Safe Randomized Load-Balanced Switching By Diffusing Extra Loads

SEN YANG, Georgia Institute of Technology, USA
BILL LIN, University of California, San Diego, USA
JUN XU, Georgia Institute of Technology, USA

Load-balanced switch architectures are known to be scalable in both size and speed, which is of interest due to the continued exponential growth in Internet traffic. However, the main drawback of load-balanced switches is that packets can depart out of order from the switch. Randomized load-balancing of application flows by means of hashing on the packet header is a well-known simple solution to this packet reordering problem in which all packets belonging to the same application flow are routed through the same intermediate port and hence the same path through the switch. Unfortunately, this method of load-balancing can lead to instability, depending on the mix of flow sizes and durations in the group of flows that gets randomly assigned to route through the same intermediate port. In this paper, we show that the randomized load-balancing of application flows can be enhanced to provably guarantee both stability and packet ordering by extending the approach with *safety* mechanisms that can *uniformly diffuse* packets across the switch whenever there is a build-up of packets waiting to route through some intermediate port. Although simple and intuitive, our experimental results show that our extended randomized load-balancing approach outperforms existing load-balanced switch architectures.

CCS Concepts: • **Networks → Packet scheduling**; *Routers*; *Bridges and switches*; *Packet-switching networks*; • **Mathematics of computing** → Queueing theory;

Additional Key Words and Phrases: Load-balanced switches; packet reordering; throughput guarantees; low latency

ACM Reference Format:
Sen Yang, Bill Lin, and Jun Xu. 2017. Safe Randomized Load-Balanced Switching By Diffusing Extra Loads. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 2, Article 29 (December 2017), 37 pages. https://doi.org/10.1145/3154487

1 INTRODUCTION

Internet traffic continues to grow exponentially, fueled by an increasing adoption of cloud computing and video streaming and by an explosion of network-connected devices with increasing access speeds. To keep up with the relentless traffic growth with reliable service, network operators need high-performance switch architectures that can scale well in both size and speed, provide throughput guarantees, achieve low latency, and maintain packet ordering. However, conventional switch architectures like centrally-scheduled input-queued crossbar switches are not scalable.

Authors' addresses: Sen Yang, Georgia Institute of Technology, School of Electrical and Computer Engineering, Atlanta, GA, USA, sen.yang@gatech.edu; Bill Lin, University of California, San Diego, Department of Electrical and Computer Engineering, La Jolla, CA, USA, billlin@ece.ucsd.edu; Jun Xu, Georgia Institute of Technology, School of Computer Science, Atlanta, GA, USA, jx@cc.gatech.edu.

2476-1249/2017/12-ART29 $15.00
https://doi.org/10.1145/3154487

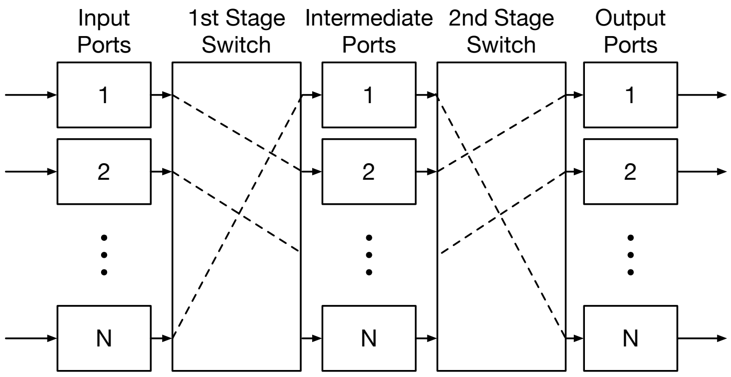


Fig. 1. Generic load-balanced switch.

Alternatively, a promising scalable class of switch architecture is the load-balanced switch, which was first introduced by Chang et al. [6, 7], and later further developed by others (e.g. [10, 14, 16, 18, 27]). These architectures all build upon the idea of Valiant load-balancing [26] which predated the design of any IP switch or router. They rely on two switching stages for routing packets. Figure 1 shows a diagram of a generic two-stage load-balanced switch. The first switching stage connects the first stage of input ports to the center stage of intermediate ports, and the second switching stage connects the center stage of intermediate ports to the final stage of output ports. Both switching stages execute a deterministic connection pattern such that each input is connected to each output of a switching stage $\frac{1}{N}$th of the time. This can be implemented for example using two identical $N \times N$ crossbar switching stages where each switching stage goes through the same predetermined cyclic-shift connection pattern such that each input is connected to each output of a switching stage exactly once every N cycles (time slots). Alternatively, as shown in [16], the deterministic connection pattern can also be efficiently implemented using optics in which all inputs are connected to all outputs of a switching stage in parallel at a rate of $\frac{1}{N}$ the line rate.

Although the basic load-balanced switch originally proposed in [6] is capable of achieving throughput guarantees, it has the critical problem that packet departures can be badly out of order. In the basic load-balanced switch, consecutive packets at an input port are spread to all N intermediate ports upon arrival. Packets going through different intermediate ports may encounter *different queueing delays*. Thus, some of these packets may arrive at their output ports out-of-order. This is detrimental to Internet traffic since the widely used TCP transport protocol falsely regards out-of-order packets as indications of congestion and packet loss. Therefore, a number of researchers have explored this packet ordering problem.

One simple approach for ensuring packet ordering, called Application Flow-Based Routing (AFBR) algorithm [16], is based on the following insight: To prevent harmful effects in TCP performance due to out-of-order packets, only packets belonging to the same application flow (e.g. a TCP/IP flow) have to depart from their output port in order. This can be achieved by forcing all packets that belong to the same application flow to go through the same intermediate port. In doing so, all packets belonging to the same application flow are guaranteed to take the same path through the switch, which avoids reordering among them. The selection of intermediate port can be easily achieved by hashing on the header field of every packet (source and destination IP addresses, source and destination ports, and protocol identification) to obtain a value from 1 to N. Hence this approach is nicknamed TCP hashing. Although simple and intuitive, the main drawback of TCP hashing is that stability cannot be guaranteed, as we will elaborate shortly.

Alternatively, most existing approaches that can guarantee both stability and packet ordering are based on some form of complete or partial aggregation of packets into frames or stripes. Uniform Frame Spreading (UFS) [16], Full-Order Frames First (FOFF) [16], Padded Frames (PF) [14], and Sprinklers [10] are representative examples of such approaches. However, these methods pay a significant price for ensuring packet ordering in that they perform significantly worse than the originally proposed basic load-balanced switch [6].

1.1 Our Approach

In this paper, we investigate the sources of instability in the TCP hashing approach in order to derive mechanisms that can mitigate them. In particular, at the first switching stage, each input port i is only connected to an intermediate port once every N time slots, or equivalently at $\frac{1}{N}$ of the line rate, via a deterministic connection pattern. Persistent overloading occurs at an input port when the arrival rate of packets hashed to the same intermediate port exceeds $\frac{1}{N}$ of the line rate for a long period of time, which can occur depending on the mix of flow sizes and durations in the group of flows that gets randomly hashed to route through the same intermediate port. For example, such persistent overloading can happen if there is a long-lived elephant flow in their midst. Although the notion of instability used here is a practical one (with respect to the limited packet buffer a switch has on each input or intermediate port), we will explain that TCP hashing could become unstable also under a theoretical and more restrictive notion of stability called rate-stability [9]. In comparison, the approach that we propose in this paper is provably rate-stable.

Similarly, at the second switching stage, each intermediate port m is also only connected to an output port j at $\frac{1}{N}$ of the line rate. Packets queued at an intermediate port may come from different input ports, possibly from all N of them. Overloading occurs at an intermediate port when the arrival rate of packets destined to the same output port, from all N inputs, exceeds $\frac{1}{N}$ of the line rate.

1.1.1 Two Safety Mechanisms. To remedy these problems, we extend the basic flow randomization scheme with two *safety* mechanisms. First, let λ_{ij} be the arrival rate for VOQ_{ij}, the Virtual Output Queue (VOQ) of packets arriving at input port i with output destination j. Depending on the hash values of their flow identifiers, the set of TCP/UDP flows within VOQ_{ij} can be partitioned into N subsets called bins. Each bin m, $m = 1, 2, \ldots, N$, corresponds to the set of flows that are hashed (and hence need to be switched) to intermediate port m. Using a simple credit scheme that we will describe in Section 3.1.4, without any knowledge or measurement of the value of λ_{ij}, we can limit the rate at which packets in each bin are served (switched) to at most $\frac{\lambda_{ij}}{N}$. We will show in Section 3.1.4 that, the first safety mechanism, when enforced on every VOQ, ensures no overloading at any input or intermediate port, *by the "normal" (i.e., rate-limited) traffic*, under any *admissible* arrival traffic, and that it does so in the least restrictive manner in the following sense: $\frac{\lambda_{ij}}{N}$ is indeed the maximum traffic rate that can be granted to each bin safely (i.e., without overloading any input or intermediate port).

However, by limiting the service rate of each such bin at an input port to $\frac{\lambda_{ij}}{N}$, those bins with traffic arrival rates exceeding that limit (e.g., bins that contain elephant flows as mentioned above) can grow in size. To ensure that these bins do not grow infinitely, thus leading to instability, we implement a second safety mechanism in which once a build-up of packets at a bin exceeds some threshold $W \geq N$ in size, we "evacuate" the excess load by *uniformly diffusing* the build-up of packets across all intermediate ports (i.e., a full-frame of N packets are uniformly spread one-to-one to the N intermediate ports). We introduce an easy-to-implement technique to ensure packet ordering when this evacuation mechanism kicks in, which involves requiring a "to-be-evacuated" bin to wait till it is safe (from packet reordering) to do so. Due to this waiting, careful scheduling is needed to coordinate an “orderly evacuation" of all bins being evacuated at any input port to ensure that every bin has a fair chance to have its backlog duly cleared, which we will explain in Section 3.1.5.

1.1.2 The Stability Proof. We prove that our Safe Randomization Switch (SRS) scheme can achieve 100% throughput (i.e., rate-stability), while guaranteeing packet order, under any admissible input traffic that is allowed to change rapidly dynamically over time. Proving the stability of SRS is very challenging partly because it appears hard to make the standard machinery of fluid analysis [8] work for this problem, despite our considerable efforts.

In arriving at this proof, we have invented a general methodology for proving the stability of queues. Our proof is based on the following extremal argument. Suppose SRS is not stable so that the total length $Q(t)$ of a subset of queues in SRS, as a function of time t, does not satisfy the stability condition $\lim_{t\to\infty} \frac{Q(t)}{t} = 0$. Then we define $\gamma \equiv \limsup_{t\to\infty} \frac{Q(t)}{t} > 0$. Let t_i, $i = 1, 2, \ldots$, be a sequence of times such that $\lim_{i\to\infty} t_i = \infty$ and $\lim_{i\to\infty} \frac{Q(t_i)}{t_i} = \gamma$. Starting with this time sequence t_i, by the properties of the aforementioned credit scheme (for rate-limiting “TCP hashed" traffic into each intermediate port) and the aforementioned scheduler for the orderly evacuation, and through standard busy period arguments, we can construct another sequence of times t'_i, $i = 1, 2, \ldots$, such that $\lim_{i\to\infty} t'_i = \infty$ and $\limsup_{i\to\infty} \frac{Q(t'_i)}{t'_i} > \gamma$, which contradicts the definition of γ.

1.2 Contributions of the Paper

The basic ideas of SRS were first proposed in an extended abstract [28]. In this paper, we expand on these ideas significantly and provide a stability proof of SRS. This paper makes the following three major contributions:

- First, we propose a new load-balanced switch architecture, called SRS, that solves the packet reordering problem of load-balanced switches and has a delay performance that compares favorably with other solutions.
- Second, we prove that, like other load-balanced switch solutions, the SRS architecture can achieve 100% throughput under any admissible arrival traffic, while guaranteeing packet order. The methodology used in the proof, described above, is novel and general, and is a major contribution by itself.
- Third, although the basic flow randomization approach has long been regarded as a simple and intuitive solution to the load-balanced switching problem, its lack of stability guarantees thus far has led many researchers to develop more complex solutions. We show that the basic flow randomization approach can indeed be made stable through the proposed safety extensions.

The rest of the paper is organized as follows. In Section 2, we describe the packet reordering problem in more details and provide background on the Uniform Frame Spreading (UFS) algorithm. In Section 3, we describe the SRS architecture in details. In Section 4, we prove the stability of the SRS architecture. In Section 5, we compare the average delay performance of SRS with other LBS solutions. In Section 6, we provide a brief review of the literature, before concluding in Section 7.

2 BACKGROUND ON BASIC LBS AND UFS

In this section, we provide a brief description of the periodic sequences of connections executed at both switching fabrics shown in Figure 1, explain the packet reorder problem of the basic Load-Balanced Switch (LBS) in more details, and describe the Uniform Frame Spreading (UFS) algorithm, a solution to this packet reorder problem and building block of our SRS architecture. In this work, we make 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 standard homogeneity assumption that every input, intermediate, or output port operates at the same speed: Each can process and transmit exactly one (fixed-size) packet per time slot. We refer to this service rate as 1. Every connection made in a switching fabric also has speed of 1 (i.e., one packet can be switched per time slot). Since N connections are made by a switching fabric at any time slot, up to N packets can be switched by it during each time slot.

The first switching fabric executes a periodic "increasing" sequence, that is, at any time slot t, each input port i is connected to the intermediate port $((i + t) \mod N) + 1$. The second switching fabric, on the other hand, executes a periodic "decreasing" sequence, that is, at any time slot t, each intermediate port m is connected to the output port $((m-t) \mod N)+1$. Following the sequence of connection patterns in the first switching fabric, each input port i can "stripe" a frame of N packets, or all packets that are ready for service if there are less than N of them, to intermediate ports $1, 2, \ldots, N$ respectively, in N consecutive time slots. As explained before, two packets belonging to the same VOQ can go to two different intermediate ports, experience different queueing delays, and arrive at the output port out of order.

Unlike the basic LBS, in which a frame of packets could come from different VOQs and be only partially filled (when there are less than N of them as mentioned above), Uniform Frame Spreading (UFS) [16] requires that every frame of packets belong to the same VOQ and the frame is fully filled, before it can be served. Each such full frame is then striped across the N intermediate ports like in a basic LBS. This ensures that for each full frame of N packets, the set of N queues at the N intermediate ports that they travel through are of equal length, and hence induce the same queueing delay on these N packets. Consequently, these N packets will (shortly) start to appear respectively at the heads of these N queues in N consecutive time slots and be striped to their respective destination output ports in the correct order by the second switching fabric. A drawback of UFS, however, is that a low-rate VOQ could incur a long buffering delay waiting for a full frame of N packets to fill in order for it to be eligible for service. SRS does not inherit this buffering delay issue from UFS, because as mentioned earlier, it uses UFS only for "evacuating" a bin (consisting of a subset of flows in a VOQ) that has too many (rather than too few) packets in it.

3 DESIGN OF SRS

Since the connection patterns at both switching fabrics are deterministic periodic sequences, as explained in Section 2, the actions of an SRS switch are completely determined by its policies of scheduling packets (for switching service) at both the input ports and the intermediate ports. We describe the operations, including such scheduling policies and other supporting mechanisms, at input ports and at intermediate ports in Sections 3.1 and 3.2 respectively.

3.1 Operations at an Input Port

Throughout this section, whenever possible, we describe only the operations at an input port i, since those at other input ports are identical. We emphasize that all these operations are fully distributed and have a total computational complexity of $O(1)$ per packet per port when properly implemented. Readers may refer to Appendix C for detailed discussions on this and other complexities (e.g., space).

