Received December 5, 2017, accepted January 15, 2018, date of publication February 1, 2018, date of current version March 16, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2800686

Highest Rank First: A New Class of Single-Iteration Scheduling Algorithms for Input-Queued Switches

BING HU[1,2], (Senior Member, IEEE), FUJIE FAN[1], (Student Member, IEEE), KWAN L. YEUNG[3], (Senior Member, IEEE), AND SUGIH JAMIN[4]

[1]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China
[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100088, China
[3]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong
[4]Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109, USA

Corresponding author: Fujie Fan (fjfan@zju.edu.cn)

This work was supported in part by the Open Foundation of State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications under Grant SKLNST-2013-1-20, in part by the National Natural Science Foundation of China under Grant 61401394, and in part by the Huawei Project YBN2016090060.

ABSTRACT In this paper, we study a new class of single-iteration scheduling algorithms for input-queued switches based on a new arbitration idea called *highest rank first* (HRF). We first demonstrate the effectiveness of HRF by a simple algorithm named Basic-HRF. In Basic-HRF, virtual output queues (VOQs) at an input port are ranked according to their queue sizes. The rank of a VOQ, coded by $\log(N+1)$ bits, where N is the switch size, is sent to the corresponding output as a request. Unlike all existing iterative algorithms, the winner is selected based on the ranks of the requests/grants. We show that the rank-based arbitration outperforms the widely adopted queue-based arbitration. To improve the performance under heavy load and maximize the match size, Basic-HRF is integrated with an embedded round-robin scheduler. The resulting HRF algorithm is shown to beat almost all existing single-iteration algorithms. But, the complexity of HRF is high due to the use of multi-bit requests. A novel request encoding/decoding mechanism is then designed to reduce the request size to a single bit while keeping the original performance of HRF. A unique feature of the resulting coded HRF (CHRF) algorithm is that the single-bit request indicates an *increase* or *decrease* of a VOQ rank, rather than an empty VOQ or not. We show that the CHRF is the most efficient single-bit-single-iteration algorithm.

INDEX TERMS High-speed networks, scheduling algorithm, switches.

I. INTRODUCTION

In cloud computing architectures [1]–[3], services and data reside in shared data centers, and are accessed by users over the Internet. Huge amount of IP traffic will be transported between users and data centers, and among hundreds of thousands of servers within each data center. Besides IP traffic, most data centers have storage area networks for carrying data between servers and disk arrays. Some data centers also have high-performance computing interconnects for low-latency inter-process communications. There are many kinds of data center networks [4]–[8]. Among them, switches are always the core devices for data transmission. To meet the needs of high-speed and low-latency switching in these data center networks, it is highly desirable to have a unified switch fabric for different types of traffic. Therefore, the next generation switch/router design becomes more urgent than ever.

As compared to output-queued switch, input-queued switch [9], [10] is more suitable for high-speed implementation because of its reduced requirement for memory bandwidth – at most one packet is sent/received by an input/output port in each time slot. For input-queued switches, iterative scheduling algorithms are widely accepted due to the use of massive parallel processing [11], [12]. In general, each iteration of an iterative scheduling algorithm consists of three phases, *request*, *grant* and *accept*. In the *request* phase, each input sends a matching request to each output. In the *grant* phase, each output selects one request to grant. And in the final *accept* phase, each input selects one grant to accept, and notifies the selected output. The matched inputs and outputs will not participate in the subsequent iterations. In each iteration, both the grant and accept messages are single-bit (for notification). But the request messages can be either single- or multi-bit, according to the information that each input needs to transmit.

For $N \times N$ switches, an iterative scheduling algorithm can be executed up to N iterations in order to guarantee a maximal match size. Although each more iteration aids in the match size, the scheduling overhead is also increased. The scheduling time is proportional to the number of iterations. In high-speed switches, scheduling algorithms with multiple iterations are almost unachievable. For example, on a 100Gbps line carrying 64-byte packets, one time slot is about 5ns. It means that the scheduling time should be much less than 5ns, as most time should be assigned to data transmission. Such an extremely small time slot puts a huge pressure on the design of scheduling algorithms. To minimize the scheduling time, as in [13] and [14], we are devoted to designing a *single-iteration algorithm*. In a single-iteration algorithm, there is no need to send accept messages in the accept phase. Thus, there are only two times of communication between any pair of input and output in each time slot.

Besides the number of iterations, the scheduling time is also related to the transmission delay between inputs and outputs. In a multi-bit algorithm, a request consists of multiple bits, and it must be serialized via a device called SerDes (serializer/deserializer) before transmission. A SerDes consists of a transmitter and a receiver. The transmitter at an input converts parallel data to serial form for transmission and the receiver at an output converts serial data back to parallel form for processing. The transmission delay between an input and an output mainly depends on the speed of the SerDes. The state-of-the-art technology allows a SerDes to operate at 25Gbps. Efforts are made to raise it to 100Gbps based on a multi-lane approach of running four 25Gbps SerDes in parallel [15]. Even though the speed will be higher, it cannot catch up the requirement of high-speed switches, as the time slot is becoming shorter and shorter (e.g., 5ns or less). Therefore, we consider to design a *single-bit algorithm*, which eliminates the need of SerDes and minimize the transmission delay between inputs and outputs.

