mathbf{S}}(n) \ge W^*(n) - C$, $\mathbf{S} \in \mathbf{\Pi}'$. Let $\mathbf{\Lambda}$ be the $N \times N$ matrix with entries $\lambda_{ij}$. Then, in addition to (6), we have another fluid equation

$$\sum_{\mathbf{S}\in\mathbf{\Pi}'} \bar{T}_{\mathbf{S}}(t) = t. \tag{8}$$

Let $(\bar{D}, \bar{T}, \bar{X})$ be a fluid model solution with $\bar{\mathbf{X}}(0) = \mathbf{0}$.

$$\begin{aligned} \left\langle \bar{\mathbf{X}}(t), \dot{\bar{\mathbf{D}}}(t) \right\rangle &= \left\langle \bar{\mathbf{X}}(t), \sum_{\mathbf{S}\in\mathbf{\Pi}'} \mathbf{S}^{\mathbf{O}} \dot{\bar{T}}_{\mathbf{S}}(t) \right\rangle \\ &= \sum_{\mathbf{S}\in\mathbf{\Pi}'} \left\langle \bar{\mathbf{X}}(t), \mathbf{S}^{\mathbf{O}} \dot{\bar{T}}_{\mathbf{S}}(t) \right\rangle \\ &= \sum_{\mathbf{S}\in\mathbf{\Pi}'} \bar{W}_{\mathbf{S}}(t) \dot{\bar{T}}_{\mathbf{S}}(t), \end{aligned} \tag{9}$$

where $\bar{W}_S(t) = \left\langle \bar{\mathbf{X}}(t), \mathbf{S}^{\mathbf{O}} \right\rangle$. From the fluid limit procedure,

$$\begin{aligned} \left\langle \bar{\mathbf{X}}(t), \mathbf{S}^{\mathbf{O}} \right\rangle &= \left\langle \lim_{r\to\infty} \frac{\mathbf{X}(rt)}{r}, \mathbf{S}^{\mathbf{O}} \right\rangle \\ &= \lim_{r\to\infty} \frac{1}{r} \left\langle \mathbf{X}(rt), \mathbf{S}^{\mathbf{O}} \right\rangle \\ &\ge \lim_{r\to\infty} \frac{1}{r} (\langle \mathbf{X}(rt), \mathbf{S}^* \rangle - C) \\ &= \left\langle \bar{\mathbf{X}}(t), \mathbf{S}^* \right\rangle . \end{aligned} \tag{10}$$

Therefore, following [4], in the fluid limit scale, we have

$$\bar{W}_{\mathbf{S}}(t) = \bar{W}^*(t).$$

Equation (9) becomes

$$\begin{aligned} \left\langle \bar{\mathbf{X}}(t), \dot{\bar{\mathbf{D}}}(t) \right\rangle &= \sum_{\mathbf{S}\in\mathbf{\Pi}'} \bar{W}^*(t) \dot{\bar{T}}_{\mathbf{S}}(t) \\ &= \bar{W}^*(t) \sum_{\mathbf{S}\in\mathbf{\Pi}'} \dot{\bar{T}}_{\mathbf{S}}(t) \\ &= \bar{W}^*(t). \end{aligned}$$

Thus,

$$\begin{aligned} \left\langle \bar{\mathbf{X}}(t), \dot{\bar{\mathbf{X}}}(t) \right\rangle &= \left\langle \bar{\mathbf{X}}(t), \mathbf{\Lambda} \right\rangle - \left\langle \bar{\mathbf{X}}(t), \dot{\bar{\mathbf{D}}}(t) \right\rangle \\ &= \left\langle \bar{\mathbf{X}}(t), \mathbf{\Lambda} \right\rangle - \bar{W}^*(t), \end{aligned} \tag{11}$$

where $\mathbf{\Lambda} = [\lambda_{ij}]$ is a doubly stochastic matrix. From the Birkhoff-von Neumann theorem, $\mathbf{\Lambda}$ can be written as a convex combination of permutation matrices $\mathbf{S}_k$, i.e., for $k = 1, 2, \ldots, K$, $\gamma_k > 0$ and $\sum_{k=1}^{K} \gamma_k \le 1$,

$$\mathbf{\Lambda} = \sum_{k=1}^{K} \gamma_k \mathbf{S}_k.$$

Thus, we have

$$\left\langle \bar{\mathbf{X}}(t), \mathbf{\Lambda} \right\rangle = \left\langle \bar{\mathbf{X}}(t), \sum_{k=1}^{K} \gamma_k \mathbf{S}_k \right\rangle = \sum_{k=1}^{K} \gamma_k \left\langle \bar{\mathbf{X}}(t), \mathbf{S}_k \right\rangle .$$