3.1.1 Two Modes of Operations: RSP and UFS. As described earlier, for each output (destination) port $1 \leq j \leq N$, packets in VOQ_{ij} are divided into N bins via "TCP-hashing", which we denote as

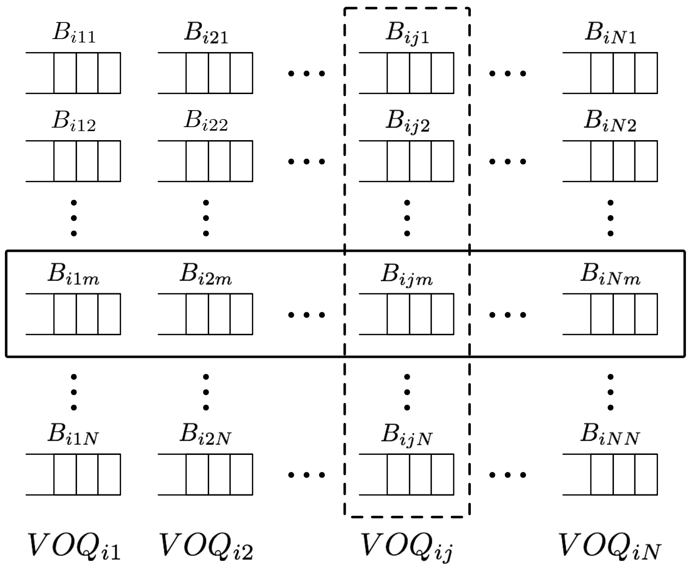


Fig. 2. The $\mathbf{N}^2$ bins at input port **i**.

$B_{ij1}, B_{ij2}, \ldots, B_{ijN}$. A logical arrangement of the N^2 bins at input port i is illustrated in Figure 2. Each column of bins correspond to a VOQ. For example, the j^{th} column, highlighted in the figure, contains N bins B_{ij1}, B_{ij2}, ..., B_{ijN} that belong to VOQ_{ij}. Each bin is a FIFO queue: All packet arrivals to the bin are to be served in the FIFO order.

By default, for any $1 \leq m \leq N$, packets in bin B_{ijm} are routed through the intermediate port m. We refer to this mode of operation as the Random Single Path (RSP) mode. A bin is in the *RSP mode* by default, unless it enters the other mode of operation called the *UFS mode* (described next) due to exceeding its rate limit, as mentioned earlier. With the RSP mode, all packets belonging to the same application (TCP or UDP) flow are routed along the *same path* through the switch, thereby ensuring their packet order. As explained earlier, we *limit*, the rate at which packets in each bin B_{ijm} can be served under the RSP mode to $\frac{\lambda_{ij}}{N}$, where λ_{ij} is the arrival rate of VOQ_{ij}. This rate-limiting is achieved using a simple credit-based mechanism (to be described in Section 3.1.4), in which a bin B_{ijm} is eligible for service under the RSP mode, if and only if the bin is in the RSP mode (i.e., has not entered the UFS mode) and has enough credit left to pay for the (RSP) service; we call such a bin *RSP-ready*.

When the packet arrival rate to a bin B_{ijm} exceeds this rate limit $\frac{\lambda_{ij}}{N}$, the nonconforming packets have to be queued at the bin, due to having no credit left to pay for their service under the RSP mode, and the length of the queue can become very long. Our solution is, once the queue length reaches a threshold W ($\geq N$), the switch will *eventually* allow all packets in the bin to "evacuate" through all N intermediate ports simultaneously, one frame (of N packets) at a time and hence at a very high rate, until the queue is cleared. We refer to this mode of operation as UFS, named after the prior work (described earlier) that serves packets within each VOQ frame by frame to avoid packet reorder [16]. As discussed earlier, when the service of a bin is switched from the RSP mode to the UFS mode, care needs to be taken so that packet reorder will not happen during the transition, which we will elaborate on in Section 3.1.3. In the meantime, we simply call a bin *UFS-ready* when it is safe (from packet reorder) to do so.

Algorithm 1 The "master" bin scheduling policy at input port i.

```
When connected to intermediate port 1:
1: if at least one bin is UFS-ready then
2:     Transmit the HOL frame (of $N$ packets) of a UFS-ready bin (say $B_{ijm}$) in the next $N$ time
       slots;
3: else
4:     Switch packets in the RSP mode in the following $N$ time slots, as described in Algorithm 2;
```

3.1.2 The "Master" Bin Scheduling Policy. The "master" bin scheduling policy, which governs the order in which these N^2 bins are serviced (by the first switching fabric), is shown in Algorithm 1. It essentially states that, when there are both RSP-ready and UFS-ready bins waiting for their respective services, UFS-ready bins take priority. This policy makes sense because UFS kicks in only when a bin has a very long queue and needs to be "evacuated" quickly. However, as we will elaborate in Appendix F, it may unfairly starve (i.e., deny service to) certain bins while sparing others, when the switch is persistently overloaded, although ideally it should starve every bin in a proportional fair way.

The "pseudocode" of this policy is shown in Algorithm 1. Whenever the input port i is connected to intermediate port 1 (by the first switching fabric), it checks whether one or more of these N^2 bins are UFS-ready. If so, the input port i selects one of the UFS-ready bins – according to the aforementioned "orderly evacuation" policy that we will elaborate on in Section 3.1.5 – for a full frame of UFS service: It transmits the HOL frame (i.e., the first N packets) of the selected bin in the next N time slots to the intermediate ports $1, 2, \ldots, N$ respectively.

Otherwise, input port i instead serves packets in the RSP mode during the next N time slots as follows. Recall that the input port i is connected to intermediate port $1, 2, ..., N$ respectively in next N time slots. This corresponds to rows $1, 2, \ldots, N$ (of bins) in Figure 2 taking turns to receive a unit (packet) of switching service. When it is the turn of row m (highlighted in Figure 2 and containing bins $B_{i1m}, B_{i2m}, \ldots, B_{inm}$), to receive service, if one or more of these bins are both RSP-ready and non-empty, one such bin will be selected – in a round-robin manner – to receive RSP service (i.e., to have its HOL packet switched to intermediate port m) during this time slot. This round-robin scheduling can be implemented by maintaining, for each "row" m, a linked list of non-empty RSP-ready bins, and the computational complexity of this implementation is only $O(1)$ per packet per port, as we will elaborate in Appendix C. Note that, adopting the round-robin policy here for scheduling non-empty RSP-ready bins (in each row m) is for optimizing the overall delay performance of the switch, and for providing a certain degree of fairness to these bins. It is not essential for ensuring switch stability: Our rate-stability proof assumes only that this scheduling policy is work-conserving in the sense the input port i must serve a non-empty RSP-ready bin in row m if at least one such bin exists.

3.1.3 UFS Waiting and Evacuation Phases for Packet Reorder Avoidance. Recall that when the queue length of a bin B_{ijm} reaches or exceeds the aforementioned evacuation threshold W, its service mode is changed to UFS (from RSP). When this happens, B_{ijm} is no longer eligible for RSP service, even if it still has unused credits, until its queue is eventually cleared by UFS. In this case, the intermediate port m is informed of this change. This notification can be piggybacked to the next packet (RSP or UFS) destined for intermediate port m, and hence does not have to consume a time slot.

This bin B_{ijm} is however not eligible for the UFS service right away (i.e. not UFS-ready) for the following reason. One or more packets sent earlier from the same input bin B_{ijm} to the intermediate port may still be queued at intermediate port m, and more specifically at intermediate bin H_{ijm}, as will be explained in Section 3.2. Suppose B_{ijm} is allowed to start receiving UFS service right away and sends out one or more UFS frames to the intermediate ports. Because UFS packets also take strict priority over RSP packets at intermediate ports, as we will explain in Section 3.2, those UFS packets sent to intermediate port m from B_{ijm} may arrive at and then depart from the output port j before those RSP packets queued in H_{ijm} do, causing packet reorder. We refer to the status of bin B_{ijm} at this moment as in the *UFS waiting phase*, in the sense that it has entered the UFS mode, but is not yet eligible for UFS service.

When the aforementioned intermediate bin H_{ijm} has been cleared at intermediate port m, a notification message is sent back to input port i. Upon receiving the notification, B_{ijm} exits the UFS waiting phase, becomes UFS-ready, and joins the ranks of other UFS-ready bins for the "orderly evacuation". We say that the bin B_{ijm} enters the *UFS evacuation phase* at this moment. By waiting for H_{ijm} to clear before evacuating B_{ijm}, which prevents the reordering between an earlier RSP packet and a later UFS packet within the same VOQ, SRS ensures correct packet order in every VOQ because, as discussed earlier, reordering cannot happen between two RSP packets or two UFS packets within the same VOQ. Note the overhead caused by such notifications is quite small, considering that even an excessively overloaded bin (say with a traffic rate close to 90% of the VOQ it belongs to) triggers such a notification no more frequent than once every $O(W)$ time slots, where $W \geq N$ is the aforementioned UFS evacuation threshold.

So far, our description of the operations at an input port i is complete except for the following two critical components: (i) the credit-based mechanism for limiting the rate at which each bin B_{ijm} can receive (switching) service under the RSP mode to $\frac{\lambda_{ij}}{N}$, where λ_{ij} is the traffic arrival rate to VOQ_{ij}, and (ii) the scheduling policy for ensuring the "orderly evacuation" of UFS-ready bins. They will be described in the next two sections respectively.

3.1.4 Credit-Based Mechanism for RSP Rate-Limiting.

In this section, we describe the aforementioned credit-based mechanism for limiting the rate, at which each bin B_{ijm} can receive switching service under the RSP mode, to $\frac{\lambda_{ij}}{N}$. Before we do so, however, we need to first explain the rationale behind setting this rate limit to $\frac{\lambda_{ij}}{N}$. Recall that RSP is the preferred and default mode of operation due to its low buffering delay, so we would like to make this rate limit as high as possible. Our rationale for this rate-limit is simple yet subtle: *For any $1 \leq i, j, m \leq N$, $\frac{\lambda_{ij}}{N}$ is the highest RSP traffic rate the input port i can grant, under any "nondiscriminatory policy" (i.e., with a policy statement that does not "discriminate against" any particular values of i, j, m), to the bin B_{ijm} without risking compromising the rate-stability of the set of bins that buffer packets destined for the output port j, at the intermediate port m.* We will elaborate on the details and the subtleties of this rationale in Appendix A.

It is actually more appropriate to consider this objective of rate-limiting a fair resource allocation scheme: Under this scheme, for any $1 \leq i, j \leq N$, the switch provides almost equal amount of RSP service to the N bins $B_{ij1}, B_{ij2}, \ldots, B_{ijN}$ that traffic in VOQ_{ij} splits into (via TCP-hashing). How to perform such a fair resource allocation, using various token bucket, leaky bucket, credit counter primitives, and their combinations, has been thoroughly studied for more than three decades [3, 17, 21, 24, 25]. In fact, many techniques for accomplishing this fair resource allocation task, all of which are slight variants of one another and provide similar or identical guarantees, can be pieced together using these "off-the-shelf" primitives.

We piece together one that is simple to state, cheap to implement, and low in computational complexity ($O(1)$ per packet per port), but do not consider it a contribution of this work. It is a credit redistribution mechanism, shown in Algorithm 2 and Algorithm 3. In this mechanism, each

Algorithm 2 Scheduling RSP packets at input port i

```
When connected to intermediate port $m = 1, 2, \ldots, N$:
1: Pick an nonempty RSP-ready bin $B_{ijm}$ with $C^R_{ijm} \geq 1$, in round-robin order, from
   $\{B_{i1m}, B_{i2m}, \ldots, B_{iNm}\}$ (i.e., bins in row $m$ as highlighted in Figure 2);
2: if such a $B_{ijm}$ exists then
3:     Switch the HOL packet of $B_{ijm}$;
4:     Update RSP credit counters per Algorithm 3;
5: else
6:     Idle;
```

Algorithm 3 Updating RSP credit counters at input port i

```
1: Initialize: Set all RSP credit counters $C^R_{ijm}$, $j, m = 1, 2, \ldots, N$ to a positive integer constant
   $C \geq 1$;
After switching the HOL packet of $B_{ijm}$, for $i, j, m = 1, 2, \ldots, N$:
2: Decrement counter $C^R_{ijm}$ by 1;
3: Increment counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijN}$ each by $\frac{1}{N}$;
```

bin B_{ijm} has a credit counter C^R_{ijm} associated with it. As shown in Line 1 of Algorithm 3, initially all credit counters are initialized to a positive constant C. Once an RSP-ready bin (say B_{ijm}) is chosen for service in the RSP mode – in the round-robin fashion as described above (Line 1 in Algorithm 2) – 1 unit of credit is subtracted from C^R_{ijm} (Line 2 in Algorithm 3), and $\frac{1}{N}$ unit of credit is deposited to each of the N credit counters associated respectively with the N bins in the j^{th} column highlighted in Figure 2 (Line 3 in Algorithm 3). In other words, the unit of credit paid by B_{ijm} for the RSP service, will be evenly distributed to all N bins belonging to VOQ_{ij}, including B_{ijm} itself. We will prove in Appendix G that this credit redistribution mechanism provides the following fair resource allocation guarantee.

LEMMA 1. *Let $D^R_{ijm}(t)$, $i, j, m = 1, \ldots, N$, be the cumulative number of packet departures from B_{ijm} in RSP mode by time slot t. Then for any two different input port bins in the same VOQ, say B_{ijm} and $B_{ijm'}$ in $VOQ(i, j)$ and for any time $t > 0$, we have*

$$|D^R_{ijm}(t) - D^R_{ijm'}(t)| \leq NC \tag{1}$$

It follows as an immediate "corollary" from this lemma that during any time interval $[t_1, t_2]$ that is "long enough," the respective average rates at which RSP service is provided to two different bins belonging to the same VOQ is roughly the same. This is precisely the aforementioned fair resource allocation objective we would like to achieve. While this credit mechanism is easy to describe, it is very computationally expensive ($O(N)$ per packet per port) to implement. In Appendix C, we will describe a low complexity ($O(1)$ per packet per port) algorithm that provides almost the same guarantee.

3.1.5 Orderly Evacuation in UFS Mode. Recall there are altogether N^2 bins at input port i, and many of them can be UFS-ready (i.e., in the UFS evacuation phase) at the same time, especially when the traffic load is heavy. As discussed in Section 1.1, these UFS-ready bins need to be evacuated (via UFS) in an orderly fashion. The idealized objective of this orderly evacuation is that, for any bin, once it enters the UFS evacuation phase, its queue length should be strictly non-increasing over time despite new packet arrivals to this bin, until it exits the UFS mode (after its queue length drops under N). This objective is however unrealistic in practice due to the following service granularity restriction: The smallest unit of UFS service is a frame (of N consecutive time slots), and while a bin is being served during these N time slots, queue lengths of some other UFS-ready bins can increase due to new packet arrivals. Hence our realistic objective is to ensure that the queue length of any UFS-ready bin is roughly non-increasing over time except for a fluctuation caused by this service granularity restriction.

This objective again can be formulated as a proportional fair resource (traffic rate) allocation problem: to serve a bin at a rate no smaller than its average packet arrival rate. Again, we achieve this objective using a scheduler comprised of aforementioned "off-the-shelf" primitives, but do not consider this scheduler a contribution of this work. More specifically, this scheduler achieves the following guarantee (that the bin length is roughly non-increasing), which we prove in Appendix J.

LEMMA 2. *Let $B_{ijm}(t)$, $i, j, m = 1, \ldots, N$, denote the queue length of the bin B_{ijm} at time slot t. If B_{ijm} remains in UFS evacuation phase during a time interval $[t_1, t_2]$, we must have*

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq N^3$$

The basic idea of this scheduler is to keep track of *increases in backlog* to a bin *after* it has become UFS-ready. This is achieved by associating a UFS "pressure" counter with each bin that keeps track of the number of new packet arrivals since the bin has become UFS-ready (i.e., the "pressure build-up"), minus the "pressure relief" in the form of UFS service provided to this bin. Whenever the scheduler needs to pick a UFS frame to serve next, it will pick the HOL frame of the bin with the highest pressure counter value. The detailed design of this scheduler is shown in Appendix B. The computational complexity of this scheduler is $O(1)$ per packet per port, as we explain in Appendix C.

3.2 Operations at an intermediate port

In SRS, intermediate ports play a passive role in packet and frame scheduling. They mostly follow the RSP and UFS scheduling decisions made by the input ports. More specifically, like input ports, intermediate ports also prioritize UFS frames over RSP packets. In addition, intermediate ports serve UFS frames in the order dictated by the input ports, which, as far as the intermediate ports are concerned, is the FIFO order.