The resulting scheduling algorithms are called *single-bit-single-iteration algorithms*. Such a class of algorithms incurs minimum scheduling overhead and is very suitable for future high-speed switch implementation.

However, the existing single-iteration algorithms [13], [14] are not ideal in performance. So we propose a new class of single-iteration scheduling algorithms for input-queued switches based on the notion of *Highest Rank First* (HRF). In essence, we rank all Virtual Output Queues (VOQs) at each input according to their queue sizes, where rank 1 is given to the longest VOQ. Arbitrations are based on VOQ ranks and highest rank first. Different from the queue-based arbitration [13], [16], the rank-based arbitration will select a shorter VOQ as the winner if its rank is higher than a longer VOQ. This solves the synchronizing problem of queue-based solutions where all outputs tend to grant the same "busy" input [17]. We first design a basic algorithm named Basic-HRF. In Basic-HRF, each request message has $\log(N+1)$ bits for carrying the full rank information from 0 to N, where N is the switch size and rank 0 is reserved for empty VOQs. Basic-HRF is used to demonstrate the effectiveness of rank-based arbitration over queue-based arbitration. To enhance the performance under heavy load, Basic-HRF is integrated with an embedded round-robin scheduler. The idea is to put more efforts on maximizing the match size under heavy load. We show that the refined Basic-HRF algorithm, or HRF in short, outperforms almost all existing single-iteration algorithms. However, HRF is a multi-bit (i.e., $\log(N+1)$ bits) algorithm. Aiming at cutting down the request size to a single bit, a novel request encoding/decoding mechanism is then designed. We call the resulting algorithm Coded Highest Rank First (CHRF). Unlike all existing iterative algorithms, a single-bit request in CHRF is used to indicate an *increase* or *decrease* of a VOQ rank, rather than a VOQ is empty or not. As we will see, CHRF outperforms all other single-bit-single-iteration algorithms and even most multi-bit-single-iteration algorithms. The time complexities of rank-based algorithms are also pretty low, e.g., $O(\log N)$ for HRF and $O(1)$ for CHRF.

The rest of the paper is organized as follows. In the next section, we provide the related work and qualitatively compare the rank-based algorithms with other iterative scheduling algorithms in the literature. In Section III, we introduce the Basic-HRF and HRF algorithms. In Section IV, the encoding/decoding mechanism is detailed and the resulting CHRF algorithm is presented. In Section V, we quantitatively compare our algorithms with existing single-iteration scheduling algorithms by simulations. In Section VI, we construct an analytical model for delay performance of HRF under low traffic load. Finally, we conclude the paper in Section VII.

II. RELATED WORK

In the past twenty years, various iterative scheduling algorithms have been designed. They differ mainly in the information sent in the request phase, and the arbitration mechanisms adopted in the grant and accept phases. Usually a subtle change in the design can result in a big difference in performance. To the best of our knowledge, only a few iterative scheduling algorithms [17] have been adopted by the industry.

PIM [18] is one of the earliest iterative scheduling algorithms. It is a single-bit algorithm. In the request phase, each input sends a "1" (i.e., 1-request) to each backlogged output and a "0" (i.e., 0-request) to each empty output. The subsequent arbitrations at output (for selecting a 1-request to grant) and input (for selecting a grant to accept) are based on random selection. In [17], it was shown that random selection is ineffective in desynchronizing the winners selected by different outputs/inputs. As a result, the match size obtained by PIM is limited. Aiming at desynchronizing the winners, iSLIP [17] uses a local round-robin (RR) pointer/arbiter at each output/input and the highest scheduling priority is given to the request/grant currently pointed to by the RR pointer. An output advances its RR pointer only if its grant is accepted – this ensures that different RR pointers will shift to different inputs especially when the load is high.

Unlike iSLIP, DRR [11] and its variants [12] only allow each input to send at most one 1-request. And the 1-request is sent to the output that is pointed to by the local RR pointer if the corresponding VOQ is backlogged. Otherwise, the 1-request will be sent to the next nonempty VOQ. The grant phase is the same as that of iSLIP, i.e., the 1-request from the input pointed to by the local RR pointer has the highest priority. When an output grants an input, the output knows for sure that its grant will be accepted because the input has only sent one 1-request. Accordingly, no arbitration at input is needed because each input will receive at most one grant. So the 3-phase iteration in iSLIP becomes 2-phase in DRR. It was found that with a single iteration, both iSLIP-1 and DRR-1 give comparable performance. With x iterations ($1 < x < N$), the performance of iSLIP-x is noticeably better than DRR-x because iSLIP allows multiple 1-requests to be sent by each input in the request phase.