Since $\bar{W}^*(t) = \max \left\langle \bar{\mathbf{X}}(t), \mathbf{S}(t) \right\rangle$,

$$\bar{W}^*(t) \ge \sum_{k=1}^{K} \gamma_k \left\langle \bar{\mathbf{X}}(t), \mathbf{S}_k \right\rangle .$$

Therefore,

$$\left\langle \bar{\mathbf{X}}(t), \dot{\bar{\mathbf{X}}}(t) \right\rangle = \left\langle \bar{\mathbf{X}}(t), \mathbf{\Lambda} \right\rangle - \bar{W}^*(t) \le 0.$$

From Lemma 1 in [4], the fluid model is weakly stable and according to Theorem 3 in [4] the switch is rate stable. ■

## IV. The Stable Queue Input-output Scheduler with Hamiltonian walk

In this section, we present the Stable QUeue Input-output Scheduler with Hamiltonian walk (SQUISH). By using the Lemma proved in the previous section, we prove that SQUISH can achieve 100% throughput.

Like we did for IQSME, we define the Input-Queued Switch Hamiltonian Walk Matching Emulation (IQSHWME) schedule $\mathbf{H} = [H_{ij}]$ for buffered crossbar switches as an emulation of the matching generated by a Hamiltonian walk for input-queued switches [9]. For an input-queued switch, there are $N!$ distinct matchings. A Hamiltonian walk $\mathbf{H}(n)$ visits each of the $N!$ distinct matchings exactly once during times $n = 1, 2, \ldots, N!$. For $n > N!$, $\mathbf{H}(n) = \mathbf{H}(n \bmod N!)$. A Hamiltonian walk can be generated with a simple algorithm with a time complexity of $O(1)$ [17]. The weight $W_{\mathrm{H}}$ of $\mathbf{H}$ is defined as

$$W_{\mathrm{H}} = \langle \mathbf{H}, \mathbf{X} \rangle .$$

Next, we define the SQUISH algorithm for a buffered crossbar switch with a crosspoint buffer size of $B$, for any $B > 0$.

At time $n+1$, a new possible schedule $\mathbf{M}(n+1) = [M_{ij}^I(n+1), M_{ij}^O(n+1)]$ is generated as follows:

*Input schedule:* For each input $i$, if $S_{ij}^I(n) = 1$, $B_{ij}(n+1) = 0$ and $Q_{ij}(n+1) > 0$, then $M_{ij}^I(n+1) = 1$. Otherwise, among the VOQs with non-full crosspoint buffers, pick the longest VOQ for service.

*Output schedule:* At each time slot, each output $j$ serves the non-empty crosspoint buffer corresponding to the largest $X_{ij}$. This determines $M_{ij}^O(n+1)$.

Then the schedule $\mathbf{S}(n+1)$ used by SQUISH at time $n+1$ is:

$$\mathbf{S}(n+1) = \arg\max_{\mathbf{S} \in \{\mathbf{M}(n+1), \mathbf{H}(n+1)\}} W_{\mathbf{S}}(n+1). \quad (12)$$

We will use an example to illustrate SQUISH operation. Consider a $3 \times 3$ buffered crossbar switch with a crosspoint buffer size of 4. At time slot $n+1$, assume the states of $VOQs$ and crosspoint buffers are

$$\begin{bmatrix} 3 & 5 & 4 \\ 4 & 2 & 4 \\ 3 & 5 & 4 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 4 \\ 1 & 3 & 4 \\ 2 & 0 & 3 \end{bmatrix},$$

respectively. We also assume that the input schedule $\mathbf{S}^{\mathrm{I}}(n)$ used at time slot $n$ is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Then according to SQUISH, $\mathbf{M}^{\mathrm{I}}(n+1)$ and $\mathbf{M}^{\mathrm{O}}(n+1)$ are

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

respectively. The weight of $\mathbf{M}(n+1)$ is $4(VOQ_{21}) + 1(CB_{21}) + 2(VOQ_{22}) + 3(CB_{22}) + 4(VOQ_{13}) + 4(CB_{13}) =$

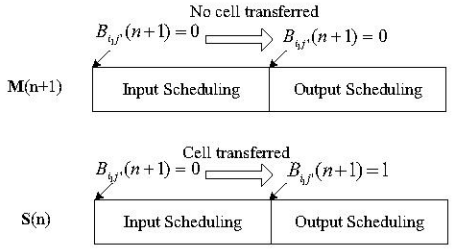


Fig. 3: Cross-point buffer $CB_{i_1 j'}$ state under schedules $\mathbf{M}(n+1)$ and $\mathbf{S}(n)$ in time slot $n+1$.