We describe operations at an intermediate port m: Operations at other intermediate ports are identical. As shown in Figure 3, at intermediate port m, N^2 RSP bins H_{ijm} are maintained for $i, j = 1, 2, \ldots, N$. Bin H_{ijm} buffers RSP packets switched from input port i that are destined for output port j. It is clear that all these packets come from input bin B_{ijm}. In addition, N UFS bins U_{jm}, for $j = 1, 2, \ldots, N$, are maintained for buffering UFS packets sent to it from all N input ports, with all UFS packets destined for output port j appended to the end of the bin U_{jm}. Whenever intermediate port m is connected to output port j, the intermediate port m first checks if the corresponding UFS queue U_{jm} is non-empty. If so, it *first* serves its HOL packet. Otherwise, it serves a RSP packet from one of $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ in the round-robin[1] order. Finally, when a bin H_{ijm} is cleared and B_{ijm} is in the UFS waiting phase, a notification to input port i is triggered.

[1]Note that our stability proof, shown in the next section, assumes only that this service discipline is work-conserving. We choose round-robin scheduler simply because it is computationally cheap ($O(1)$ per packet per port) to implement, as we explain in Appendix C.

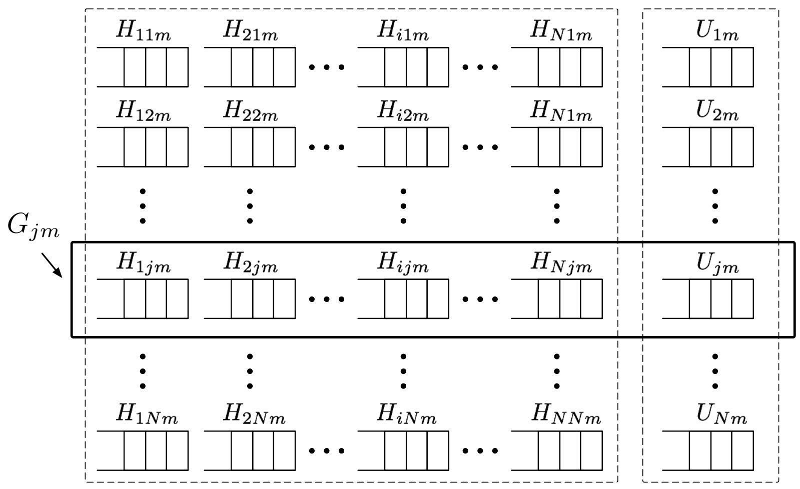

G_{jm}
H_{11m} H_{21m} H_{i1m} H_{N1m} U_{1m}
H_{12m} H_{22m} H_{i2m} H_{N1m} U_{2m}
H_{1jm} H_{2jm} H_{ijm} H_{Njm} U_{jm}
H_{1Nm} H_{2Nm} H_{iNm} H_{NNm} U_{Nm}

Fig. 3. How bins are served at intermediate port m. Queue group G_{jm}, highlighted here, will be defined in Section 4.

3.3 Variation based on input port load

In this section, we describe a variant of the baseline SRS architecture that can improve its delay performance when the traffic load is high, as will be shown in Section 5. Its basic idea is that if the total traffic arrival rate to an input port is very high, the switch simply serves it using the UFS scheme. Note this UFS scheme, which serves each VOQ frame by frame, is different than the UFS mode, which serves each bin frame by frame. We call this variant "SRS-UFS". A more detailed description of this variation is provided in Appendix D.

4 STABILITY ANALYSIS

In this section, we prove that SRS can achieve 100% throughput[2]. Equivalently, we prove that the total backlog in the B_{ijm}, H_{ijm} or U_{jm} bins does not accumulate at a positive rate over time, even when the switch is 100% loaded. This notion of (queue) stability is known as rate-stability [9], because it implies that the long-run average of packet departure rate is equal to that of packet arrival rate when the switch is no more than 100% loaded.

At the first glance, this stability is guaranteed since as queues grow longer the algorithm switches to UFS, which is known to be (rate) stable. However, the short stability explanation/proof in the UFS paper [16] does not apply to SRS for several reasons. Chief among them is that, in SRS, an input port bin can start transmitting in the UFS mode only after its corresponding intermediate port bin is cleared, which may never happen with poorly designed safety mechanisms, whereas in UFS, this transmission does not have to wait. To implicitly (i.e., woven into the fabric of the stability proof) prove that this clearance will happen because of the properly designed safe mechanisms (fair RSP rate-limiting and orderly UFS evacuations) contributes to the length and the perceived complexities of the proof.

Recall that $B_{ijm}(t)$ denotes the queue length of bin B_{ijm} at time t; $H_{ijm}(t)$ and $U_{jm}(t)$ are defined similarly. Our main result is:

[2]The rate-stability of SRS-UFS can be derived from that of SRS and of UFS.

Theorem 1.

$$(a)\quad \lim_{t\to\infty} \frac{B_{ijm}(t)}{t} = 0, \quad i,j,m = 1,\dots,N$$
$$(b)\quad \lim_{t\to\infty} \frac{H_{ijm}(t)}{t} = 0, \quad i,j,m = 1,\dots,N$$
$$(c)\quad \lim_{t\to\infty} \frac{U_{jm}(t)}{t} = 0, \quad j,m = 1,\dots,N$$

for any deterministic packet arrival process that satisfies a mild admissible condition (to be stated next) and for any arbitrary initial queue lengths of B, H, and U (at time 0).

Note that in the proof, we actually assume all queues in the switch are empty at time 0. It is however not hard to extend this proof to accommodate arbitrary initial queue lengths, as will be explained at the end of Section 4.2.

We now state the only (mild) admissible condition that we have to exogenously impose on the traffic arrival process. Let $A_{ijm}(t)$ be the cumulative number of packet arrivals into bin B_{ijm} by time slot t (since time 0). Define $A_j(t) \equiv \sum_{i,m=1}^{N} A_{ijm}(t)$. $A_j(t)$ is the total number of packets that have arrived at all input ports by time slot t and are destined for output port j. The admissible condition, which we exogenously impose, is that, for each output port j, that there exist a constant $\lambda_j \in [0,1]$ such that

$$\lim_{t\to\infty} \frac{A_j(t)}{t} = \lambda_j \quad j = 1,2,\dots,N \tag{2}$$

In other words, the long-run-average total rate of all traffic destined for output port j must exist and is no more than 1 (i.e., 100% loaded). This admissible condition is weaker than the usual notion of admissibility, in which the long-run-average of each λ_{ij}, for $i = 1,2,\dots,N$, must exist.

In the worst case, up to N packets destined for output j can arrive, one at each input port, during a single time slot. Therefore, for any $t \geq 0$, we always have

$$A_j(t) \leq Nt \tag{3}$$

Note that, in our proof, we do assume another admissible condition that at most one packet can arrive at each input port in a single time slot. This condition is however imposed "endogenously" by the maximum rate of the network link and the clock speed of the input line card circuitry, not "exogenously" by us.

In the following, we first develop a property of the packet arrival process $A_j(t)$ in the form of a general lemma, and then describe some important queuing dynamics of the switch that result from the aforementioned RSP credit redistribution mechanism and the aforementioned UFS orderly evacuation scheduler. Then we prove Theorem 1 in Section 4.2.

4.1 System dynamics and notations

We develop only the dynamics of queues that hold packets destined for (i.e., associated with) an arbitrary (but fixed) output port j; Queues associated with any other output port have the same dynamics. To facilitate the following presentation, for $i,m = 1,2,\dots,N$, we define a set of queue groups as follows

$$B_j \equiv \left\{B_{ijm} \,\middle|\, i,m = 1,2,\dots,N\right\}$$
$$B_{ij} \equiv \left\{B_{ijm} \,\middle|\, m = 1,2,\dots,N\right\}$$
$$G_{jm} \equiv \left\{U_{jm}\right\} \cup \left\{H_{ijm} \,\middle|\, i = 1,2,\dots,N\right\}$$

Let $B_j(t) \triangleq \sum_{i,m=1}^{N} B_{ijm}(t)$ be the total number of packets in queue group B_j at time slot t. Similarly we define $G_{jm}(t) \triangleq \sum_{i=1}^{N} H_{ijm}(t) + U_{jm}(t)$, $m = 1, \ldots, N$.

As mentioned earlier, throughout this paper, time is slotted and is numbered by nonnegative integers, and we use the terms "time" and "time slot" interchangeably. By convention, time starts at (slot) 0 and packets start to arrive at or after (slot) 1. When we say "at/by time (slot) t", we mean "at/by *the end of* time slot t". For example, the queue length of B_{ijm} at time t refers to that at the end of time slot t, which accounts for any (packet) arrival and/or any departure that has happened during time slot t.

4.1.1 Arrival process dynamics. In this section, we state a purely mathematical lemma (i.e., has nothing to do with switching or networking by itself) that applies to any deterministic arrival process whose long-run average converges to a constant, including the aforementioned $A_j(t)$.

Lemma 3. *Let $\{X(t),\ t \geq 0\}$, be an arbitrary deterministic time series. If there exists a constant $\lambda \in \mathbb{R}$ such that $\lim_{t\to\infty} \frac{X(t)}{t} = \lambda$, then for any $\epsilon > 0$ and $p \in (0, 1]$, there exists a constant $T_X > 0$, such that*

$$\left.\begin{cases} T \geq T_X \\ 0 \leq t_1 < t_2 \leq T \\ t_2 - t_1 \geq pT \end{cases}\right\} \Rightarrow X(t_2) - X(t_1) \leq (\lambda + \epsilon)(t_2 - t_1) \tag{4}$$

Proof. As $\lim_{t\to\infty} \frac{X(t)}{t} = \lambda$, for $\epsilon' = p\epsilon/4$, there exists a constant $T' > 0$, such that

$$t \geq T' \Rightarrow \left|\frac{X(t)}{t} - \lambda\right| < \frac{\epsilon'}{2}$$

Let $T_X = \max\left(\frac{2T'}{p}, \frac{T'}{p} + \frac{X(T')}{\frac{\epsilon}{2}\cdot p}\right)$.

As $T \geq T_X \geq \frac{2T'}{p}$ and $t_2 \geq pT$, we have $t_2 - T' \geq t_2 - pT/2 \geq pT/2$. Then we must have $T' \leq (1 - \frac{1}{2}p)t_2$ or equivalently $T' \leq \frac{2-p}{p}(t_2 - T')$. Otherwise, we'll have $t_2 \geq T' + \frac{1}{2}pT > (1 - \frac{1}{2}p)t_2 + \frac{1}{2}pT$, which implies $t_2 > T$. But this contradicts our assumption that $t_2 \leq T$. Note that $t_2 \geq pT > T'$, we must have,

$$\begin{aligned}
X(t_2) - X(T') &< (\lambda + \frac{\epsilon'}{2})t_2 - (\lambda - \frac{\epsilon'}{2})T' \\
&= (\lambda + \frac{\epsilon'}{2})(t_2 - T') + \epsilon' T' \\
&\leq (\lambda + \frac{\epsilon'}{2})(t_2 - T') + \epsilon' \frac{2-p}{p}(t_2 - T') \\
&= \left(\lambda + \frac{\epsilon'}{2} + \frac{2-p}{p}\epsilon'\right)(t_2 - T') \\
&= \left(\lambda + \frac{4-p}{2p}\epsilon'\right)(t_2 - T') \\
&\leq \left(\lambda + \frac{2}{p}\epsilon'\right)(t_2 - T') \\
&= \left(\lambda + \frac{\epsilon}{2}\right)(t_2 - T')
\end{aligned}$$

Furthermore, we have $t_2 \geq pT \geq pT_X \geq \frac{X(T')}{\epsilon/2} + T'$, which implies $X(T') - X(t_1) \leq X(T') \leq \frac{\epsilon}{2}(t_2 - T')$. Hence,

$$\begin{aligned} X(t_2) - X(t_1) &= (X(t_2) - X(T')) + (X(T') - X(t_1)) \\ &= \left(\lambda + \frac{\epsilon}{2}\right)(t_2 - T') + \frac{\epsilon}{2}(t_2 - T') \\ &\leq (\lambda + \epsilon)(t_2 - T') \\ &\leq (\lambda + \epsilon)(t_2 - t_1) \end{aligned}$$

□

Remark: We will show in Appendix E that an arrival process could be extremely bursty yet still has a long-run average rate. Hence, for such an arrival process, a time interval $[t_1, t_2]$ has to be "very long" (at least a constant fraction of T here) for its average rate during the interval to be bounded by $\lambda + \epsilon$.

4.1.2 RSP rate-limiting dynamics. How RSP rate-limiting mechanism affects the queuing dynamics of the switch is captured in the following two lemmas.

Lemma 4. *For any two intermediate ports m and m', we have $|I_{jm}(t) - I_{jm'}(t)| \leq N^2C + 1$.*

Here $I_{jm}(t)$ is defined as the cumulative number, by time slot t, of packets that arrive at the queue group G_{jm}, the j^{th} row of bins at intermediate port m as shown in Figure 3. In other words, $I_{jm}(t)$ accounts for all packets destined for output port j that arrive at intermediate port m during time interval $[0, t]$. Lemma 4 states that, the set of packets destined for output j that depart from all input ports of the switch during $[0, t]$ (denoted as $\Phi_j(t)$), arrive at every intermediate port in roughly equal numbers. Its proof, based on Lemma 1, is provided in Appendix H.

We denote as $D_j(t)$ the size of this set $\Phi_j(t)$, i.e., $D_j(t) \triangleq |\Phi_j(t)|$. The following lemma states that if a "long enough" busy period of queue group G_{jm} starts at t_1 and contains $t_2 > t_1$, then $D_j(t_2) - D_j(t_1)$, the number of packets that belong to the set $\Phi_j(t)$ and have departed from the input ports during $[t_1, t_2]$, is roughly equal to $t_2 - t_1$. Intuitively, this is because (i) Lemma 4 implies these $D_j(t_2) - D_j(t_1)$ packets arrive at at every intermediate port also in roughly equal numbers, and (ii) at least $\frac{1}{N}(t_2 - t_1)$ such packets must arrive at G_{jm} during $[t_1, t_2]$ to keep it continuously busy. Its formal proof is provided in Appendix I.

Lemma 5. *For $0 \leq t_1 < t_2$, if there exists an intermediate port m such that $G_{jm}(t_1) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_1 + 1, t_2]$, we have $D_j(t_2) - D_j(t_1) \geq (t_2 - t_1) + NG_{jm}(t_2) - 5N^3C$.*

4.1.3 UFS orderly evacuation dynamics. The effect of the aforementioned "orderly evacuation" scheduler on the dynamics of input port bins is characterized in Lemma 2 (see Section 3.1.5). For proving Theorem 1 however, we need the following lemma that is a slight generalization of Lemma 2.

Corollary 1. *If B_{ijm} has never been in UFS waiting phase during $[t_1, t_2]$, we have*

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq W^3$$

Proof. If B_{ijm} is in RSP mode at time t_2, we have $B_{ijm}(t_2) < W$ and thus

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq B_{ijm}(t_2) < W \leq W^3$$

Otherwise, if B_{ijm} is in UFS evacuation phase at t_2, it must be in UFS evacuation phase throughout $[t_1, t_2]$. By Lemma 2, we have $B_{ijm}(t_2) - B_{ijm}(t_1) \leq N^3 \leq W^3$. □

4.2 Proof of Theorem 1

In this section we present the proof of Theorem 1 in details. We first prove Theorem 1(a) (i.e., the stability of B bins) using the aforementioned extremal argument. Theorem 1(b) and (c) (i.e., the stability of H and U bins) are then much easier to prove, and their proofs will be presented in Appendix L. The following theorem implies that $\lim_{t\to\infty} \frac{B_{ijm}(t)}{t} = 0$, $i, j, m = 1, 2, \ldots, N$, which is Theorem 1(a).

Theorem 2.

$$\limsup_{t\to\infty} \frac{B_j(t)}{t} = 0 \quad j = 1, 2, \ldots, N$$

To prove Theorem 2, we need the following technical lemma concerning the busy period of a bin.