Another interesting effort in enhancing the performance of iSLIP is SRR [13]. SRR is a single-bit-single-iteration algorithm. Like DRR [11], SRR only allows each input to send at most one 1-request in the request phase. Unlike DRR (or iSLIP), no *local* RR pointer is maintained at individual input/output ports. Instead, like our HRF algorithm, a *global* RR scheduler is maintained among all inputs and outputs by assigning each input i a distinct preferred output j in each time slot. In the request phase, if VOQ(i,j) is backlogged, input i sends a 1-request to output j; otherwise, input i sends a 1-request to its longest VOQ. In the grant phase, an output grants the 1-request from its preferred input. If there is no 1-request from its preferred input, the output randomly selects a 1-request to grant. Due to the better desynchronization ability, SRR outperforms iSLIP-1 when the traffic is uniform and heavy. But its performance under nonuniform traffic is generally poor. In Section V, we further show that its performance under hotspot input traffic pattern (as shown in FIGURE 8) is the worst among all algorithms simulated.

There are some other single-bit scheduling algorithms. For example, pDRR [12] generalizes DRR to support multiple priorities. With LQD [19], each input sends one request probabilistically based on the VOQ length. For the algorithm in [20], an input sends requests to a subset of the outputs whose corresponding VOQs are backlogged. Finally, SRRR [21] adopts a 4-phase iteration, where the extra/first phase is for an output to inform an input if the input is pointed to by the RR pointer at this output. But the complexity involved is rather significant.

On the other hand, it is generally believed that with multi-bit requests, the performance of iterative scheduling algorithms can be enhanced (at the cost of extra communication and processing overheads). Notably, the requests sent by iLQF [16] carry the VOQ size, and the requests sent by iOCF [16] carry the "age" of the Head-of-Line (HoL) packet. If an output/input receives more than one request/grant, iLQF prefers the longest VOQ and iOCF likes the VOQ with the oldest HoL packet. iLQF and iOCF are comparable in both performance and implementation complexity. In Section III-A, we will show that rank-based arbitration is not only more effective than the queue-based arbitration, but also simpler to implement.

π-RGA [14] is a multi-bit-single-iteration algorithm. The multi-bit request is used to indicate the last time when the VOQ transforms from empty to nonempty. The key idea of π-RGA is to reuse the matches established in the previous slots. The VOQs that are backlogged earlier and the one matched in the previous slot are given higher scheduling priority. π-RGA gives the best performance under bursty traffic when the input load is very high (e.g., >0.7 in FIGURE 6). But the algorithm is too complicated for high-speed implementation, and its performance under other traffic patterns is generally poor.

References [22] and [23] provide the preliminary thinking of highest rank first. The solutions presented by these papers obtain the best performance among the compared solutions. However, they are far from a mature scheduling algorithm for practical switches. The solution in [22] does not consider the effects of empty queues. In this paper, we will point out that the outputs should avoid to grant any empty queues (by using the special rank 0) and the requests of preferred inputs must be sent explicitly. We use a single-bit request to indicate an increase or decrease of a VOQ rank, rather than the actual rank of a VOQ as in [22]. Compared with the solution in [23], we construct a systematic theory of highest rank first (in Section III and Section IV), redesign the encoding/decoding mechanism in a more elegant way, and provide the implementation for queue ranking with low time complexity. We also provide an theoretical model to analyze the delay and throughput performance for HRF in this paper.

Round-Robin with Longest Queue First (RR/LQF) [24] is a new single-bit algorithm. Unlike all existing algorithms, RR/LQF replaces the request phase by a report phase. In the report phase, depending on if there is a packet arrival to a particular VOQ, the input sends either "1" or "0" to the output. Each output keeps track of the number of packets destining to it and awaiting at the inputs. RR/LQF also uses a global RR scheduler, implemented based on the preferred input-output relationship as in [13] to maximize the match size, and a LQF scheduler to serve the most critical VOQs. Unlike the (local) LQF scheduler in [13], the winner chosen by the (global) LQF in RR/LQF is the longest among all VOQs destining to it. When RR/LQF is executed for a single iteration, it was shown that the resulting single-bit-single-iteration RR/LQF-1 outperforms all existing single-bit-single-iteration scheduling algorithms. But the switch-on-a-chip (SoC) implementation of RR/LQF-1 can be challenging. This is because each output port needs to maintain N packet counters, one for each VOQ destining to it. Let the maximum VOQ size be B packets. Then N^2 counters, each with $\log B$ bits, must be maintained. Besides, $\log B$-bit comparators are needed to determine the longest queue. The complexity involved is thus non-trivial. Last but not least, as will be shown in this paper, the queue-based arbitration adopted by RR/LQF is not as effective as our rank-based arbitration.