18. Assume the Hamiltonian walk $\mathbf{H}(n+1)$ is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Then the weight of Hamiltonian walk is $3(VOQ_{11}) + 5(VOQ_{32}) + 4(VOQ_{23}) + 4(CB_{23}) = 16$. Since $W_{\mathbf{M}(n+1)} > W_{\mathrm{H}(n+1)}$, $\mathbf{M}(n+1)$ is used as the schedule at time $n+1$.

*Theorem 1:* The SQUISH scheduling algorithm is rate stable if the input traffic is admissible and satisfies (1).

First, we define how a schedule $\mathbf{S}(n-p)$ (used at time slot $(n-p)$, $p > 0$) is applied at time $n$. Note that the schedule at $(n-p)$ may not be a feasible schedule for time $n$, i.e., if the crosspoint buffer is full, then the input cannot send a cell to it, or if a crosspoint buffer is empty and the output cannot serve it. Therefore, we set $S_{ij}^I(n-p)$ to be 0 if the crosspoint buffer $CB_{ij}$ is full. Similarly, $S_{ij}^O(n-p)$ is set to be 0 if the crosspoint buffer $CB_{ij}$ is empty. We denote the schedule $\mathbf{S}(n-p)$ after this transformation as $\tilde{\mathbf{S}}(n-p)$. Then we can define the weight of schedule $\mathbf{S}(n-p)$ at time $n$ as

$$W_{\mathbf{S}(n-p)}(n) = \left\langle \tilde{\mathbf{S}}^{\mathbf{O}}(n-p), X \right\rangle.$$

With the above definition, we prove the following Lemma:

*Lemma 2:* Let $\mathbf{S}(n)$ and $\mathbf{S}(n+1)$ be the schedules used at time $n$ and $n+1$ respectively, then $W_{\mathbf{S}(n+1)}(n+1) \geq W_{\mathbf{S}(n)}(n+1)$.

This Lemma means that at time slot $n+1$, the schedule generated by SQUISH at time slot $n+1$ has a larger weight than the schedule used in the previous time slot $n$.

*Proof:* We prove this by contradiction. Assume

$$W_{\mathbf{S}(n+1)}(n+1) < W_{\mathbf{S}(n)}(n+1).$$

Then from the definition of the SQUISH algorithm,

$$W_{\mathbf{M}(n+1)}(n+1) < W_{\mathbf{S}(n)}(n+1),$$

where $W_{\mathbf{M}(n+1)}(n+1) = \sum_{j=1}^{N} \sum_{i=1}^{N} M_{ij}^O(n+1) X_{ij}(n+1)$ and $W_{\mathbf{S}(n)}(n+1) = \sum_{j=1}^{N} \sum_{i=1}^{N} \tilde{S}_{ij}^O(n) X_{ij}(n+1)$.

This is only possible when there exists at least one output $j'$ under schedule $\mathbf{S}(n)$, such that output $j'$ serves crosspoint buffer $CB_{i_1 j'}$, while with schedule $\mathbf{M}(n+1)$ output $j'$ serves a different crosspoint buffer $CB_{i_2 j'}$ ($i_1 \neq i_2$), and

$$X_{i_1 j'}(n+1) > X_{i_2 j'}(n+1).$$

Since with $\mathbf{M}(n+1)$, output $j'$ will serve the non-empty crosspoint buffer corresponding to the largest $X_{ij'}$, $i = 1, 2, \ldots, N$, it means that with $\mathbf{M}(n+1)$, at the output scheduling phase, $B_{i_1 j'}(n+1) = 0$ (as shown in Fig. 3). This implies that at the beginning of time slot $n+1$, $B_{i_1 j'}(n+1) = 0$ and with $\mathbf{M}(n+1)$, input $i_1$ does not send a cell to $CB_{i_1 j'}$, that is $M^I_{i_1 j'}(n+1) = 0$ (Fig. 3).

Note that at the beginning of time slot $n+1$, both $\mathbf{S}(n)$ and $\mathbf{M}(n+1)$ have the same VOQs and crosspoint buffers states. Therefore, with $\mathbf{S}(n)$, at the beginning of time slot $n+1$, $B_{i_1 j'}(n+1) = 0$. However, with $\mathbf{S}(n)$, at the output scheduling phase, crosspoint buffer $CB_{i_1 j'}$ is served. This is only possible if with $\mathbf{S}(n)$, at the input scheduling phase, input $i_1$ sends a cell to $CB_{i_1 j'}$, that is $S^I_{i_1 j'}(n) = 1$ and $Q_{i_1 j'}(n+1) > 0$.