Lemma 6. *Suppose B_{ijm} remains in UFS waiting phase throughout time slots t_1 to t_2, there must exist a time slot $t_0 \le t_1$ such that $G_{jm}(t_0) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_0 + 1, t_2]$.*

Proof. Since H_{ijm} should be non-empty whenever B_{ijm} is in the UFS waiting phase, we have $G_{jm}(t) \ge H_{ijm}(t) > 0$ for any $t \in [t_1, t_2]$. Note that initially we have $G_{jm}(0) = 0$. Thus there must exist a time slot $t_0 \le t_1$ such that $G_{jm}(t_0) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_0 + 1,\ t_2]$. □

4.2.1 Intuition to the proof of Theorem 2. We provide some intuitions to the proof of Theorem 2 in this section, deferring the formal proof to Section 4.2.2. In describing these intuitions, we will liberally use vague phrases such as “roughly", “not by much", and “very large". To rid the formal proof of this vagueness accounts for the bulk of its length and perceived complexities. Following the “limsup argument" mentioned in Section 1.1.2, we assume

$$\limsup_{t\to\infty} \frac{B_j(t)}{t} = \gamma > 0 \tag{5}$$

This implies that given any $\epsilon > 0$ we have

(i) There exists an integer $T' \ge 0$, such that $t \ge T' \Rightarrow \frac{B_j(t)}{t} < (\gamma + \epsilon)$.
(ii) Given any integer $T \ge 0$, there exists $t \ge T$ such that $\frac{B_j(t)}{t} > (\gamma - \epsilon)$.

We fix a “tiny" ϵ (that is “much smaller" than γ), and T' based on this choice of ϵ. Let T_2 be a “very large" positive integer (with respect to T' and a few other large constants) such that

$$\frac{B_j(T_2)}{T_2} > (\gamma - \epsilon) \tag{6}$$

and $T_2(\gamma - \epsilon)$ is also “very large". Starting with this inequality, we would like to show that there exists T_1 that satisfies $T' < T_1 < T_2$, such that

$$B_j(T_1) > (\gamma + \epsilon)T_1 \tag{7}$$

a contradiction to Implication (i) of Equation (5). The argument we use to establish Inequality (7) is straightforward: It is literally “something just doesn't add up". More specifically, we prove that this T_1 is smaller than T_2 by “a significant amount" (of time), but the increase of $B_j(t)$, over this “very long" time interval $[T_1, T_2]$, is smaller than $(T_2 - T_1)\gamma - (T_2 + T_1)\epsilon$, the minimum amount needed to prevent Inequality (7) from happening, as follows.

Recall that the queue group B_j consists of N^2 bins, N at each input port, whose packets are destined for the output port j. They are $\{B_{ijm}\}_{i,m=1}^{N}$. Let $[t_1^{[ijm]}, t_2^{[ijm]}]$, $i, m = 1, 2, \ldots, N$, be the last UFS waiting phase of B_{ijm} before or at time T_2. If B_{ijm} has never been in the UFS waiting phase throughout $[0, T_2]$, we simply set $t_1^{[ijm]} = t_2^{[ijm]} = 0$ (the degenerated case). If B_{ijm} is still in a UFS waiting phase at T_2, we set $t_2^{[ijm]} = T_2$ (truncated by T_2). We sort these bins in the decreasing order

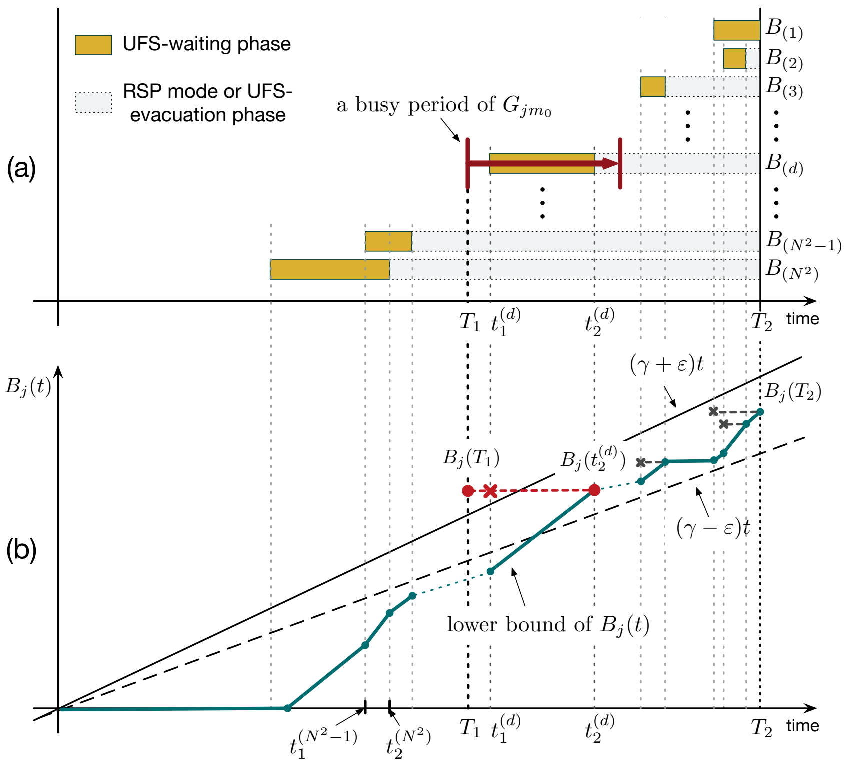


Fig. 4. Proof of Theorem 2.

of their $(t_2^{[ijm]})$'s, the ending times (possibly truncated or degenerated) of their last UFS evacuation phases before or at T_2, and relabel these bins as $B_{(k)}$, $k = 1, \ldots, N^2$ by the standard (reversed) order statistics notation (A more rigorous definition of this relabeling is provided in Appendix K). For each bin $B_{(k)}$, we also relabel this starting and ending times as $t_1^{(k)}$ and $t_2^{(k)}$ respectively. Hence, we have $T_2 \geq t_2^{(1)} \geq t_2^{(2)} \geq \cdots \geq t_2^{(N^2)}$, and for any k, $B_{(k)}$ should never be in the UFS waiting phase throughout $[t_2^{(k)}, T_2]$. Figure 4 (a) shows, from top to bottom, an example instance of these N^2 bins in this sorted order. For each bin $B_{(k)}$, its last UFS waiting phase before T_2 is shown as a yellow strip; it is followed by the interval $[t_2^{(k)}, T_2]$, shown as a light grey strip (if not degenerated).

Recall that, according to Lemma 2, once a bin enters the UFS evacuation phase, its queue length will not increase "much" (i.e., by more than a constant) and may go down. Recall also that when a bin is in the RSP mode, its queue length is less than W, the threshold that would trigger the bin entering the UFS mode. Now we zoom in to see how $B_j(t)$ can grow to the very large value of (at least) $T_2(\gamma - \epsilon)$ at time T_2. For each bin $B_{(k)}$, its queue length $B_{(k)}(t)$ at time $t_1^{(k)}$, when $B_{(k)}$ just enters its last UFS accumulation phase (from the RSP mode), is at most W as just explained. In addition, $B_{(k)}(t)$ will not increase "much" (and may decrease) during $[t_2^{(k)}, T_2]$, because $B_{(k)}$ is either in the UFS evacuation phase or in the RSP mode at any time $t \in [t_2^{(k)}, T_2]$. In other words, for each $B_{(k)}$, it can grow its queue length (of no more than W at time $t_1^{(k)}$) at a rate no more than 1 (packet per time slot) – and contribute to the growth of $B_j(t)$ – "pretty much" *only* during its "yellow strip" $[t_1^{(k)}, t_2^{(k)}]$.

Based on this "growth picture" of $B_j(t)$, we establish a lower bound of $B_j(t)$ for $t \in [0, T_2]$ by "walking back in time" starting from $t = T_2$. Figure 4 (b) shows this lower bound curve. The rightmost point on this curve $(T_2, B_j(T_2))$ lies above the line $y = (\gamma - \epsilon)x$ due to Implication (ii) of Inequality (5). As shown in Figure 4 (b), within each interval $[t_1^{(k)}, t_2^{(k)}]$ (for $k = 1, \ldots, N^2$), this lower bound curve drops as t *decreases*; otherwise (i.e., outside all these intervals) this curve is flat. The rate (i.e., slope) of this drop follows the aforementioned fact that each queue $B_{(k)}$ grows at a rate strictly no more than 1 when t *increases*. For example, in Figure 4 (b), the slope of the curve is 2 during the interval $[t_1^{(N^2-1)}, t_2^{(N^2)}]$ because the bins $B_{(N^2-1)}$ and $B_{(N^2)}$ can each potentially grow at the maximum rate of 1 during this interval.

Suppose, as shown in Figure 4 (b), $[t_1^{(d)}, t_2^{(d)}]$ is the *first* "very long" yellow strip, in the sense that none of the $d - 1$ "earlier" yellow strips (i.e., $[t_1^{(k)}, t_2^{(k)}]$, for $1 \le k \le d - 1$) is "very long". Suppose $B_{(d)}$ corresponds to $B_{i_0 j m_0}$. We know from Lemma 6 that $[t_1^{(d)}, t_2^{(d)}]$ is covered by a busy period, highlighted in Figure 4 (a), of the intermediate port queue group G_{jm_0}. The starting time of this busy period is precisely the aforementioned T_1 we are looking for. In Lemma 5, we proved that the average departure rate of queue group B_j is "roughly" 1 during this "very long" busy period. However, since this busy period is "very long", the average (total) arrival rate to all queues in the queue group B_j during this busy period is "roughly" $\lambda_j \le 1$, according to Lemma 3. Hence, during the interval $[T_1, t_2^{(d)}]$, the value of $B_j(t)$ "roughly" does not increase. This effect is illustrated in Figure 4 (b) as a horizontal line from the point $\left(t_2^{(d)}, B_j(t_2^{(d)})\right)$ "back in time" to the point $\left(T_1, B_j(T_1)\right)$. However, since the lower bound curve does not drop much when t decreases from T_2 to $t_2^{(d)}$ (because none of the $d - 1$ yellow strips before $[t_1^{(d)}, t_2^{(d)}]$ is "very long") and as just explained flattens out when t decreases from $t_2^{(d)}$ to T_1, the point $\left(T_1, B_j(T_1)\right)$ is "forced" to stick out above the curve $(\gamma + \epsilon)t$. In other words, we have $B_j(T_1) > (\gamma + \epsilon)T_1$, the Inequality (7) we are trying to prove.

4.2.2 *Formal proof of Theorem 2.*

Proof. We prove the theorem by contradiction. Suppose there exists an output port j such that

$$\limsup_{t \to \infty} \frac{B_j(t)}{t} = \gamma > 0 \tag{8}$$

Note that we must have $\gamma \le 1$ as $\limsup_{t\to\infty} \frac{B_j(t)}{t} \le \lim_{t\to\infty} \frac{A_j(t)}{t}$. Let $f(x) \triangleq \frac{\gamma+5x}{2\left(1+\frac{1}{\gamma+x}\right)^{N^2}} - 3x$; this f is defined only for introducing the number sequence $\{a_k\}_{k=0}^{N^2}$ in (13). Since $f(x)$ is a continuous function of x in a neighborhood of 0 and $f(0) > 0$, there must exist an $\epsilon' > 0$ such that $f(x) > 0$ for $x \in (0, \epsilon')$. Let $\epsilon = \min(\frac{\gamma}{4}, \frac{\epsilon'}{2})$, by (8), there exists an integer $T' > 0$, such that

$$t > T' \Rightarrow \frac{B_j(t)}{t} < (\gamma + \epsilon) \tag{9}$$

By Lemma 3, for $p = \min\left(\frac{f(\epsilon)}{2(\gamma+\epsilon)}, 1\right)$, there exists an integer $T_A > 0$, such that

$$\left\{ \begin{array}{l} T > T_A \\ 0 \le t_1 < t_2 \le T \\ t_2 - t_1 \ge pT \end{array} \right\} \Rightarrow A_j(t_2) - A_j(t_1) \le (1 + \epsilon)(t_2 - t_1) \tag{10}$$

From (8), for the same ϵ, there exists another integer

$$T_2 > \max\left(T_A, \frac{16W^5C}{f(\epsilon)}, \frac{4W^5}{\gamma - \epsilon}, \frac{2(NT' + 8W^5C)}{\gamma - 3\epsilon}\right) \tag{11}$$

such that

$$\frac{B_j(T_2)}{T_2} > (\gamma - \epsilon) \tag{12}$$

Define two number sequences $\{a_k\}_{k=0}^{N^2}$ and $\{S_k\}_{k=0}^{N^2}$ as

$$a_k = \begin{cases} f(\epsilon)T_2 + 8W^5C\left(\frac{1}{\left(1+\frac{1}{\gamma+\epsilon}\right)^{N^2}} - 1\right) & if k = 0 \\ \frac{1}{\gamma+\epsilon}\left(1 + \frac{1}{\gamma+\epsilon}\right)^{k-1}\left(a_0 + 3\epsilon T_2 + 8W^5C\right) & if k \geq 1 \end{cases} \tag{13}$$

$$S_k = \sum_{i=0}^{k} a_i \qquad k = 0, 1, \ldots, N^2$$

It is not hard to verify the following properties of these two sequences.

- The following equation holds for $k = 1, 2, \ldots, N^2$
$$a_k = \frac{1}{\gamma + \epsilon}S_{k-1} + \frac{1}{\gamma + \epsilon}(3\epsilon T_2 + 8W^5C) \tag{14}$$
- Since $\epsilon < \epsilon'$ and $T_2 > \frac{16W^5C}{f(\epsilon)}$, we have $f(\epsilon) > 0$ and
$$a_0 \geq f(\epsilon)T_2 - 8W^5C \geq \frac{f(\epsilon)}{2}T_2$$
Note that $a_{k+1} > a_k$ for $k \geq 1$, thus for $k = 1, 2, \ldots, N^2$ we have
$$a_k \geq a_1 \geq \frac{1}{\gamma + \epsilon}a_0 \geq \frac{f(\epsilon)}{2(\gamma + \epsilon)}T_2 \geq pT_2$$
- When $k = N^2$, we have
$$S_{N^2} = \frac{\gamma - \epsilon}{2}T_2$$

As mentioned in Section 4.2.1, we relabel the input port bins $\{B_{ijm}\}_{i,m=1}^{N}$ as $\{B_{(k)}\}_{k=1}^{N^2}$ in the decreasing order of the end times of their last UFS waiting phases before T_2, and denote as $[t_1^{(k)}, t_2^{(k)}]$ the last UFS waiting phase of $B_{(k)}$ before T_2. Also to make the reasoning easier, we define a dummy bin $B_{(0)}$ with $B_{(0)}(t) \equiv 0$, and a corresponding dummy interval $t_1^{(0)} = t_2^{(0)} = T_2$. Thus, for any integer $k \in [1, N^2]$, we have $t_2^{(k)} \leq t_2^{(k-1)} \leq T_2$, and $B_{(k)}$ should never be in UFS waiting phase throughout $[t_2^{(k)}, T_2]$, as shown in Figure 4 (a). From Corollary 1, we know that

$$B_{(k)}(T_2) - B_{(k)}(t_2^{(k)}) \leq W^3 \quad k = 0, 1, \ldots, N^2 \tag{15}$$