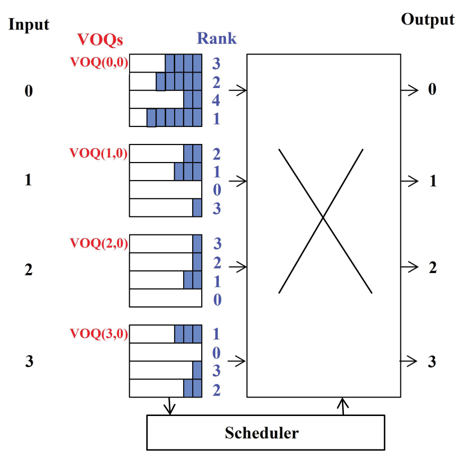

FIGURE 1. A 4 × 4 input-queued switch. Each input has 4 VOQs which are ranked according to their sizes. All empty VOQs have the special rank 0.

III. HIGHEST RANK FIRST

A. BASIC-HRF ALGORITHM

Consider the 4×4 switch in FIGURE 1, where input 0 is a traffic "hotspot." With a Longest Queue First (LQF) algorithm such as iLQF [16], each input sends a request (carrying the specific VOQ size) to each output. At output 0, four requests are received and among them, VOQ(0,0) is the longest (with size 4). Output 0 selects the request from VOQ(0,0) to grant (by sending a "1"). Similarly, outputs 1, 2 and 3 select VOQ(0,1), VOQ(0,2) and VOQ (0,3), respectively. In this case, although each output selects its own winner independently, the selected winners are synchronized. As a result, input 0 receives 4 grants, but only one of them (i.e., VOQ(0,3)) can be accepted. Accordingly, the single-iteration algorithm based on LQF can only produce a match of size 1 and weight 6 (i.e., the queue size of VOQ(0,3)).[1]

With the Basic Highest Rank First (Basic-HRF) algorithm, at output 0 in FIGURE 1, the four requests received from VOQ(0,0), VOQ(1,0), VOQ(2,0) and VOQ(3,0) have ranks 3, 2, 3 and 1, respectively. The highest rank is from VOQ(3,0), and its request is granted *although the queue size of* VOQ(3,0) *is shorter than* VOQ(0,0). Similarly, outputs 1, 2 and 3 grant the requests from VOQ(1,1), VOQ(2,2), and VOQ(0,3), respectively. As a result, every input receives a grant, and a match of size 4 and weight 14 is obtained by using Basic-HRF.

From the examples above, we can see that ranks are of *local significance* (of a particular input), whereas queue sizes are of *global significance* (across all inputs). To desynchronize the requests chosen by outputs (so as to maximize both match size and match weight), rank-based arbitration is arguably better than queue-based arbitration because of the following salient points:

- If an output grants a request with rank 1, its grant will be accepted for sure (by the corresponding input); but if an output grants a request with the longest queue (among all requests it received), the grant will be rejected if the input receives another grant corresponding to a longer VOQ.
- At an input, the ranks of all VOQs (except empty ones) are distinct, but their queue sizes can be the same.
- For the switch as a whole, there can be at most N VOQs with the same rank (excluding the special rank 0), but there can be at most N^2 VOQs with the same size.

[1]Match size is the number of VOQs that have been matched to outputs in a switch configuration. Match weight is the sum of the sizes of all VOQs that have been matched to outputs in that time slot.

The last two points above imply that when queue-based arbitration is used, an output will encounter more situations of multiple requests with the same highest scheduling priority. As a result, the output have to select the winner randomly (among VOQs with the same queue size). As is pointed out in [17], random selection (as in PIM [18]) can actually intensify grant synchronization – multiple outputs, though acting independently, grant the requests from the same input simultaneously.

In addition to the better performance, we can also see that the rank-based arbitration is simpler to implement than queue-based one because (a) *the request message size is reduced*: without regard to coding the request, there are at most $(N + 1)$ rank values, but the range of queue size can be much larger, and (b) *the associated processing overhead is reduced*: at each output/input, the arbitration is done by comparators. With only $(N + 1)$ possible rank values, rank comparators are simpler to implement than those in queue-based algorithms.

B. HRF ALGORITHM

Basic-HRF aims at serving the most *critical* queues first, and the criticality is measured by queue ranks. That means maximizing match size is not of primary importance to Basic-HRF. When the input load is light, this is not an issue because a less-than-perfect match size is good enough to deliver all arriving packets efficiently. But when the load is heavy, more packets will be waiting for transmission at inputs, and the packet delay will increase rapidly with the queue size. To maximize the match size under heavy load, we propose to enhance the Basic-HRF algorithm with an embedded round-robin (RR) scheduler [25], and we simply call the enhanced algorithm HRF.