Therefore, with schedule $\mathbf{M}(n+1)$, at the input scheduling phase, $B_{i_1 j'}(n+1) = 0$, $S^I_{i_1 j'}(n) = 1$ and $Q_{i_1 j'}(n+1) > 0$. According to the definition of $\mathbf{M}(n+1)$, this implies that $M^I_{i_1 j'}(n+1) = 1$ and we have a contradiction. Therefore, we conclude that

$$W_{\mathbf{S}(n+1)}(n+1) \geq W_{\mathbf{S}(n)}(n+1).$$

■

Now, we are ready to prove Theorem 1.

*Proof:* From Lemma 1, to prove that the algorithm is rate stable, it suffices to show that

$$W_{\mathbf{S}}(n) \geq W^*(n) - C,$$

where $C$ is a bounded constant. Since there is at most one arrival or departure from each $X_{ij}$ in each time slot, we obtain that for any schedule $\mathbf{P}$

$$W_{\mathbf{P}}(n) \geq W_{\mathbf{P}}(n+k) - kN. \tag{13}$$

Consider any given time $T$, let $\mathbf{S}^*$ denote the IQSME at time $T$. By the property of Hamiltonian walk, there exists a $n' \in [T - N!, T]$, such that the IQSHWME at $n'$ is the IQSME at $T$, i.e., $\mathbf{H}(n') = \mathbf{S}^*$. Thus we have

$$\begin{aligned} W_{\mathbf{S}(n')}(n') &\geq W_{\mathbf{H}(n')}(n') \\ &= W_{\mathbf{S}^*}(n') \\ &\geq W_{\mathbf{S}^*}(T) - (T - n')N, \end{aligned} \tag{14}$$

where the first inequality is from the definition of the SQUISH algorithm, and the last inequality is from (13).

Again, from Lemma 2 and (13),

$$W_{\mathbf{S}(T)}(T) \geq W_{\mathbf{S}(T-1)}(T) \geq W_{\mathbf{S}(T-1)}(T-1) - N.$$

Using this repeatedly, we obtain

$$W_{\mathbf{S}(T)}(T) \geq W_{\mathbf{S}(n')}(n') - (T - n')N. \tag{15}$$

Combining (14) and (15), we have

$$W_{\mathbf{S}(T)}(T) \geq W_{\mathbf{S}^*}(T) - 2(T - n')N.$$

Since $T - n' \leq N!$, this implies

$$W_{\mathbf{S}(T)}(T) \geq W_{\mathbf{S}^*}(T) - 2NN!,$$

and it is true for any $T$. From Lemma 1, the SQUISH algorithm is therefore rate stable. ■

In SQUISH, a centralized scheduler is needed to compute the weights of the schedules generated by $\mathbf{M}(n)$ and the Hamiltonian walk, respectively. Therefore, the lengths of up to two queues per port need to be sent to the scheduler. The scheduler will add up the queue lengths for each schedule, with time complexity $O(\log N)$. The scheduler then compares the two weights and selects the schedule with larger weight. The time complexity of generating the next vertex in a Hamiltonian walk is $O(1)$, and the time complexity to find the longest queue is $O(\log N)$. Therefore, the complexity of SQUISH is $O(\log N)$.

SQUISH can be implemented in a pipelined manner to reduce the complexity and/or address the issue of latency between the line cards and scheduler. With a pipeline depth $p$, at time slot $n$, the queue length information is sent to the scheduler. After $p$ time slots, a new schedule is obtained and used at time $n + p$. We can prove that the pipelined SQUISH algorithm can also achieve 100% throughput using an approach similar to the proof of Theorem 1. The proof is not included here for the sake of brevity. Finally, we include in the Appendix an alternative proof for the case of Bernoulli arrivals using the Lyapunov technique.

## V. Performance Studies

In this section, we study the delay performance of the SQUISH scheduling algorithm via simulations. The traffic patterns studied in this section are uniform and non-uniform traffic with Bernoulli and bursty arrivals. For bursty traffic, all the cells in the same burst go to the same destination. The burst lengths are chosen independently from the following truncated Pareto distribution:

$$P(\text{A burst has a length } l) = \frac{c}{l^\alpha}, l = 1, \ldots, 10000,$$

where $\alpha$ is the Pareto parameter and $c$ is the normalization constant; we vary $\alpha$ to get different average burst lengths. In our simulations, all inputs are equally loaded on a normalized scale $\rho \in (0, 1)$, and we measure the fixed length packet (cell) delay.

### A. Uniform Bernoulli and bursty traffic

For uniform traffic, the destination of a new cell (burst) is uniformly distributed among all the output ports, i.e., $\lambda_{ij} = \rho/N$. Figure 4 shows the delay performance under uniform Bernoulli and bursty traffic with different Pareto parameters. When $\alpha = 1.9$, the average burst length is about 9. When $\alpha = 1.7$, the average burst length is about 24. We can see that SQUISH has delay performance very close to an output-queued switch. Indeed, even a simple round-robin scheduler has a delay performance close to an output queued switch [18]. However, if the traffic is nonuniform, the round-robin scheduler fails to achieve 100% throughput. We therefore study the delay performance of SQUISH under nonuniform traffic next.