Hence

$$\begin{aligned} B_j(T_2) &= \sum_{i,m=1}^{N} B_{ijm}(T_2) = \sum_{k=0}^{N^2} B_{(k)}(T_2) \\ &= \sum_{k=0}^{N^2}\left(B_{(k)}(t_2^{(k)}) - B_{(k)}(t_1^{(k)})\right) + \sum_{k=0}^{N^2} B_{(k)}(t_1^{(k)}) \\ &\quad + \sum_{k=0}^{N^2}\left(B_{(k)}(T_2) - B_{(k)}(t_2^{(k)})\right) \end{aligned}$$

$$\begin{aligned}
&\leq \sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)}\right) + \sum_{k=1}^{N^2} W + \sum_{k=1}^{N^2} W^3 && (16)\\
&\leq \sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)}\right) + 2W^5
\end{aligned}$$

which implies that $\sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)}\right) \geq B_j(T_2) - 2W^5 \geq (\gamma - \epsilon)T_2 - 2W^5 \geq \frac{\gamma-\epsilon}{2}T_2 = S_{N^2} \equiv \sum_{k=0}^{N^2} a_k$. The last inequality holds since $T_2 \geq \frac{4W^5}{\gamma-\epsilon}$ (due to (11)). Therefore, there must exist an integer $k' \in [1, N^2]$ such that $t_2^{(k')} - t_1^{(k')} \geq a_{k'}$ (note that $t_2^{(0)} - t_1^{(0)} = 0 < a_0$). Let d be the smallest such integer; the corresponding interval $[t_1^{(d)}, t_2^{(d)}]$ is precisely the *first* "very long" yellow strip as defined in the last paragraph of Section 4.2.1. In other words, we have $t_2^{(k)} - t_1^{(k)} < a_k$ for $k = 0, 1, \ldots, d-1$ and $t_2^{(d)} - t_1^{(d)} \geq a_d$. Note that for any $k \geq d$, $B_{(k)}$ must never be in UFS waiting phase throughout $[t_2^{(d)}, T_2]$. By Corollary 1, we have $B_{(k)}(T_2) - B_{(k)}(t_2^{(d)}) \leq W^3$, $k = d, \ldots, N^2$. This inequality and (15) will be used in the first inequality below. Similar to (16), we can prove that

$$\begin{aligned}
B_j(T_2) &= \sum_{k=0}^{N^2} B_{(k)}(T_2) = \sum_{k=0}^{d-1} B_{(k)}(T_2) + \sum_{k=d}^{N^2} B_{(k)}(T_2)\\
&= \sum_{k=0}^{d-1} B_{(k)}(t_1^{(k)}) + \sum_{k=0}^{d-1} \left(B_{(k)}(t_2^{(k)}) - B_{(k)}(t_1^{(k)})\right)\\
&\quad + \sum_{k=0}^{d-1} \left(B_{(k)}(T_2) - B_{(k)}(t_2^{(k)})\right)\\
&\quad + \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=d}^{N^2} \left(B_{(k)}(T_2) - B_{(k)}(t_2^{(d)})\right)\\
&\leq \sum_{k=0}^{d-1} W + \sum_{k=0}^{d-1} \left(t_2^{(k)} - t_1^{(k)}\right) + \sum_{k=0}^{d-1} W^3\\
&\quad + \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=d}^{N^2} W^3\\
&\leq \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=0}^{d-1} \left(t_2^{(k)} - t_1^{(k)}\right) + 3W^5\\
&\leq \sum_{k=0}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=0}^{d-1} a_k + 3W^5C\\
&= B_j(t_2^{(d)}) + S_{d-1} + 3W^5C && (17)
\end{aligned}$$

By Lemma 6, there exists a time slot $T_1 \leq t_1^{(d)}$ and an intermediate port queue group G_{jm} such that $G_{jm}(T_1) = 0$ and $G_{jm}(t) > 0$ for any $t \in [T_1 + 1, t_2^{(d)}]$. Since $T_2 \geq T_A$ and $t_2^{(d)} - T_1 \geq a_d > pT_2$, by (10), we know that

$$A_j(t_2^{(d)}) - A_j(T_1) \leq (1 + \epsilon)(t_2^{(d)} - T_1) \tag{18}$$

By Lemma 5, we have $D_j(t_2^{(d)}) - D_j(T_1) \geq (t_2^{(d)} - T_1) - 5N^3C$. Hence,

$$\begin{aligned} B_j(t_2^{(d)}) &= B_j(T_1) + \big(A_j(t_2^{(d)}) - A_j(T_1)\big) - \big(D_j(t_2^{(d)}) - D_j(T_1)\big) \\ &\leq B_j(T_1) + (1+\epsilon)(t_2^{(d)} - T_1) - \big((t_2^{(d)} - T_1) - 5N^3C\big) \\ &= B_j(T_1) + \epsilon\,(t_2^{(d)} - T_1) + 5N^3C \\ &\leq B_j(T_1) + \epsilon T_2 + 5W^3C \end{aligned} \tag{19}$$

Substituting (19) into (17), we have

$$\begin{aligned} B_j(T_2) &\leq B_j(t_2^{(d)}) + S_{d-1} + 3W^5C \\ &\leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^3C \end{aligned}$$

We need only to consider the following two cases. Both lead to conclusions that contradict (12).

(i) If $T_1 > T'$, we have $T_1 \leq T_2 - (t_2^{(d)} - t_1^{(d)}) \leq T_2 - a_d$ and $B_j(T_1) \leq (\gamma+\epsilon)T_1 \leq (\gamma+\epsilon)(T_2 - a_d)$ (due to (9)). We have

$$\begin{aligned} B_j(T_2) &\leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^5C \\ &\leq (\gamma+\epsilon)(T_2 - a_d) + \epsilon T_2 + S_{d-1} + 8W^5C \\ &= (\gamma+2\epsilon)T_2 - \big((\gamma+\epsilon)a_d - S_{d-1}\big) + 8W^5C \\ &= (\gamma+2\epsilon)T_2 - (3\epsilon T_2 + 8W^5C) + 8W^5C \\ &= (\gamma-\epsilon)T_2 \end{aligned} \tag{20}$$

The second equality above holds due to Equation (14).

(ii) If $T_1 \leq T'$, by (3), we have $B_j(T_1) \leq A_j(T_1) \leq NT_1 \leq NT'$ and then

$$\begin{aligned} B_j(T_2) &\leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^5C \\ &\leq NT' + \epsilon T_2 + S_{N^2} + 8W^5C \\ &= NT' + \epsilon T_2 + \frac{\gamma-\epsilon}{2}T_2 + 8W^5C \\ &= NT' + \frac{\gamma+\epsilon}{2}T_2 + 8W^5C \\ &\leq (\gamma-\epsilon)T_2 \end{aligned} \tag{21}$$

The last inequality holds because $\epsilon = \min(\frac{\gamma}{4}, \frac{\epsilon'}{2})$, which implies $\gamma - 3\epsilon > 0$, and because $T_2 \geq \frac{2(NT'+8W^5C)}{\gamma-3\epsilon}$. □

Remark. As mentioned at the beginning of Section 4, SRS remains stable when B, H, U queues are not empty to start with (at time 0). More specifically, it can be shown that all results in Sections 4.1 and 4.2 continue to hold with some minor changes, when initial queue lengths can be nonzero. For example, in that case, we can only claim $G_{jm'}(t_1) \leq G_{jm'}(0)$ in Lemma 5, (instead of $G_{jm'}(t_1) = 0$) so its conclusion has to be changed to $D_j(t_2) - D_j(t_1) \geq (t_2 - t_1) + N(G_{jm'}(t_2) - G_{jm'}(0)) - 5N^3C$.

5 EVALUATION

In this section, we compare the performance of our proposed SRS approach, as well as the SRS-UFS variant, with other existing load-balanced switching algorithms, including the basic load-balancing scheme [6], Uniform Frame Spreading (UFS) [16], Full-Ordered Frame First (FOFF) [16], Padded Frames (PF) [14], the Sprinklers scheme [10] and the Concurrent Matching Switch (CMS) [18]. The basic load-balancing scheme (labeled "Basic") does not guarantee packet ordering, but it provides the lower bound of the delay that a load-balanced switch can achieve. UFS, FOFF, PF, Sprinklers and CMS are known to provide reasonably good performance and all of them guarantee packet ordering.

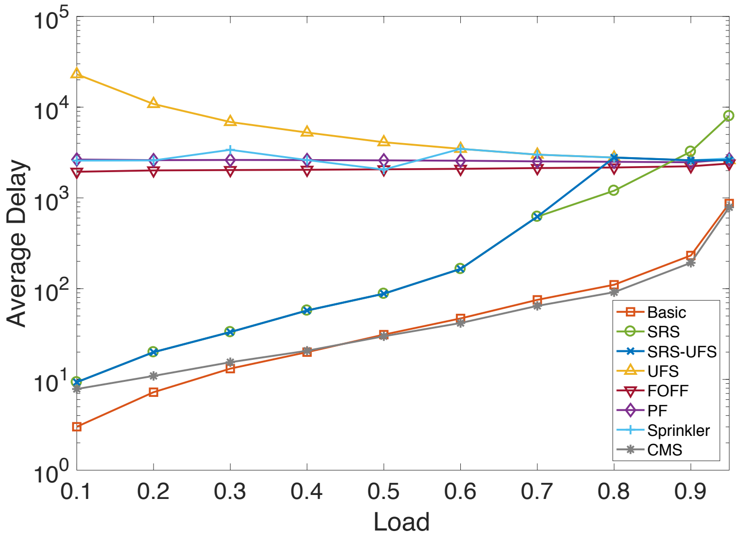
To simulate the effects of grouping application flows (i.e., TCP/UDP flows) into bins by hashing in SRS, we generate application flows over the simulation period using flow statistics measured from real-world Internet traffic traces as follows. We assume that the arrivals of new application flows to an input port follows a Poisson process, and each application flow contains only fixed-length packets, each of which takes exactly 1 time slot to switch. We also assume that the *rate* (in number of fixed-length packets per second) and the *duration* of each new application flow, viewed as a random vector $\langle \nu, \psi \rangle$, follows the joint empirical distribution measured from the traffic traces. In measuring this rate ν from a packet trace, we "segment" each variable-length packet (whose length information is included in the trace) into fixed-length packets in the sense that we consider a packet of length L (bytes) in the trace $\lceil \frac{L}{500} \rceil$ fixed-length (500-byte-long) packets. The rate of this Poisson process is set according to the intended traffic rate λ of the input port in a simulation run, and the measured empirical average size (in number of fixed-length packets) of an application flow (i.e., $\overline{\nu\psi}$). When a new flow is thus generated with rate $\nu(\omega)$, its traffic arrival process is modeled as i.i.d. Bernoulli in the sense during each time slot, there is a (fixed-length) packet arrival from this flow with probability $\nu(\omega)$.

The traces that we used were collected by University of North Carolina (UNC) on a 1 Gbps access link connecting the campus to the rest of the Internet on April 24, 2003. It contains 198,944,706 packet headers and around 13.5 million flows. In our simulation study, traffic into each input port is generated according to the empirical flow statistics measured from this trace. We also use different traffic patterns in our evaluation. The size of the switch in the simulation study is $N = 64$. The RSP-to-UFS mode threshold is set to $W = 2N$ and the initial RSP credit is set to $C = 100N$. For the SRS-UFS variant, the load thresholds that trigger the transitioning between the baseline SRS and the UFS at an input port is set to $\underline{\rho} = 0.75$ and $\bar{\rho} = 0.85$ (see Appendix D for the definitions of $\underline{\rho}$ and $\bar{\rho}$).

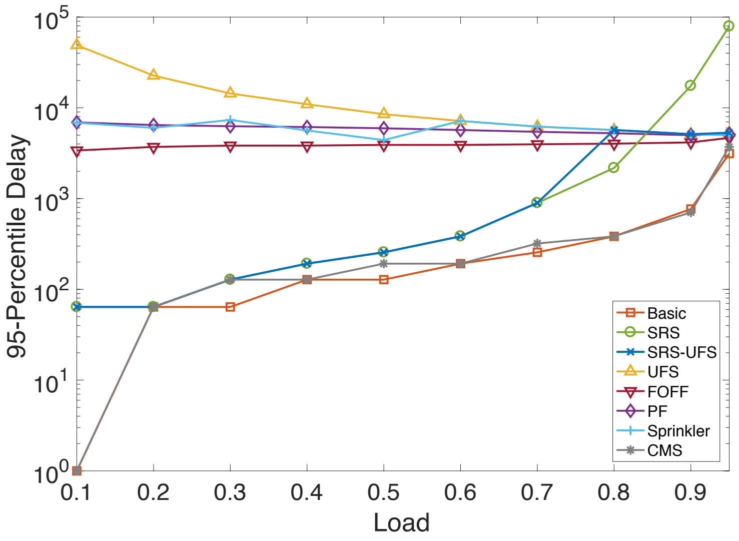
Here are the rough guidelines we follow in setting these parameters. First, we make $C \gg W$ to make sure that if the service of a bin is switched from the RSP mode to the UFS mode, it is likely not due to the lack of credit. Second, in setting parameters W and C, we take into consideration the system performance under different traffic loads. Generally speaking, larger W and C values lead to better system performance when traffic load is low to moderate, as such settings keep the system operating mainly in the RSP mode, resulting in a lower delay. However smaller W and C values work better when the traffic load is heavy, as the packets could accumulate rapidly in this case, and should be evacuated in the UFS mode as soon as possible.

Our first set of experiments assumes uniform distribution of the destination ports for the arrival flows – i.e. a new flow goes to output j with probability $\frac{1}{N}$. The results are shown in Figure 5. The second set of experiments assumes a quasi-diagonal distribution. A new flow arriving at input port i goes to output $j = i$ with probability $\frac{1}{2}$, and goes to any other output port with probability $\frac{1}{2(N-1)}$. The results are shown in Figure 6.

The following three observations can be made from the average delay results (Figures 5a and 6a) of the above experiments. First, for uniform traffic, the average delay of baseline SRS is significantly better than the existing methods of UFS, FOFF, PF, and Sprinklers for all traffic loads up to about $\lambda < 0.85$. Similarly, for quasi-diagonal traffic, the average delay of baseline SRS is significantly better than the existing methods of UFS, FOFF, PF, and Sprinklers for all traffic loads up to about $\lambda < 0.8$. Second, in comparison to the basic LBS that does not guarantee packet order, the performance of SRS follows roughly the same trend in both experiments. Third, our SRS-UFS variant improves the



(a) Average delay

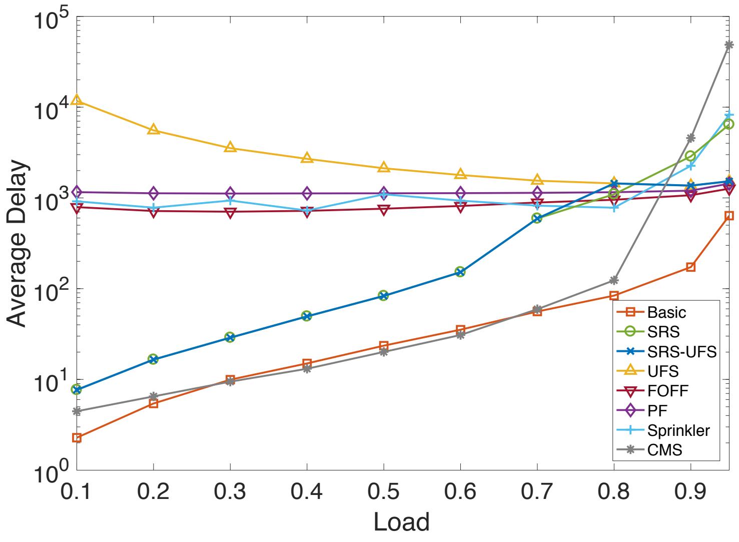


(b) 95-percentile delay

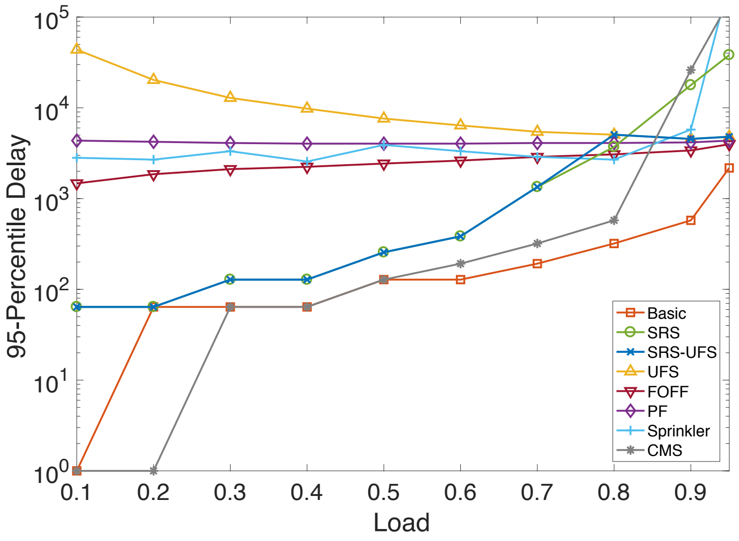
Fig. 5. Average and 95-percentile delay under uniform traffic.

performance of the baseline SRS at high loads. In both experiments, the SRS-UFS variant scheme performs almost the same as the baseline SRS when the traffic load is low to moderate ($\lambda \leq 0.75$), and as the UFS scheme when traffic load is high ($\lambda > 0.75$).