The embedded RR scheduler is based on the notion of *preferred input-output pairs*. Consider an $N \times N$ input-queued switch. In each time slot, each input is assigned a distinct preferred output. Without loss of generality, for input i at time slot t, its preferred output j is

$$j = (i + t) \bmod N. \tag{1}$$

When input i prefers output j, output j also prefers input i. So the preferred relationship is reciprocal. From (1), we can see that each input prefers each output exactly once in every N slots, forming a round-robin schedule among all input-output pairs. Consider the special case that all inputs always have packets for their preferred outputs. The match size formed in each time slot, by giving the highest priority to the preferred input-output pairs, is always maximum, i.e., N.

In the HRF algorithm, scheduling priority is given to the preferred input-output pairs *first* (for maximizing the match size), and the VOQs with the highest rank *next* (for serving the most critical queues). At an arbitrary time slot t, HRF functions as follows:

- *Request*: If output j is the preferred output and VOQ(i,j) is not empty, input i sends a request with rank 1 to output j and *a request with rank 0 to everyone else*. If VOQ(i,j) is empty, input i sends requests with the *actual* rank values to all outputs. Note that rank 1 is the highest rank, and rank 0 is reserved for non-preferred input-output pairs or empty VOQs which should not attract any grants.
- *Grant*: An output grants the request (with rank 1) from its preferred input. Otherwise, the output grants the request with the highest rank (and breaks tie randomly).
- *Accept*: An input accepts the grant from its preferred output. If there is no preferred grant, the input accepts the grant with the highest rank. (Note that the VOQ ranking is available locally at each input.)

The pseudo code of HRF algorithm is given in Algorithm 1. Note that (a) all inputs run procedures Request and Accept in parallel, (b) all outputs run procedure Grant in parallel, and (c) no message needs to be sent in procedure Accept.

In the request phase, if an input has packets destining to its preferred output, the input sends a request with rank 1 to its preferred output, *indicating that the preferred VOQ is not empty* (instead of indicating the longest VOQ). At the same time, the input sends a request with rank 0 to everyone else even though the corresponding VOQs are not empty. This subtle design is to ensure that the input does not *unnecessarily* attract any grants from other outputs, because the input knows for sure that, with HRF algorithm, its request will be granted by its preferred output. It should be emphasized that in designing iterative scheduling algorithms, subtle differences like this can make a big difference in performance.

In HRF algorithm, the preferred input-output pairs always have higher priority than the rank-based arbitration. Does it imply that the rank-based arbitration is always overshadowed by the embedded RR scheduler? The answer depends on the input traffic load. When the traffic is light, most inputs do not have packets waiting for their preferred outputs. As a result, the embedded RR scheduler will not take effect, and the rank-based arbitration mechanism dominates. But when the traffic is heavy, most inputs tend to have packets waiting for their preferred outputs. Accordingly, the embedded RR scheduler will automatically kick in to maximize the match size. In this case, the rank-based arbitration will play a secondary/supplementary role.

Simulation results show that Basic-HRF algorithm and HRF algorithm give almost identical (and the best) performance when the input traffic is below 0.6. This confirms that the performance of HRF algorithm is dominated by its rank-based arbitration mechanism when the traffic is not heavy.

Algorithm 1 Highest Rank First Algorithm

```
1:  procedure Request($in$, $time$)
2:      $\mathbf{L}[i]$ = the length of VOQ $i$ at input $in$
3:      $pout = (in + time) \bmod N$
4:      if $\mathbf{L}[pout] > 0$ then
5:          send 1 to output $pout$
6:          send 0 to other outputs
7:      else
8:          sort and rank L
9:          $\mathbf{K}[i]$ = the rank of VOQ $i$
10:         send each $\mathbf{K}[i]$ to the corresponding output
11:     end if
12: end procedure
13: procedure Grant($out$, $time$)
14:     $\mathbf{R}[i]$ = the request from input $i$
15:     $pin = (out + N - time \bmod N) \bmod N$
16:     if $\mathbf{R}[pin] == 1$ then
17:         send 1 to input $pin$
18:         send 0 to other inputs
19:     else
20:         $m = \arg\min_{i \in [0,N)} \mathbf{R}[i] \neq 0$
21:         send 1 to input $m$
22:         send 0 to other inputs
23:     end if
24: end procedure
25: procedure Accept($in$, $time$)
26:     $\mathbf{K}[i]$ = the rank of VOQ $i$ got from Request
27:     $\mathbf{G}[i]$ = the grant from output $i$
28:     $pout = (in + time) \bmod N$
29:     if $\mathbf{G}[pout] == 1$ then
30:         accept the grant from output $pout$
31:     else
32:         $n = \arg\min_{\mathbf{G}[i]=1} \mathbf{K}[i] \neq 0$
33:         accept the grant from output $n$
34:     end if
35: end procedure
```

When the traffic load is greater than 0.6, HRF algorithm gives a much better delay performance than Basic-HRF algorithm. This is because the embedded RR scheduler has kicked in to boost the match size.