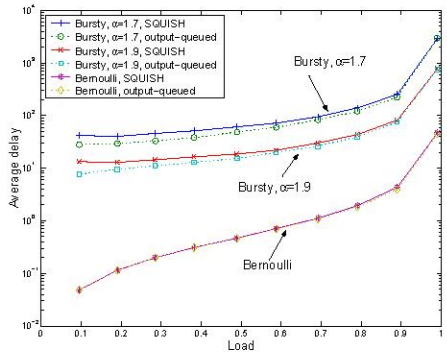


Fig. 4: Average delay under uniform Bernoulli and bursty traffic, $N = 32$, crosspoint buffer size $B = 1$.

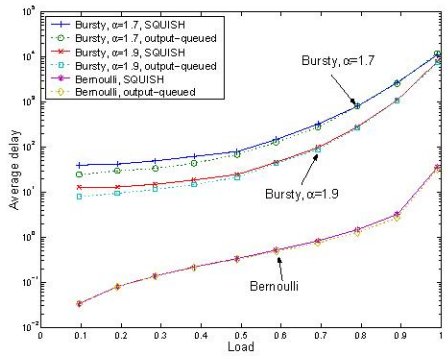


Fig. 5: Average delay under log-diagonal traffic, $N = 32$, $B = 1$.

## B. Nonuniform traffic

We use the following traffic patterns to test the performance of SQUISH.

- Log-diagonal [19]: arrival rates at the same input differ exponentially, i.e., $\lambda_{i(i+j)} = 2\lambda_{i(i+j+1)}$, where $0 \leq j \leq N-2$.
- Lin-diagonal [19]: arrival rates at the same input differ linearly, i.e., $\lambda_{i(i+j)} - \lambda_{i(i+j+1)} = 2\rho/N(N+1)$, where $0 \leq j \leq N-2$.
- Hot-spot [18]: For any input port, $\lambda_{ii} = w\rho$, $\lambda_{ij} = (1-w)\rho/(N-1)$, for $i \neq j$. By changing the factor $w$, we can get different nonuniform traffic patterns.

Figures 5 and 6 shows the delay performance for log-diagonal and lin-diagonal traffic. We can see that the average delay performance for all traffic patterns is close to an output-queued switch.

Figure 7 shows the delay performance for Bernoulli traffic with $w = 0.25$, 0.5 and 0.75. For the hot-spot bursty traffic, $\alpha = 1.9$ and $w = 0.5$. We can see that for different $w$, the average delays are always close to that of an output-queued switch. Note that under hotspot traffic, the round-robin scheduler has a throughput around 85% [18] and the SBF-LBF [20] has a throughput around 87%.

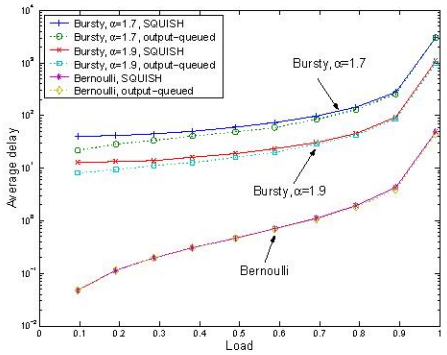


Fig. 6: Average delay under lin-diagonal traffic, $N = 32$, $B = 1$.

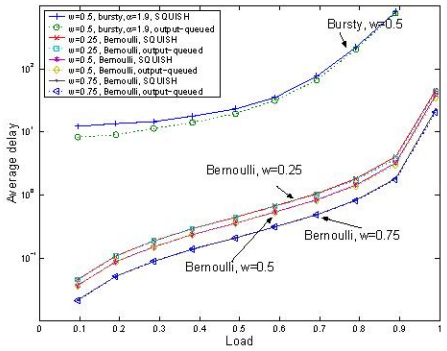


Fig. 7: Average delay under hotspot traffic with different values of $w$, $N = 32$, $B = 1$.

## C. Switch size effect

Generally, for input-queued switches, the average delay increases linearly with the switch size. For output-queued switches, the delay is independent of the switch size. In this simulation, we test the delay performance with different switch sizes. Figure 8 shows the average delay with different switch sizes. For a given switch size, we simulated uniform Bernoulli, nonuniform Bernoulli ($w = 0.5$) and uniform bursty traffic ($\alpha = 1.9$). We can see that for Bernoulli traffic, the average delays are almost the same for different switch sizes. For bursty traffic, the average delay does not increase with the switch size.