Similar observations can be made from the 95-percentile delay results shown in Figures 5b and 6b, except that the degree to which the baseline SRS performs worse than other algorithms under very heavy traffic loads (0.9 and above) in terms of 95-percentile delay is larger than that in terms



(a) Average delay



(b) 95-percentile delay

Fig. 6. Average and 95-percentile delay under quasi-diagonal traffic.

of average delay. In other words, the baseline SRS has also a larger delay variance than other algorithms under very heavy loads. Our explanation for this phenomenon is as follows. In SRS, since UFS packets take strict priority over RSP packets at both input and intermediate ports, the latter in general have larger delays than the former. The differences between the delays of these two types of packets widen when the traffic load is high, because in this case the RSP packets stuck at an intermediate port can take a long time to clear as they have to “yield” to “passing-by” UFS packets, delaying not only themselves but also packets waiting on them (in the UFS waiting phase) at their respective input ports. As shown in Figures 5b and 6b, the SRS-UFS variant effectively mitigates this problem, because as described in Section 3.3, when the total traffic arrival rate is high, the switch simply serves it using the UFS scheme and hence avoids such long waitings.

As shown in Figures 5 and 6, CMS has excellent delay performance under low-to-moderate loads. However, this comes with high communication and implementation costs, as compared to SRS and other LBS schemes. In terms of the communication cost, CMS requires, for each packet transmission, the exchange of two token messages. In terms of the implementation cost, CMS requires the computation of a matching between the input ports and the output ports, every N time slots. Although in CMS, the complexity of this computation is amortized (over N time slots) to $O(1)$ per time slot, using the SERENA matching algorithm [11], which has $O(N)$ complexity, the need to implement, and to have a fast processor execute, SERENA undoubtedly adds to the implementation cost of the switch. In comparison, no other LBS solution requires any matching computation. Finally, the delay performance of CMS is poor under high loads (≥ 0.9) for non-uniform traffic, as shown in Figure 6.

6 RELATED WORK

This section briefly reviews existing solutions to the packet reordering problem in load-balanced switches. As already discussed, Uniform Frame Spreading (UFS) [16] prevents reordering by requiring that each VOQ first accumulates a full-frame of N packets before they are uniformly spread across the N intermediate ports. The main drawback of UFS is that it suffers from $O(N^3)$ delay in the worst-case. Full Ordered Frames First (FOFF) [16] also uniformly spreads full-frames when available. When no full-frame is available, FOFF will serve incomplete frames in a round-robin manner, but it suffers from $O(N^2)$ delay in the worst-case for packet reordering at the output. Padded Frames (PF) [14] is another method that avoids the need to accumulate full-frames. When no full-frame is available, PF will pad the longest incomplete frame with fake packets to create a full-frame, which is then uniformly spread across the N intermediate ports, just like UFS. However, its worst-case delay is still $O(N^3)$. Recently, an approach called Sprinklers [10] was proposed based on the idea of variable-size striping. Sprinklers uses the arrival rate of packets to a VOQ to determine a variable stripe size L rather than always requiring a full-frame. It then only requires the accumulation of L packets in a VOQ before uniformly spreading them across a randomly chosen contiguous block of L intermediate ports, where VOQs with slower arrival rates are given smaller stripe sizes. Finally, packet ordering can be guaranteed via another approach called a Concurrent Matching Switch (CMS) [18], which enforces packet ordering throughout the switch by using a fully distributed load-balanced scheduling approach.

The technique of flow hashing (i.e., "TCP-hashing") has also been used to solve network-level traffic load balancing problems in hierarchical switch architectures (e.g., a datacenter) [12, 20]. It is widely known [1, 4, 5, 15, 22, 23] however that a naive flow hashing scheme (e.g., Equal-Cost Multipath) does not perform well in balancing loads since it could map too many large long-lived flows to the same path, leading to instability. One possible way of avoiding this instability issue is to divide each flow into small pieces and route them along different paths [2, 13], but this may again lead to the packet reorder problem. To address this packet reorder problem, in CONGA [2], packets in a flow are divided into bursts of packets (with a sizable time gap between any two consecutive bursts) called "flowlets"; if the time gap between two consecutive flowlets is larger than the maximum difference in latency among the paths, the second flowlet can be sent along a different path than the first without causing packet reordering. However, there could be a long tail in the length distribution of the flowlets, and in this case the collision on the hashing of elephant flowlets can still result in non-negligible congestion issue [13]. Presto [13] avoids this congestion issue by dividing each flow into uniform-sized pieces called flowcells and re-sequencing the out-of-order packets at the receiver, with a caveat that a significantly out-of-order (late) packet might be dropped due to the timeout of the re-sequencing timer.

7 CONCLUSIONS

In this paper, we proposed SRS, a simple randomized load-balanced switch architecture based on the hashing of application flows, combined with safety mechanisms to prevent unstable build-up of packets at intermediate queues throughout the switch. It has the lowest possible computational complexity ($O(1)$ per packet per port) and is easy to implement. We rigorously proved that the proposed SRS guarantees both stability under arbitrary admissible traffic and packet ordering. We showed experimentally that SRS has competitive delay performance compared to other solutions. Finally, we believe that this work will serve as a catalyst to a rich family of solutions based on the simple principles of flow randomization.

A SERVICE RATE LIMIT OF INPUT PORT BINS UNDER RSP MODE

In this section, we justify the following statement that we made earlier, which serves as the rationale for limiting the rate, at which each bin B_{ijm} can receive switching service under the RSP mode, to $\frac{\lambda_{ij}}{N}$.

> For any $1 \leq i, j, m \leq N$, $\frac{\lambda_{ij}}{N}$ is the highest RSP traffic rate the input port i can grant, under any "nondiscriminatory policy" (i.e., with a policy statement that does not "discriminate against" any particular values of i, j, m), to the bin B_{ijm} without risking compromising the (rate) stability of the set of bins that buffer packets destined for the output port j, at the intermediate port m.

Since it has been proven in Theorem 1 that the rate limit $\frac{\lambda_{ij}}{N}$ is safe (i.e., it will not compromise the rate stability of the switch), we need only to show here that this rate limit cannot be exceeded, which we "prove by contradiction" as follows. We come up with a set of traffic arrival rates (to the bins) that are admissible, and show that a certain set of queues at the intermediate port m have a larger total packet arrival rate than their total departure rate, and hence are not rate-stable. Now fix the values of j and m and let θ_{ijm} be the (upper) rate limit under which traffic in bin B_{ijm} can be served under the RSP mode, for $i = 1, 2, \ldots, N$. Let the arrival rates (to VOQ_{ij}) $\lambda_{ij}, j = 1, 2, \ldots, N$ satisfy $\sum_{i=1}^{N} \lambda_{ij} = 1$; this set of rates is clearly admissible (for rate-stability).

Suppose, for any $1 \leq i \leq N$, the RSP rate limit imposed on each bin B_{ijm}, satisfies $\theta_{ijm} > \frac{\lambda_{ij}}{N}$, for $i = 1, 2, \ldots, N$. We set each λ_{ijm} to $\min\{\theta_{ijm}, \lambda_{ij}\}$, for $i = 1, 2, \ldots, N$. In other words, we assume each input port i TCP-hashes its incoming traffic in the amount of exactly θ_{ijm} to bin B_{ijm}. This assumption is valid for this "proof by contradiction", because this situation could in theory happen (e.g., when, at each input port i, VOQ_{ij} contained a long-lived elephant flow that is TCP-hashed to bin B_{ijm}). In this situation, the total amount of RSP traffic that arrives at the intermediate port m and is destined for intermediate port j is equal to $\sum_{i=1}^{N} \lambda_{ijm} > \sum_{i=1}^{N} \frac{\lambda_{ij}}{N} = \frac{1}{N}$. In other words, the total arrive rate to the set of bins that buffer packets destined for the output port j at the intermediate port m is larger than $\frac{1}{N}$. However, these bins are serviced at rate $\frac{1}{N}$, as the second switching fabric connects intermediate port m with output port j once every N time slots, and hence are not (rate) stable.

Readers may wonder that to avoid this overload situation, some input port bins can be allowed to exceed this rate limit while others are not. However, the resulting rate-limit policy is no longer "nondiscriminatory" since it "discriminates" against the latter set of bins.

B UFS ORDERLY EVACUATION SCHEDULING SCHEME

Algorithm 4 Scheduling UFS packets at input port i

```
1: Initialize: Set all UFS pressure counters to $C^U_{ijm} = 0, j, m = 1, 2, \ldots, N$ ;
Upon the arrival of a packet to $B_{ijm}$:
2: if $B_{ijm}$ is in UFS evacuation phase then
3:     $C^U_{ijm} \leftarrow C^U_{ijm} + 1$;
Upon the departure of a packet from $B_{ijm}$:
4: if $B_{ijm}$ is in UFS evacuation phase then
5:     $C^U_{ijm} \leftarrow \max\left(0, C^U_{ijm} - 1\right)$;
If there are multiple USF-ready bin waiting to be served:
6: Pick the UFS-ready bin $B_{ijm}$ with the largest UFS pressure counter value to serve in the next $N$
   time slots;
```

To achieve the goal of orderly evacuation among UFS-ready bins, we piece together a scheduler that is low in both computational and implementation complexities, using “off-the-shelf" rate control mechanisms. Its pseudocode is shown in Algorithm 4.

As shown in Algorithm 4, each bin B_{ijm} is associated with a UFS “pressure” counter C^U_{ijm} which is set to 0 (Line 1 in Algorithm 4) when B_{ijm} becomes UFS-ready. This counter C^U_{ijm} tracks, since the last time it was set or decreased to 0, whether the queue length of B_{ijm} has increased – due to the number of new packet arrivals (“pressure build-up") exceeding the number of B_{ijm} packets that has been served (“pressure relief") – and if so, by how much, as follows. This pressure counter remains 0 when B_{ijm} is in the UFS waiting phase. After B_{ijm} enters the UFS evacuation phase, we increment C^U_{ijm} by 1 with each new packet arrival to B_{ijm} (Line 3). When a frame of N packets depart from a UFS-ready bin B_{ijm}, we decrement C^U_{ijm} by N. If the value of C^U_{ijm} becomes negative after this decrement, we reset it to 0. This is equivalent to executing Lines 4 - 5 in Algorithm 4 N times. We purposefully write the pseudo code this way to make it consistent with the proof of Lemma 7.

As is clear from our orderly evacuation goal, the larger the value of C^U_{ijm} is, the more urgently B_{ijm} should be served. Hence our scheduler adopts the following simple service discipline: Among all UFS-ready bins, the one with the largest UFS pressure counter value (with ties broken arbitrarily) is served next (Line 6). To implement this selection policy, we need only to maintain a heap of UFS-ready bins indexed by pressure counter values. Although the complexity of this implementation is $O(\log N)$, this complexity is not a major concern because, as we will show in Appendix C, our scheduler only incur this complexity once per frame (of N packets), and hence the amortized complexity (per packet per port) is just $O(1)$.

C IMPLEMENTATION AND COMPLEXITY

In this section, we discuss in details how to efficiently implement the SRS operations described in Section 3 and Appendix B so that their total computational complexity is $O(1)$ per packet per port.

The first issue, mentioned in Section 3.1.2, is that RSP-ready bins among bins $B_{i1m}, B_{i2m}, \ldots, B_{iNm}$ at input port i (those in the “m^{th} row" highlighted in Figure 2) should receive RSP service (i.e., to have an HOL packet switched to intermediate port m) in the round-robin order. Suppose, among these RSP-ready bins, B_{ijm} (where the "m^{th} row" intersects "j^{th} column" in Figure 2) was the last to receive RSP service. Clearly, our computation task here is to locate the next (per the round-robin order) RSP-ready bin in this "m^{th} row" when needed. A naive implementation of this computation task is to "linearly scan", starting from $B_{i[j+1]m}$, the "m^{th} row" for the next RSP-ready bin. This naive implementation, however, could incur a high computational complexity of $O(N)$ per packet per port in the worst-case, e.g., when B_{ijm} is the only RSP-ready bin in this "m^{th} row". A better approach, which has a computational complexity of only $O(1)$ per packet per port, is to maintain a linked list of non-empty RSP-ready bins. A bin is inserted to the linked list when it becomes both RSP-ready (say after receiving a unit of credit due to the above-mentioned round-robin credit distribution) and non-empty, and is deleted from the linked list when it is no longer both RSP-ready and non-empty. In addition, a pointer is associated with this linked list to remember (i.e., to point to) the RSP-ready bin to be served next. To the best of our knowledge, such a linked list implementation of efficient round-robin rotation among a dynamic set of entities can be traced as far back as to the Deficit Round Robin (DRR) packet scheduler [24].

The second implementation issue, mentioned in Section 3.1.4, is how to avoid the high computational complexity of $O(N)$ per packet per port involved in distributing credits (Line 3 of Algorithm 3), in the amount of $\frac{1}{N}$ each, to all N credit counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijn}$ (those in the "j^{th} column" highlighted in Figure 2). Our solution is to increment, instead of all these N counters by $\frac{1}{N}$ each, a single counter by 1 in the round-robin manner, thereby reducing the computational complexity to $O(1)$ per packet per port. In other words, if we last incremented credit counter C^R_{ijm}, then we should increment counter $C^R_{ij[m+1]}$ this time. To implement this round-robin rotation, we simply need to remember, for each such a set of counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijN}$, the index of the counter that was last incremented, using a pointer p_{ij}. This round-robin rotation distributes credits almost as evenly as the original scheme of incrementing all N counters: It can be shown that Lemma 1 continues to hold with a slightly larger constant bound ($NC + 1$ instead of NC). Our stability proof, which assumes the bound NC, only needs to be slightly modified to accommodate the slightly larger bound $NC + 1$.

The third issue, mentioned in Section 3.1.5 and Appendix B, is how to efficiently implement Algorithm 4 so that its computational complexity is only $O(1)$ per packet per port. With the standard data structure of organizing the pressure counters as a heap keyed by their values, the time complexity of Algorithm 4 is $O(\log N)$ per packet per port. Our solution is to associate another counter, called lazy pressure counter, with each bin. We increment the value of a lazy pressure counter only once by N after every N increments to the corresponding normal pressure counter. Hence, on average only one lazy pressure counter is incremented (by N) every N time slots. In our solution, the heap data structure is keyed instead by the values of lazy pressure counters. This way, on average only one heapify operation, which has a time complexity of $O(\log N)$, is triggered every frame (i.e., N time slots).[3] It can be shown that, by making scheduling decisions based on the slightly outdated "pressure readings" from lazy pressure counters, our streamlined scheduler increases the aforementioned fluctuation bound N^3 only slightly. Our stability proof, based on the guarantees of the original scheduler, only needs to be slightly modified to accommodate this increase.

The fourth issue, mentioned in Section 3.2 in a footnote, is how to efficiently schedule those intermediate bins among $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ that are non-empty (i.e., have at least one packet in the bin), in the round-robing manner. This issue is almost identical to the first issue, except that

[3]Although in the worst case there can be a burst of N^2 consecutive increments to the lazy counters, using a (counter) "seed value" randomization technique introduced in [29], we can ensure that, with overwhelming probability, at most $O(1)$ increments to lazy pressure counters can be triggered during every N consecutive time slots.

there is no RSP credit payment and redistribution to worry about (so it is even easier to solve). Hence the solution to the second issue, which has a computational complexity of $O(1)$ per packet per port, applies to this one.

The last issue is about the extra space complexity needed for SRS to maintain N^2 logical bins, instead of N VOQs, at each input port. This extra space complexity is relatively modest for the following reason. Like a VOQ in a standard input-queued crossbar switch, each bin here is typically implemented as a linked list of nodes, each of which contains a *pointer* to the memory address in the packet buffer space where the actual packet is stored. With such an implementation, at each input port, the only difference in memory requirement between an SRS and a standard input-queued crossbar switch, is that the former requires N^2 head pointers, one for each bin that points to (the first node of) the corresponding linked list, whereas the latter requires only N of them, one for each VOQ. Since each pointer requires only 4 bytes to store whereas each packet can be hundreds or thousands of bytes long, the extra memory space needed to store $N^2 - N$ more pointers is typically relatively modest compared to that needed to store the actual packets.