IV. CODED HIGHEST RANK FIRST

Despite its outstanding delay-throughput performance, HRF algorithm requires a request with $\log(N+1)$ bits to carry the full rank information. As compared to the class of single-bit-single-iteration algorithms, the implementation complexity of HRF algorithm is relatively high. In this section, we aim at minimizing the size of request messages.

A. A SIMPLE 2-BIT APPROACH

A closer examination of Basic-HRF reveals that the probability that a VOQ is matched successfully reduces quickly with the rank. In other words, most successfully matched VOQs have ranks 1 or 2 (i.e., the longest or second-longest VOQs). To minimize the request size, instead of sending the full $(N+1)$ ranks, we only need to send *the most important ranks*.

Assume only two binary bits are used to carry the rank information. A total of four ranks can be identified. In this case, we argue that the four most important *literal* ranks are: "longest," "second-longest," "others" and "empty." To avoid to attract more than one grants, at each input port there can be at most one "longest" VOQ, and one "second-longest" VOQ. But there can be multiple "empty" VOQs. When a VOQ is not the longest/second-longest/empty, it belongs to "others." In other words, ranks 3 to N in a full rank algorithm (Basic-HRF or HRF) are merged to a single rank "others." Among the four ranks designed above, "longest" and "empty" are the most important ones because granting a non-longest queue is inefficient, and granting an empty queue is a waste.

Although the above approach of using two-bit requests can give a decent delay-throughput performance, our goal is to use single-bit requests so that the HRF algorithm can be converted into a single-bit-single-iteration algorithm.

B. USING SINGLE-BIT REQUEST

We observe that existing single-bit algorithms [11], [13], [17] convey two VOQ states to an output, *empty* or *not*. We propose to use a single-bit request to indicate an *increase* or *decrease* of the VOQ rank. Without loss of generality, let X_t be the single-bit request sent by a VOQ to a specific output at time slot t. Based on its rank at the previous time slot $(t-1)$, the VOQ sets $X_t = 1$ (or 0) if its current rank is increased (or decreased).

But what if its rank remains the same at both slot t and slot $(t-1)$? Note that the two possible values of X_t have already been used. Besides, when X_t is received by an output, how to determine its rank at the corresponding input? We will address these two questions in the next subsection based on a three-rank model. At each input, we classify each VOQ into one of the three literal ranks: "empty," "others" and "longest." If a VOQ belongs to "others," its queue size is neither empty nor the longest. Obviously, there can be multiple VOQs having the rank of "empty" or "others." If there are multiple VOQs with the same longest queue size, we only label one of them as "longest" and the rest will be demoted to become "others." Again, this subtle design is to prevent an input from attracting multiple grants (while it can only accept one of them).

C. BASIC-HRF WITH REQUEST CODING

We first focus on the request coding scheme while assuming the Basic-HRF algorithm is used. Let X_t be the single-bit request sent by a VOQ at time slot t. With the three-rank model, our request coding scheme at each input is summarized in FIGURE 2, and is detailed below.

- If a VOQ changes from "empty" (at slot $t-1$) to "others" *or* "others" to "longest" *or* "empty" to "longest," its rank increases and $X_t = 1$ is sent.
- If a VOQ changes from "longest" to "others" *or* "others" to "empty" *or* "longest" to "empty," its rank decreases and $X_t = 0$ is sent.
- If a VOQ remains in "others," $X_t = !X_{t-1}$ is sent (i.e., an alternate sequence of 0 and 1 will be generated).
- If a VOQ remains in "longest," $X_t = 1$ is sent.
- If a VOQ remains in "empty," $X_t = 0$ is sent.

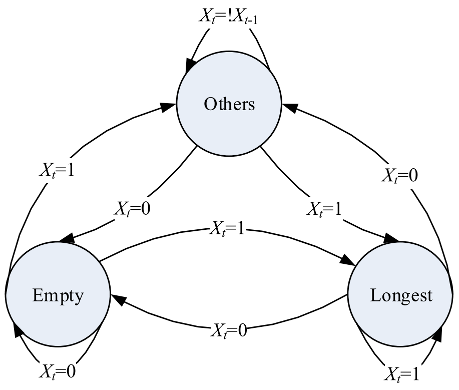

FIGURE 2. A single-bit request coding scheme based on the three-rank model.

TABLE 1. Decoding the rank of a VOQ.

X_tX_{t-1}	00	01	10	11
Ranks	Empty	Empty Others	Others Longest	Longest

In the above coding scheme, special attention is required if the rank of a VOQ remains unchanged. Specifically, for the VOQ remaining in "longest," since there is no rank higher than "longest," we keep $X_t = 1$. Similarly, for the VOQ remaining in "empty," we keep $X_t = 0$. But for a VOQ remaining in "others," we use *an alternate sequence of 0 and 1*. For example, if a request sent in the previous slot is $X_{t-1} = 0$, the request sent in the current slot will be $X_t = !X_{t-1} = 1$.