## D. Buffer size effect

Note that when the crosspoint buffer size is infinite, a buffered crossbar switch would become equivalent to an output-queued switch. Therefore, if we increase the crosspoint buffer size, the average delay will decrease, and eventually converge to the average delay of an output-queued switch. In our previous simulations, we can see with a crosspoint buffer size of one, the average delay is already very close to an output-queued switch. Therefore, increasing the crosspoint buffer size will only lead to a marginal delay improvement. Next, we report the simulation results for a very skewed loading-diagonal traffic. With diagonal traffic, $\lambda_{ii} = \rho/2$, $\lambda_{ij} = \rho/2$, for $j = (i+1)\ mod\ N$. Under diagonal traffic,

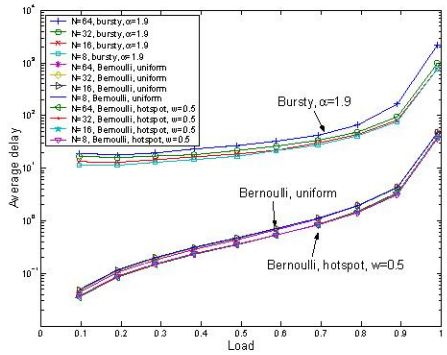


Fig. 8: Average delay with different switch sizes under uniform Bernoulli, hot-spot Bernoulli and uniform bursty traffic, $B = 1$.

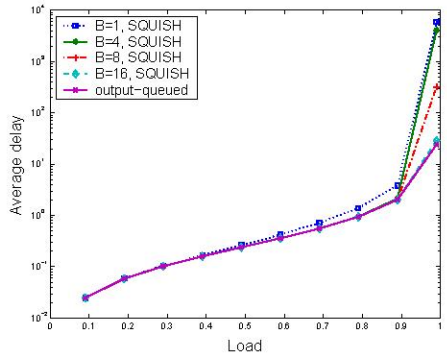


Fig. 9: Average delay under diagonal traffic with different crosspoint buffer sizes, $N = 32$.

for output $j$, only crosspoint buffer $CB_{jj}$ and $CB_{(j-1)j}$ will receive packets, the remaining crosspoint buffers are not utilized.

Figure 9 shows the average delay under diagonal traffic with different crosspoint buffer sizes. We can see under diagonal traffic, when the loading is 0.99, and with a crosspoint buffer size of 1, the average delay is high as compared to the output-queued switch. However, if we increase the crosspoint buffer size, i.e., when $B = 16$, the delay drops to a number very close to that of an output-queued switch. For switch size $N = 64$, the delays are almost the same as for $N = 32$ and we get similar results.

## VI. Conclusion

In this paper, we have proposed a Stable Queue Input-output Scheduler with Hamiltonian walk (SQUISH) for buffered crossbar switches with a crosspoint buffer size as small as one and no speedup. The complexity of SQUISH is $O(\log N)$. With SQUISH, a buffered crossbar switch can achieve 100% throughput under any admissible input traffic satisfying the strong law of large numbers. Our simulation studies indicate that for most realistic traffic scenarios the delay performance of SQUISH is close to that of an ideal output-queued switch.

## References


[1] M. J. Karol, M. Hluchyj, and S. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Trans. on Communications*, vol. 35, pp. 1347–1356, 1987.

[2] 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*, vol. 37, no. 12, pp. 1936–1949, December 1992.

[3] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Trans. Communications*, vol. 47, no. 8, pp. 1260–1267, Aug 1999.

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

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

[6] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. on Networking*, vol. 7, pp. 188–201, April 1999.

[7] Y. Li, S. Panwar, and H. J. Chao, "On the performance of a dual round-robin switch," in *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

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

[9] P. Giaccone, B. Prabhakar, and D. Shah, "Toward simple, high-performance schedulers for high-aggregate bandwidth switches," in *Proceedings of IEEE INFOCOM*, New York, 2002.

[10] Y. Li, S. S. Panwar, and H.J. Chao, "Exhaustive service matching algorithms for input queued switches," in *Proceedings of IEEE Workshop on High Performance Switching and Routing*, 2004.

[11] S-T. Chuang, S. Iyer, and N. McKeown, "Practical algorithms for performance guarantees in buffered crossbars," in *Proceedings of IEEE Infocom*, Miami FL, March 2005.

[12] J. Turner, "Strong performance guarantees for asynchronous crossbar schedulers," in *Proceedings of IEEE Infocom*, Spain, April 2006.

[13] M. Berger, "Delivering 100% throughput in a buffered crossbar with round robin scheduling," in *Proceedings of IEEE High Performance Switch and Routing*, Poznan, Poland, 2006.