D SRS-UFS VARIATION

In this section we provide a more detailed description of the SRS-UFS variation.

(i) When a packet arrives, besides being hashed into a bin B_{ijm} as in SRS, it is also inserted into a "shadow" VOQ Z_{ij}. When a packet is removed from B_{ijm}, it is also removed from its shadow VOQ Z_{ij} (and vice versa when Z_{ij} is served by UFS as described next).

(ii) Each input port tracks its traffic arrival rate, using a lightweight measurement mechanism such as "sample and count". When the arrival rate to an input port i exceeds some threshold $\bar{\rho}$, say 0.75, the input port i stops "TCP-hashing" its traffic into the N^2 B_{ijm} bins, and instead demultiplexes all future packet arrivals to their respective shadow VOQs (there are N of them). These shadow VOQs will soon be served via UFS. However, before this "SRS to UFS" transition happens, the input port i has to wait until all N^2 corresponding H_{ijm} queues at the N intermediate ports are cleared, to prevent packet reordering.

(iii) Once an input port starts to serve all its VOQs via UFS, it continues to do so until the traffic arrival rate to the input port drops below a smaller threshold $\underline{\rho}$ (than $\bar{\rho}$), say 0.65, at which point the switch transitions back to serving the traffic via the baseline SRS. Keeping a "safe distance" between these two thresholds prevents the switch from transitioning back and forth frequently between the baseline SRS and the UFS scheme, when the traffic arrival rate to an input port fluctuates around the higher threshold $\bar{\rho}$. It also improves the tolerance of the switch to the inaccuracies in tracking the traffic arrival rates to the input ports using a lightweight measurement mechanism.

E EXTREMELY BURSTY PROCESS CAN STILL HAVE LONG-RUN AVERAGE RATE

In this section, we show that an arrival process could be extremely bursty yet still has a long-run average rate. To do so, we introduce the notion of on-off process: In such a process, during each time slot, we say that the process is "on" if there is a packet arrival, and is "off" otherwise. Now consider the following on-off process. It is on for 1 time slot, off for 1 time slot, on for 2 time slots, off for 2 time slots, on for 3 time slots, off for 3 time slots, and so on forever. In other words, the process is on and then off each for an interval that is increasing linearly over time. Clearly, this process is very bursty because, given any (arbitrarily large) time duration τ, we can find an interval of length τ (time slots) in which the average arrival rate of the process is strictly 0 (i.e, no arrival at all during the interval) and another interval of length τ in which the average arrival rate of the process is strictly 1 (i.e., τ arrivals during the interval). It is not hard to verify, however, that the long-run average arrival rate of this arrival process converges to $\lambda = 0.5$.

F DISCUSSIONS ON STABILITY AND STARVATION

In this section, we state a few important facts concerning the stability and starvation issues of SRS. We distinguish between two types of such issues. One type, which we believe is of more interest to queueing theory researchers, is whether the length of each bin is growing at most sub-linearly over time (i.e., rate-stable) – and if not, the rate of this linear growth – under various load conditions (admissible or not). The other type, which we believe is of more interest to networking researchers, is whether the length of each bin is small stochastically, under various *admissible* load conditions (e.g., under an offered load of 0.9). In the following, we elaborate on these two types of stability and starvation issues in Sections F.1 and F.2 respectively.

F.1 The First Type

We say that a queue Q is rate-starved at (maximum) rate $\bar{\gamma}$ if its queue length $Q(t)$ satisfies $\limsup_{t\to\infty} Q(t)/t = \bar{\gamma} > 0$. Since SRS is proven to be rate-stable (i.e., no bin will have its length grow with a positive rate over time) under mild admissible conditions (see Theorem 1 and the paragraph around Equation (2) in Section 4), it guarantees no rate-starvation of any bin unless the offered load (the maximum traffic arrival rate to any input or output port) exceeds 1. However, since the traffic arrival rate to any input port cannot exceed 1 (see the paragraph under Inequality (3) in Section 4), SRS guarantees no rate-starvation unless λ_j, the long-run average traffic arrival rate to an output port j, exceeds 1.

When $\lambda_j > 1$ (for some j), some of the bins whose packets are destined for the output port j will necessarily be rate-starved, no matter what switch scheduling policy is used. However, it can be desirable for the switch to display "grace under fire" in this overload situation in the sense that the switch rate-starves every bin in a fair fashion (e.g., at a rate proportional to the traffic arrival rate to the bin). As it is, an SRS switch cannot guarantee such "grace under fire", as shown in the following example. Suppose an intermediate bin $H_{i'j'm'}$ has some RSP packets in it when the corresponding input bin $B_{i'j'm'}$ just enters the UFS waiting phase, and from this point onwards, the other $N^2 - 1$ input bins whose packets all destined for the output port j' (namely $\{H_{ij'm}\}_{i,m=1}^{N} \setminus \{H_{i'j'm'}\}$) are all in the UFS evacuation phase and their total traffic arrival rate is exactly 1. In this case, only $B_{i'j'm'}$ is rate-starved (at its traffic arrival rate $\lambda_{i'j'm'}$) and no other input bin in this group is, because the RSP packets in $H_{i'j'm'}$ will never be serviced (as the UFS packets evacuated from the other $N^2 - 1$ input bins arrive at the aforementioned intermediate port queue group $\{H_{ij'm}\}_{i,m=1}^{N} \setminus \{H_{i'j'm'}\}$ at rate 1, which is equal to the service rate of this queue group), getting $B_{i'j'm'}$ stuck in the UFS waiting phase.

However, we need only to make the following slight modifications to SRS in order for it achieve a certain degree of "grace under fire" when the switch is only slightly overloaded.

(i) *Upper-bound the length of each H queue.* Whenever the queue length of any H_{ijm} exceeds a certain threshold, H_{ijm} asks B_{ijm} to refrain from sending any more RSP packets over.
(ii) *"Hawking radiation".* All N intermediate ports dedicates 1 switching cycle (N time slots) every T (typically a large constant) cycles, in a *synchronized* manner (otherwise packet reordering could happen), to serving RSP packets, even when there are UFS packets backlogged at the intermediate ports (i.e., U bins are not empty) and/or all H queues are empty.

In other words, we upper-bound the lengths of these H queues (at intermediate ports) and in addition guarantee them (analogous to black holes in the overload situation) a small but constant minimum rate of "Hawking-radiating their mass away". With this modification, SRS can eventually (i.e., in a finite amount of time) clear all H bins whose packets are destined for output port j, thereby allowing all B bins whose packets are destined for output port j to enter the UFS evacuation phase. Again, here and in the following paragraph, this j is chosen arbitrarily, but is fixed once chosen.

The following is true when the output j is only slightly overloaded (i.e., $\lambda_j > 1$ but barely). From this point onwards (i.e., after all B bins whose packets are destined for output port j have entered the UFS evacuation phase), at each input port, each such B bin is to be serviced at a rate roughly proportional to its traffic arrival rate, since the "UFS orderly evacuation" policy (see Section 3.1.5), which as we have shown guarantees approximate proportional fair rate allocation under admissible traffic, also more roughly does so when the output port j is slightly overloaded. Therefore, each such bin is to be rate-starved at a rate roughly proportional to its traffic arrival rate in the slight overload situation.

The tradeoff however is that the throughput of the switch will be reduced to $1 - 1/T$ although this reduction can be made arbitrarily small by increasing the value of the parameter T. We have proved that the modified policy is strongly stable in the fluid sense, which implies not only rate-stability, but also positive recurrence, when the traffic arrival rate matrix lies in the interior of the reduced capacity region and each element in the traffic arrival matrix process (i.e., the arrival process to each VOQ) is a renewal process. We do not include its proof here for two reasons. First, its relevance to this paper, which does not use any fluid analysis, is rather tenuous. Second, a similar "Hawking radiation" trick was used, and its stability within the reduced capacity region studied in [19] for a very different application.

We prefer the unmodified SRS over the modified SRS because the former can provably attain 100% throughput and has good empirical delay performance under *normal* workloads (i.e., when the offered load stays away from 1). We believe the primary mission of a switch scheduling algorithm is to deliver good delay performance under *normal* workloads; such "grace under fire" 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.

F.2 The Second Type

The second type of stability and starvation issues manifest themselves mostly in the empirical delay performance of SRS, which we have studied in Section 5 through simulations. Our simulation results, presented in Section 5, show that SRS has excellent delay performance under low to moderate traffic loads, which implies at least a good degree of fairness in serving different bins and lack of starvation. These simulation studies, however, have not captured the fairness and starvation behaviors of SRS in the heavy-traffic regime, i.e. under traffic loads close to 100%. We acknowledge that, in this heavy-traffic regime, UFS may starve some unfortunate B bins that are waiting on their corresponding H queues to clear at the intermediate ports. These H queues may take a very long time to clear because the intermediate ports prioritize the servicing of UFS packets in U queues over the RSP packets in H queues, and in the heavy traffic regime, these U queues can take a very long time to clear. We emphasize however that, when the switch operates in a persistent heavy-traffic regime, this waiting pain is typically a one-off, because once these H queues are eventually cleared, all bins in the switch will keep operating in the UFS mode for as long as the traffic load is very high, and hence will not face starvation.

G PROOF OF LEMMA 1

Proof. Let $C^R_{ijm}(t)$ and $C^R_{ijm'}(t)$ denote the values of credit counters C^R_{ijm} and $C^R_{ijm'}$ at time slot t respectively. Note that for any input port queue group B_{ij}, the total RSP credits stay unchanged at anytime. So we always have $C^R_{ijm}(t) + C^R_{ijm'}(t) \leq NC$.

Let $D_{ij}^R(t) \equiv \sum_{m=1}^N D_{ijm}^R(t)$ be the cumulative number of packet departures from queue group B_{ij} in RSP mode by time slot t. From Lines 2 - 3 in Algorithm 3, we know that

$$C_{ijm}^R(t) = C - D_{ijm}^R(t) + \frac{1}{N} D_{ij}^R(t)$$

which is equivalent to $D_{ijm}^R(t) = C + \frac{1}{N} D_{ij}^R(t) - C_{ijm}^R(t)$. Similarly, we have $D_{ijm'}^R(t) = C + \frac{1}{N} D_{ij}^R(t) - C_{ijm'}^R(t)$. Thus, we have

$$\begin{aligned} |D_{ijm}^R(t) - D_{ijm'}^R(t)| &= |C_{ijm}^R(t) - C_{ijm'}^R(t)| \\ &\le |C_{ijm}^R(t) + C_{ijm'}^R(t)| \\ &\le NC \end{aligned}$$

□

H PROOF OF LEMMA 4

Proof. As mentioned earlier, the set of $I_{jm}(t)$ (defined in Section 4.1) packets that arrive at the queue group G_{jm} during the time interval $[0, t]$ can be classified into two types: those RSP packets sent to $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ and those UFS packets sent to U_{jm} both from input ports. We denote the total counts of these two types of packets by $I_{jm}^R(t)$ and $I_{jm}^U(t)$, respectively. Let $I_{ijm}^R(t)$ be the number of (RSP) packets that arrive at H_{ijm} during the time interval $[0, t]$. We have $I_{jm}^R(t) = \sum_{i=1}^N I_{ijm}^R(t), j, m = 1, \ldots, N$ by definition. We also have $I_{ijm}^R(t) = D_{ijm}^R(t), i, j, m = 1, \ldots, N$, because every RSP packet that departs from B_{ijm} by time t arrives at H_{ijm} by time t. Therefore, we have

$$\begin{aligned} I_{jm}^R(t) - I_{jm'}^R(t) &= \sum_{i=1}^N \left(I_{ijm}^R(t) - I_{ijm'}^R(t) \right) \\ &= \sum_{i=1}^N \left(D_{ijm}^R(t) - D_{ijm'}^R(t) \right) \end{aligned}$$

By Lemma 1, we have

$$\left| I_{jm}^R(t) - I_{jm'}^R(t) \right| \le \sum_{i=1}^N \left| D_{ijm}^R(t) - D_{ijm'}^R(t) \right| \le N^2 C$$

Furthermore, if an input port bin is in UFS mode, around the time it sends one packet to intermediate port m, it must also send one packet from the same frame to intermediate port m' within the same cycle (N time slots). Thus, we always have $\left| I_{jm}^U(t) - I_{jm'}^U(t) \right| \le 1$ for any $t \ge 0$. Therefore,

$$\begin{aligned} |I_{jm}(t) - I_{jm'}(t)| &= \left| (I_{jm}^R(t) + I_{jm}^U(t)) - (I_{jm'}^R(t) + I_{jm'}^U(t)) \right| \\ &\le \left| I_{jm}^R(t) - I_{jm'}^R(t) \right| + \left| I_{jm}^U(t) - I_{jm'}^U(t) \right| \\ &= N^2 C + 1 \end{aligned}$$

□

I PROOF OF LEMMA 5

Proof. Let $O_{jm}(t)$ be the cumulative number of packets departure from queue group G_{jm} by time slot t. Since G_{jm} is non-empty from time slot $(t_1 + 1)$ to t_2, it must send out 1 packet per N time slots. We have

$$\begin{aligned}
&I_{jm}(t_2) - I_{jm}(t_1)\\
&= O_{jm}(t_2) - O_{jm}(t_1) + G_{jm}(t_2) - G_{jm}(t_1)\\
&\geq \left\lfloor \frac{1}{N}(t_2 - t_1) \right\rfloor + G_{jm}(t_2)\\
&\geq \frac{1}{N}(t_2 - t_1) - 1 + G_{jm}(t_2)
\end{aligned}$$

where $\lfloor x \rfloor$ is the largest integer smaller than or equal to x. From Lemma 4, for any intermediate port $m' = 1, \ldots, N$, we always have $I_{jm'}(t_2) \geq I_{jm}(t_2) - (N^2C + 1)$ and $I_{jm'}(t_1) \leq I_{jm}(t_1) + N^2C + 1$ (which will be used in the first inequality below). Note that $D_j(t) = \sum_{m'=1}^{N} I_{jm'}(t)$. We have

$$\begin{aligned}
D_j(t_2) - D_j(t_1) &= \sum_{m'=1}^{N} I_{jm'}(t_2) - \sum_{m'=1}^{N} I_{jm'}(t_1)\\
&= \sum_{m'=1}^{N} \big(I_{jm'}(t_2) - I_{jm'}(t_1)\big)\\
&\geq \sum_{m'=1}^{N} \big(I_{jm}(t_2) - (N^2C + 1) - \big(I_{jm}(t_1) + N^2C + 1\big)\big)\\
&= \sum_{m'=1}^{N} \big(I_{jm}(t_2) - I_{jm}(t_1) - 2(N^2C + 1)\big)\\
&\geq \sum_{m'=1}^{N} \left(\frac{1}{N}(t_2 - t_1) - 1 + G_{jm}(t_2) - 2(N^2C + 1)\right)\\
&\geq (t_2 - t_1) + NG_{jm}(t_2) - 5N^3C
\end{aligned}$$

□

J PROOF OF LEMMA 2

For convenience of presentation, we simply assume that B_{ijm} experiences a (degenerated) UFS waiting period of length 0 if H_{ijm} is already cleared (i.e., has length 0) when B_{ijm} enters the UFS mode. Recall from Appendix B that C^U_{ijm} is the UFS pressure counter associated with bin B_{ijm}. Let $C^U_{ijm}(t)$ denote the value of C^U_{ijm} at time t. As mentioned earlier, when B_{ijm} is in the UFS evacuation phase, C^U_{ijm} tracks the change of its queue length except that it could be allowed to transmit a UFS packet (as a part of a UFS frame) at time t, even if $C^U_{ijm}(t)$ is 0 (Line 5 in Algorithm 4), and $C^U_{ijm}(t)$ remains 0 after such a transmission. We say B_{ijm} "steals service" at such moments (without having $C^U_{ijm}(t)$ decremented). Our scheduler purposefully allows such a behavior, since Algorithm 4 guarantees that B_{ijm} needs UFS evacuation most urgently whenever it is scheduled (Line 6). If B_{ijm} steals service at time t, UFS pressure counters of all other UFS-ready bins at input port i should also have values equal or close to 0 at that time, indicating none of them needs evacuation urgently anyway. Hence B_{ijm} should not be punished for stealing service at time t. To prove Lemma 2, we need the following technical lemma.