Next we focus on the decoding process at an output upon receiving a (coded) single-bit request X_t. Note that each output is required to keep a copy of the requests received from each input in the previous slot, i.e., X_{t-1}. When X_t is received, an output decodes the rank of the corresponding VOQ jointly with the stored X_{t-1}. The decoding TABLE 1, and explained below.

Since $X_t = 1$ indicates a rank increase, receiving consecutive 1s implies that the corresponding VOQ has a higher rank. Focusing on the three-rank model in FIGURE 2, it can be seen that two consecutive 1s, i.e., $X_tX_{t-1} = 11$, *guarantee* a state change ending at "longest," and such a guarantee is independent of the initial state of a VOQ.[2] Take an example. Assume that the initial state is "empty." With $X_{t-1} = 1$, the VOQ will be promoted, at least, to "others" at slot $(t-1)$.

[2]Initial state refers to the state of a VOQ before sending X_{t-1}, i.e., at the end of slot $(t-2)$ or at the beginning of slot $(t-1)$.

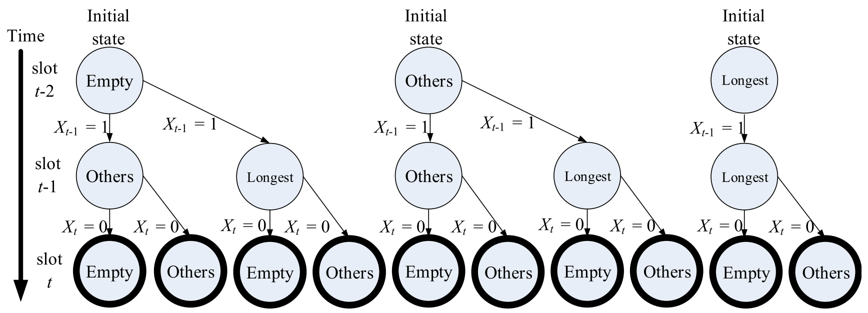

FIGURE 3. All possible state transitions for $X_tX_{t-1} = 01$. The resulting state is either "empty" or "others," no matter what the initial state is.

With $X_t = 1$ at slot t, the VOQ becomes the "longest" for sure.

On the other hand, receiving consecutive 0s implies that the VOQ has a lower rank. From FIGURE 2, it can be seen that two consecutive 0s, i.e., $X_tX_{t-1} = 00$, *guarantee* a state change ending at "empty." In other words, we can unambiguously identify a VOQ as empty ($X_tX_{t-1} = 00$) or the longest ($X_tX_{t-1} = 11$). This property is particularly important because, as we have highlighted before, granting an empty VOQ is a waste and granting a non-longest VOQ is inefficient.

If the received requests form an alternate sequence of 0 and 1, it implies no (significant) change in rank. Since the decoding is based on X_tX_{t-1}, we still want to examine if there is any difference between "01" and "10". Let us focus on $X_tX_{t-1} = 01$ first. From FIGURE 2, we can find out that the current state of the VOQ can be either "empty" or "others." To better understand it, the following scenarios are considered:

- If the initial state is "empty," $X_{t-1} = 1$ indicates a state transition from "empty" to "others" *or* "empty" to "longest." At the end of slot $(t - 1)$, there are two possible states: "others" or "longest."
  - For the case of "others," $X_t = 0$ indicates another transition from "others" to "empty" *or* "others" to "others."
  - For the case of "longest," $X_t = 0$ indicates another transition from "longest" to "empty" *or* "longest" to "others."
- If the initial state is "others," $X_{t-1} = 1$ indicates a state transition from "others" to "others" *or* "others" to "longest." At the end of slot $(t - 1)$, there are two possible states: "others" or "longest."
  - For the case of "others," $X_t = 0$ indicates another transition from "others" to "empty" *or* "others" to "others."
  - For the case of "longest," $X_t = 0$ indicates another transition from "longest" to "empty" *or* "longest" to "others."
- If the initial state is "longest," $X_{t-1} = 1$ indicates a state transition from "longest" to "longest." Then $X_t = 0$ indicates another transition from "longest" to "empty" *or* "longest" to "others."

FIGURE 3 summarizes the state changes in decoding $X_tX_{t-1} = 01$. We can see that the current state of a VOQ under consideration (i.e., circles in bold lines) will be either "empty" or "others." Similarly, the current state of a VOQ with $X_tX_{t-1} = 10$ will be either "longest" or "others," no matter what its initial state is. Between a request with $X_tX_{t-1} = 01$ and another with $X_tX_{t-1} = 10$, the request with "10" should have a higher scheduling priority because, at least, "10" guarantees the corresponding VOQ is nonempty (either "longest" or "others"), whereas "01" cannot (due to the probability of being "empty").