[14] T. Javidi, R. Magill, and T. Hrabik, "A high throughput scheduling algorithm for a buffered crossbar switch fabric," in *Proceedings of IEEE International Conference on Communications*, 2001.

[15] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the combined input-crosspoint buffered packet switch with round-robin arbitration," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1945–1951, November 2005.

[16] P. Giaccone, E. Leonardi, and D. Shah, "On the maximal throughput of networks with finite buffers and its application to buffered crossbars," in *Proceedings of IEEE Infocom*, Miami, FL, March 2005.

[17] Nijenhuis A. and Wilf H., *Combinatorial algorithms: for computers and calculators*, Academic Press, 1978.

[18] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: Combined Input-One-Cell-Crosspoint Buffered Switch," in *Proceedings of IEEE Workshop of High Performance Switches and Routers 2001*, Dallas, TX, May 2001.

[19] A. Baranowska, G. Danilewicz, W. Kabacinski, J. Kleban, D. Parniewiez, and P. Dabrowski, "Performance evaluation of the multiple output queueing switch under different traffic patterns," in *Proceedings of IEEE GLOBECOM 2005*, Dec 2005.

[20] L. Mhamdi and M. Hamdi, "MCBF: a high-performance scheduling algorithm for buffered crossbar switches," *IEEE Communications Letters*, vol. 7, no. 9, pp. 451–453, Sept 2003.

[21] P. R. Kumar and S. P. Meyn, "Stability of queueing networks and scheduling policies," *IEEE Transactions on Automatic Control*, vol. 40, no. 2, Feb 1995.


## Appendix

In this appendix, we show the stability of SQUISH by applying the Lyapunov technique. Let $a_{ij}(n)$ denote the number of arrivals to $VOQ_{ij}$ in time slot $n$, $a_{ij}(n) \in \{0, 1\}$. The arrival is a Bernoulli process with a rate of $\lambda_{ij}$ and $\sum_j \lambda_{ij} < 1$,

$\sum_i \lambda_{ij} < 1$. [1] As before, define $X_{ij}(n) = Q_{ij}(n) + B_{ij}(n)$, which is the total number of cells in $VOQ_{ij}$ and cross-point buffer $CB_{ij}$. Note that $S^O_{ij}(n)$ denotes the departure from cross-point buffer $CB_{ij}$ in time slot $n$, where $S^O_{ij}(n)$ equals one if cross-point buffer is served by output $j$ and zero otherwise. Then the evolution of the queue length is as follows:

$$X_{ij}(n+1) = X_{ij}(n) + a_{ij}(n+1) - S^O_{ij}(n),\ i,j = 1,2,\ldots,N. \tag{16}$$

The system is considered stable if the expected queue length is bounded. In other words, the system is stable if $\sup\{E[\mathbf{X}(n)]\} < \infty$.

The stability can be proved by the technique using the *Lyapunov function*. A Lyapunov function is a scalar function that is continuous, positive definite ($V(\mathbf{X}) > 0$, for any $\mathbf{X} \neq 0$), and has continuous first-order partial derivatives. Usually, a Lyapunov function takes a quadratic form $V(\mathbf{X}(n)) = \mathbf{X}^T(n)\mathbf{X}(n)$, where $\mathbf{X}^T(n)$ denotes the transpose of vector $\mathbf{X}(n)$.

To prove stability, from [21], it is sufficient to show that there is a negative expected single-step drift of the Lyapunov function. In other words,

$$E[\mathbf{X}^T(n+1)\mathbf{X}(n+1) - \mathbf{X}(n)^T\mathbf{X}(n) | \mathbf{X}(n)] < -\epsilon||\mathbf{X}(n)|| + k.$$

where $\epsilon > 0$, $k > 0$ and $||\mathbf{X}(n)|| = \sum_{i,j} X_{ij}(n)$.

The expression on the left hand side is the Lyapunov drift, representing the expected change in the Lyapunov function from one slot to the next. The above condition ensures that the Lyapunov drift is negative whenever the sum of queue lengths is sufficiently large. Intuitively, this property ensures switch stability because whenever the total queue lengths is beyond some value, there is a negative drift of the total backlog and the negative drift eventually drives it back to the bounded region.

With the Lyapunov method, it was proved in [2], [3] that for an input-queued switch, the maximum weight matching (MWM) algorithm is stable for any Bernoulli admissible traffic. In [9], the stability results were extended to a larger class of scheduling algorithms, which have a weight greater than the weight of the MWM algorithm minus a constant. Similarly, with the defined IQSME for a buffered crossbar switch, if the weight of a scheduling algorithm is at most a constant away from the weight of the IQSME, we can obtain the same stability result, as stated in the following lemma.