Lemma 7. *If none of the bins at input port i has ever stolen service throughout time slots t_1 to t_2, we must have*

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C_{ijm}^{U}(t_1)\right)$$

Proof. Let $\mathcal{B}_i^U(t)$ be the set of bins at input port i that are in the UFS evacuation phase at time t. If $\mathcal{B}_i^U(t_2) = \emptyset$, we must have $C_{ijm}^U(t_2) = 0$ for $j, m = 1, 2, \ldots, N$ and thus $\sum_{j,m=1}^{N} C_{ijm}^U(t_2) = 0 \leq N$. Otherwise, let $t' \in [0, t_2]$ be the (unique) time such that $\mathcal{B}_i^U(t') = \emptyset$ and $\mathcal{B}_i^U(t) \neq \emptyset$ for any time slot $t \in [t'+1,\ t_2]$. Note that t' must exist since $\mathcal{B}_i^U(0) = \emptyset$. Then we must have $\sum_{j,m=1}^{N} C_{ijm}^U(t') = 0$. Note that when input port i is connected to intermediate port 1 at a time slot $t'' \in [t'+1,\ t'+N]$, one of the following statements must be true.

- If $t_2 \leq t''$, we have $t_2 - t' \leq N$. Note that, as assumed in Section 4, at most one packet can arrive at each input port in a single time slot and the value of $\sum_{j,m=1}^{N} C_{ijm}^U$ is incremented by at most 1 per packet arrival (Line 3 in Algorithm 4), we can obtain

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t') + (t_2 - t') \leq N$$

- If $t_2 > t''$ and $t_1 \leq t''$, input port i must send out exactly one packet in UFS mode every time slot throughout $[t'', t_2]$. $\sum_{j,m=1}^{N} C_{ijm}^U$ should then be decremented by 1 upon every departure of such packets since none of the UFS evacuation phase bins at input port i has stolen service from t'' to t_2. On the other hand, $\sum_{j,m=1}^{N} C_{ijm}^U$ can be incremented by at most 1 every time slot. Therefore $\sum_{j,m=1}^{N} C_{ijm}^U$ is strictly not increased throughout $[t'', t_2]$. Hence,

$$\begin{aligned}\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) &\leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t'') \\ &\leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t') + (t'' - t') \\ &\leq N\end{aligned}$$

- If $t_2 > t''$ and $t_1 > t''$, similarly we can prove that $\sum_{j,m=1}^{N} C_{ijm}^U$ is strictly non-increasing throughout $[t_1, t_2]$ and thus

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t_1)$$

In summary, we always have

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C_{ijm}^{U}(t_1)\right)$$

□

Now we are ready to prove Lemma 2.

Proof of Lemma 2.

Proof. Whenever B_{ijm} is in UFS evacuation phase, C^U_{ijm} will be incremented by 1 upon the arrival of every packet and be decremented by *at most* 1 upon the departure of every packet. Thus we have

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq C^U_{ijm}(t_2) - C^U_{ijm}(t_1) \leq C^U_{ijm}(t_2)$$

Therefore it's sufficient to prove that $C^U_{ijm}(t_2) \leq N^3$.

If none of the bins at input port i has ever stolen service throughout $[0, t_2]$, by Lemma 7, we have

$$\sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C^U_{ijm}(0)\right) = N \leq N^3$$

Otherwise, let $t' \geq 1$ be the time such that none of the UFS-ready bins at input port i has ever stolen service throughout time slots t' to t_2, but some bin, say $B_{ij'm'}$, steals service in time slot $(t'-1)$ to send out a packet pkt. If so, $B_{ij'm'}$ should have been scheduled to send the first packet of the frame, to which pkt belongs, at some time slot $t'' \in [t'-N,\ t'-1]$. At that time, we must have $C^U_{ij'm'}(t'') \leq N-1$ and $C^U_{ij'm'}(t'') = \max_{j,m=1}^{N} C^U_{ijm}(t'')$. Thus we have

$$\begin{aligned}\sum_{j,m=1}^{N} C^U_{ijm}(t') &\leq \sum_{j,m=1}^{N} C^U_{ijm}(t'') + (t'-t'')\\ &\leq N^2(N-1) + N\\ &\leq N^3\end{aligned}$$

Since no bin has stolen services throughout $[t', t_2]$, by Lemma 7, we have

$$C^U_{ijm}(t_2) \leq \sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C^U_{ijm}(t')\right) \leq N^3$$

□

K RELABEL $\{B_{ijm}\}_{i,m=1}^{N}$ TO $\{B_{(k)}\}_{k=1}^{N^2}$

We relabel the N^2 input port bins $\{B_{ijm}\}_{i,m=1}^{N}$ according to the end time of their last UFS waiting phase before T_2 as follows. Let $[t_1^{[ijm]}, t_2^{[ijm]}]$, $i, m = 1, 2, \ldots, N$, be the last UFS waiting phase of B_{ijm} before T_2. Now we temporarily rewrite the term $t_2^{[ijm]}$ as $t_2^{(i,j,m)}$, and B_{ijm} as $B_{(i,j,m)}$, in order to introduce the following "(reversed) order statistics" of $t_2^{[ijm]}$ and the relabeling accordingly of B_{ijm}, for $i, m = 1, 2, \ldots, N$. Define $\mathcal{V}_j$ as

$$\mathcal{V}_j = \{(i,j,m) \mid i, m = 1, 2, \ldots, N\}$$

Let $\sigma : \mathcal{V}_j \rightarrow \{1, 2, \ldots, N^2\}$ be a bijection such that $t_2^{\sigma^{-1}(1)} \geq t_2^{\sigma^{-1}(2)} \geq \cdots \geq t_2^{\sigma^{-1}(N^2)}$. To make our presentation more succinct, we use $B_{(k)}$, $t_1^{(k)}$ and $t_2^{(k)}$ as shorthands for $B_{\sigma^{-1}(k)}$, $t_1^{\sigma^{-1}(k)}$ and $t_2^{\sigma^{-1}(k)}$, respectively. Thus, for any integer $k \in [1, N^2]$, we always have[4] $t_2^{(k)} \leq t_2^{(k-1)} \leq T_2$, and $B_{(k)}$ should never be in UFS waiting phase throughout $[t_2^{(k)}, T_2]$, as shown in Figure 4 (a).

[4]For rigorousness, one can define $t_2^{(0)} = T_2$.

L PROOF OF THEOREM 1(B) AND 1(C)

In the following we will prove Theorem 3, which implies Theorems 1(b) and 1(c).

THEOREM 3.

$$\limsup_{t\to\infty} \frac{G_{jm}(t)}{t} = 0 \quad j, m = 1, 2, \ldots, N$$

PROOF. We prove the theorem by contradiction. Suppose there exists a pair of integers $j > 0$ and $m > 0$ such that

$$\limsup_{t\to\infty} \frac{G_{jm}(t)}{t} = \gamma > 0 \tag{22}$$

Thus, for $\epsilon = \frac{\gamma}{4}$, there exists an integer $T' > 0$, such that

$$t > T' \Rightarrow \frac{G_{jm}(t)}{t} < (\gamma + \epsilon)$$

By Lemma 3, for $p = \frac{\gamma-\epsilon}{N}$, there exists an integer $T_A > 0$, such that

$$\left\{ \begin{array}{l} T > T_A \\ 0 \le t_1 < t_2 \le T \\ t_2 - t_1 \ge pT \end{array} \right\} \Rightarrow A_j(t_2) - A_j(t_1) \le (1+\epsilon)(t_2 - t_1) \tag{23}$$

From (22), for the same ϵ, there exists an increasing sequence $\{T_2^k\}_{k=1}^{\infty}$ such that $\lim_{k\to\infty} T_2^k = \infty$, and for $k = 1, 2, \ldots$ we have $T_2^k > T_A$ and $\frac{G_{jm}(T_2^k)}{T_2^k} > (\gamma - \epsilon)$.

As $G_{jm}(0) = 0$ and $G_{jm}(T_2^k) > 0$, for each T_2^k, there must exist a $T_1^k < T_2^k$ such that $G_{jm}(T_1^k) = 0$ and $G_{jm}(t) > 0$ for any $t \in [T_1^k + 1, T_2^k]$.

Note that in the following proof, k (as well as k') is not an exponent in the terms T_1^k and T_2^k, but 3 in the term $5N^3C$ is. By Lemma 5, we have

$$\begin{aligned} & D_j(T_2^k) - D_j(T_1^k) \\ & \ge (T_2^k - T_1^k) + NG_{jm}(T_2^k) - 5N^3C \\ & \ge (T_2^k - T_1^k) + N(\gamma - \epsilon)T_2^k - 5N^3C \end{aligned} \tag{24}$$

Furthermore, since $G_{jm}(T_1^k) = 0$, we have

$$N \cdot (T_2^k - T_1^k) \ge I_{jm}(T_2^k) - I_{jm}(T_1^k) \ge G_{jm}(T_2^k) \ge (\gamma - \epsilon)T_2^k$$

Therefore $T_2^k - T_1^k \ge \frac{(\gamma-\epsilon)}{N} T_2^k = pT_2^k$. Since $T_2^k > T_A$, by (23), we have

$$A_j(T_2^k) - A_j(T_1^k) \le (1+\epsilon)(T_2^k - T_1^k) \tag{25}$$

Combining (24) and (25), we have

$$\begin{aligned} B_j(T_1^k) &= B_j(T_2^k) + \left(D_j(T_2^k) - D_j(T_1^k)\right) - \left(A_j(T_2^k) - A_j(T_1^k)\right) \\ &\ge \left(D_j(T_2^k) - D_j(T_1^k)\right) - \left(A_j(T_2^k) - A_j(T_1^k)\right) \\ &\ge (T_2^k - T_1^k) + N(\gamma - \epsilon)T_2^k - 5N^3C - (1+\epsilon)(T_2^k - T_1^k) \\ &\ge N(\gamma - \epsilon)T_2^k - 5N^3C - \epsilon T_2^k \\ &\ge N(\gamma - 2\epsilon)T_2^k - 5N^3C \end{aligned}$$

We consider two cases:

(i) If $\max_{k=1}^{\infty}\{T_1^k\} < \infty$, there must exist an integer $k' > 0$ such that $T_2^{k'} > \frac{N \max_{k=1}^{\infty}\{T_1^k\}+5N^3C}{N(\gamma-2\epsilon)}$, since $\lim_{k\to\infty} T_2^k = \infty$. This however implies $B_j(T_1^{k'}) \geq N(\gamma-2\epsilon)T_2^{k'} - 5N^3C > N\max_{k=1}^{\infty}\{T_1^k\} \geq NT_1^{k'}$, which contradicts the fact that $B_j(T_1^{k'}) \leq A_j(T_1^{k'}) \leq NT_1^{k'}$ (due to (3)).

(ii) Otherwise, we must have $T_1^k \to \infty$ as $k \to \infty$, thus

$$\begin{aligned}\limsup_{k\to\infty} \frac{B_j(T_1^k)}{T_1^k} &\geq \limsup_{k\to\infty} \frac{N(\gamma-2\epsilon)T_2^k - 5N^3C}{T_1^k}\\ &\geq \limsup_{k\to\infty} \frac{N(\gamma-2\epsilon)T_1^k - 5N^3C}{T_1^k}\\ &= N(\gamma-2\epsilon)\\ &> 0\end{aligned}$$

This contradicts Theorem 2.

□

ACKNOWLEDGMENTS

This project is supported in part by NSF grants CNS-1423182 and CNS-1248117.

REFERENCES

[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks.. In *NSDI*, Vol. 10. 19–19.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 503–514.

[3] Brahim Bensaou, KT Chan, and Danny HK Tsang. 1997. Credit-based fair queueing (CBFQ): A simple and feasible scheduling algorithm for packet networks. In *IEEE ATM Workshop 1997. Proceedings.* IEEE, 589–594.

[4] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies.* ACM, 8.

[5] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. 2013. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.* ACM, 49–60.

[6] Cheng-Shang Chang, Duan-Shin Lee, and Yi-Shean Jou. 2002. Load balanced Birkhoff–von Neumann switches, part I: one-stage buffering. *Computer Communications* 25, 6 (2002), 611–622.

[7] Cheng-Shang Chang, Duan-Shin Lee, and Ching-Ming Lien. 2002. Load balanced Birkhoff–von Neumann switches, part II: multi-stage buffering. *Computer Communications* 25, 6 (2002), 623–634.

[8] J. G. Dai. 1998. *Stability of fluid and stochastic processing networks.* University of Aarhus. Centre for Mathematical Physics and Stochastics (MaPhySto)[MPS].

[9] J. G. Dai and Balaji Prabhakar. 2000. The throughput of data switches with and without speedup. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 2. IEEE, 556–564.

[10] Weijun Ding, Jun Xu, J. G. Dai, Yang Song, and Bill Lin. 2014. Sprinklers: A randomized variable-size striping approach to reordering-free load-balanced switching. In *ACM CoNext, the 10th International Conference on Emerging Networking EXperiments and Technologies.* ACM.

[11] Paolo Giaccone, Balaji Prabhakar, and Devavrat Shah. 2003. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE Journal on Selected Areas in Communications* 21, 4 (2003), 546–559.

[12] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, Vol. 39. ACM, 51–62.

[13] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 465–478.

[14] Juan José Jaramillo, Fabio Milan, and R Srikant. 2008. Padded frames: a novel algorithm for stable scheduling in load-balanced switches. *Networking, IEEE/ACM Transactions on Networking* 16, 5 (2008), 1212–1225.

[15] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 51–62.

[16] Isaac Keslassy. 2004. *The load-balanced router.* Ph.D. Dissertation. Stanford University.

[17] HT Kung, Trevor Blackwell, and Alan Chapman. 1994. Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation and statistical multiplexing. In *ACM SIGCOMM Computer Communication Review*, Vol. 24. ACM, 101–114.

[18] Bill Lin and Isaac Keslassy. 2010. The concurrent matching switch architecture. *Networking, IEEE/ACM Transactions on* 18, 4 (2010), 1330–1343.

[19] Siva Theja Maguluri, R Srikant, and Lei Ying. 2012. Stochastic models of load balancing and scheduling in cloud computing clusters. In *INFOCOM, 2012 Proceedings IEEE.* IEEE, 702–710.

[20] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, Vol. 39. ACM, 39–50.

[21] Cüneyt Özveren, Robert Simcoe, and George Varghese. 1994. Reliable and efficient hop-by-hop flow control. In *ACM SIGCOMM Computer Communication Review*, Vol. 24. ACM, 89–100.

[22] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving datacenter performance and robustness with multipath TCP. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 266–277.

[23] Mehrnoosh Shafiee and Javad Ghaderi. 2017. A simple congestion-aware algorithm for load balancing in datacenter networks. *IEEE/ACM Transactions on Networking* (2017).

[24] Madhavapeddi Shreedhar and George Varghese. 1995. Efficient fair queueing using deficit round robin. In *ACM SIGCOMM Computer Communication Review*, Vol. 25. ACM, 231–242.

[25] Jonathan Turner. 1986. New directions in communications(or which way to the information age?). *IEEE communications Magazine* 24, 10 (1986), 8–15.

[26] Leslie G. Valiant. 1982. A scheme for fast parallel communication. *SIAM journal on computing* 11, 2 (1982), 350–361.

[27] Sen Yang, Bill Lin, Paul Tune, and Jun Jim Xu. 2017. A simple re-sequencing load-balanced switch based on analytical packet reordering bounds. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE.* IEEE, 1–9.

[28] Sen Yang, Bill Lin, and Jun Xu. 2016. Safe Randomized Load-Balanced Switching by Diffusing Extra Loads. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science.* ACM, 397–398.

[29] Qi Zhao, Jun Xu, and Zhen Liu. 2006. Design of a novel statistics counter architecture with optimal space and time efficiency. *ACM SIGMETRICS Performance Evaluation Review* 34, 1 (2006), 323–334.

Received August 2017; revised October 2017; accepted December 2017.