*Lemma 1* (following [9]): Let $W_{\mathbf{S}(n)}$ denote the weight of the schedule $\mathbf{S}$ at time $n$, and let $W^*(n)$ be the weight of IQSME. If there exists a positive constant $C$ such that

$$W_{\mathbf{S}(n)} \geq W^*(n) - C$$

holds for all $n$, then the algorithm $\mathbf{S}$ is stable for any Bernoulli admissible traffic.

*Proof:* Define a Lyapunov function $V(\mathbf{X}(n)) = \mathbf{X}^T(n)\mathbf{X}(n)$. Then

$$\begin{aligned} V(\mathbf{X}(n+1)) - V(\mathbf{X}(n)) &= \sum_{i,j}[X^2_{ij}(n+1) - X^2_{ij}(n)] \\ &= \sum_{i,j}[X_{ij}(n+1) - X_{ij}(n)][X_{ij}(n+1) + X_{ij}(n)] \end{aligned}$$

From equation (16), we obtain

$$\begin{aligned} &V(\mathbf{X}(n+1)) - V(\mathbf{X}(n)) \\ &= \sum_{i,j}[a_{ij}(n+1) - S^O_{ij}(n)][2X_{ij}(n) + a_{ij}(n+1) - S^O_{ij}(n)] \\ &= \sum_{i,j}[a_{ij}(n+1) - S^O_{ij}(n)][2X_{ij}(n)] + [a_{ij}(n+1) - S^O_{ij}(n)]^2 \end{aligned}$$

Since there are at most $N$ unit elements within both vector $\mathbf{a}(n+1)$ and $\mathbf{S}^O(n)$, $\sum_{i,j}[a_{ij}(n+1) - S^O_{ij}(n)]^2 \leq 2N$, thus we have

$$\begin{aligned} &V(\mathbf{X}(n+1)) - V(\mathbf{X}(n)) \\ &\leq \sum_{i,j}[a_{ij}(n+1) - S^O_{ij}(n)][2X_{ij}(n)] + 2N. \end{aligned}$$

Taking conditional expectation with respect to $\mathbf{X}(n)$,

$$\begin{aligned} &E[V(\mathbf{X}(n+1)) - V(\mathbf{X}(n))|\mathbf{X}(n)] \\ &\leq 2\sum_{i,j} X_{ij}(n)[\lambda_{ij} - S^O_{ij}(n)] + 2N \\ &= 2\sum_{i,j} X_{ij}(n)\lambda_{ij} - 2W_{\mathbf{S}(n)} + 2N \\ &\leq 2\sum_{i,j} X_{ij}(n)\lambda_{ij} - 2W^* + 2C + 2N. \end{aligned}$$

From the proof of Lemma 1,

$$\sum_{i,j} X_{ij}(n)\lambda_{ij} \leq \sum_k \gamma_k W^*,$$

where $\sum_k \gamma_k < 1$. Therefore,

$$\begin{aligned} &E[V(\mathbf{X}(n+1)) - V(\mathbf{X}(n))|\mathbf{X}(n)] \\ &\leq -2(1 - \sum_k \gamma_k)W^* + 2C + 2N \end{aligned}$$

Since $W^* \geq \frac{||\mathbf{X}(n)||}{N}$ [2], then

$$\begin{aligned} &E[V(\mathbf{X}(n+1)) - V(\mathbf{X}(n))|\mathbf{X}(n)] \\ &\leq -2\frac{(1 - \sum_k \gamma_k)}{N}||\mathbf{X}(n)|| + 2C + 2N. \end{aligned}$$

Let $\epsilon = 2\frac{(1-\sum_k \gamma_k)}{N}$ and $k = 2C + 2N$, we conclude that the algorithm $\mathbf{S}(n)$ is stable. ■

In the proofs of Lemma 2 and Theorem 1, we showed that

$$W_{\mathbf{S}(n)} \geq W^* - 2NN!,$$

where $\mathbf{S}(n)$ is the scheduling algorithm generated by SQUISH. Therefore, the weight of a schedule generated by SQUISH is within a constant from the weight of IQSME, and we conclude that SQUISH is stable for any admissible Bernoulli traffic.

[1] Note that the admissible traffic definition is different from the fluid approach.

[2] There are totally $N!$ matchings and each $X_{ij}$ is served in $(N-1)!$ matchings. Since $W^* = \max\langle \mathbf{S}^*, \mathbf{X}(n)\rangle$, $W^* \geq \frac{1}{N!}\sum_{k=1}^{N!}\langle \mathbf{S}_k, \mathbf{X}(n)\rangle = \frac{(N-1)!}{N!}||\mathbf{X}(n)|| = \frac{1}{N}||\mathbf{X}(n)||$.