In summary, the decoding scheme at each output is shown in TABLE 1:

- If $X_tX_{t-1} = 00$, the VOQ is in "empty" state (regardless of the VOQ state when X_{t-1} was sent).
- If $X_tX_{t-1} = 01$, the VOQ is either in "empty" state or in "others" state.
- If $X_tX_{t-1} = 10$, the VOQ is either in "others" state or in "longest" state.
- If $X_tX_{t-1} = 11$, the VOQ is in "longest" state.

Based on the highest rank first arbitration, the scheduling priority is "11" > "10" > "01" > "00". Then among (up to) N requests received, an output can easily identify the VOQ with the highest rank using a 2-bit comparator only.

D. CODED HIGHEST RANK FIRST (CHRF)

Unlike Basic-HRF, HRF algorithm has an embedded round-robin scheduler to maximize the match size. Besides, HRF algorithm always schedules the preferred input-output pairs first, and applies the rank-based arbitration mechanism next. With the above in mind, Coded Highest Rank First (CHRF) can be implemented as follows:

- *Request*: At each input i, if VOQ(i,j) is nonempty, where j is the preferred output in the current time slot from (1), $X_t = 1$ it sent to output j, and $X_t = 0$ is sent to all other outputs. Otherwise, for each VOQ at input i,
  - if it changes from "empty" (at slot $t - 1$) to "others" *or* "others" to "longest" *or* "empty" to

"longest," $X_t = 1$ is sent to its corresponding output;
  - if it changes from "longest" to "others" *or* "others" to "empty" *or* "longest" to "empty," $X_t = 0$ is sent;
  - if it remains in "others," $X_t =!X_{t-1}$ is sent;
  - if it remains in "longest," $X_t = 1$ is sent;
  - if it remains in "empty," $X_t = 0$ is sent.
- *Grant*: At each output j, if $X_t = 1$ is from output j's preferred input, grant this request. Otherwise, among all the requests received, grant the one with the maximum (binary) value of X_tX_{t-1}. (If there is a tie, select the winner randomly.)
- *Accept*: At each input i, if a grant from input i's preferred output is received, accept it. Otherwise, among all grants received, accept the one with the highest rank. (If there is a tie, select the winner randomly.)

The pseudo code of CHRF is given in Algorithm 2 and the encoding/decoding mechanism is shown in Algorithm 3.

The CHRF algorithm contains some subtle features which deserve a close examination. In the request phase, if an arbitrary input i has packets for its preferred output j, it sends $X_t = 1$ to output j to indicate "grant me as I have packets for you," and $X_t = 0$ to all other outputs to indicate "do not grant me." Therefore, X_t's in this case are not coded using FIGURE 2, and they are not used to indicate the rank increase or decrease. (But if input i does not have packets for its preferred output j, X_t's are coded using FIGURE 2.)

When output j receives $X_t = 1$, it knows that $X_t = 1$ should be interpreted as "its preferred input i has packets for it, and it must grant input i." But when other outputs receive $X_t = 0$, they do not know if $X_t = 0$ should be interpreted as a coded request (i.e., a rank decrease), or an indication of "do not grant me." This is because they do not know if input i has packets for its preferred output j or not, which is only known by output j itself. Our solution is simple (yet effective): If an X_t is received from its non-preferred input, the output always treats it as a coded request. Indeed, most of the time it should be treated as a coded request, especially when traffic is light. (When traffic is heavy, treating it as a coded request or not will not affect the performance because the performance then will be dominated by the embedded round-robin scheduler.)

If this turns out to be a mistake, i.e., $X_t = 0$ should be interpreted as "do not grant me," the chance that the output mistakenly grants the corresponding input is rather low because when $X_t = 0$, the rank of the corresponding request must be either "00" or "01", which are the two lowest ranks in TABLE 1. On the other hand, if an output incorrectly treats a "do not grant me" as a rank decrease (instead of a rank increases), this will cause a rank tracking error at the output. We argue that the adverse effect is limited because (a) it will only last for two time slots, and (b) the output can still have many other inputs to grant.

Indeed, we have experimented with different ways to further enhance the request phase. For example, when input i has packets for its preferred output j, it sends $X_t = 1$ to output j and regularly codes X_t for all other outputs. This has the advantage of allowing each output to track the VOQ status more accurately. But the disadvantage, which outweighs the advantage, is that the input will attract unnecessary grants from other outputs, yet it can only accept the one from its preferred output.

In the grant phase, each output grants the request from its preferred input with the highest priority. If the corresponding

Algorithm 2 Coded Highest Rank First Algorithm

```
 1: procedure Request($in$, $time$)
 2:     L[$i$] = the length of VOQ $i$ at input $in$
 3:     mark all empty VOQs with rank 0
 4:     mark the longest VOQ with rank 1
 5:     mark other VOQs with rank 2
 6:     H[$i$] = the rank of VOQ $i$ in the last slot
 7:     K[$i$] = the rank of VOQ $i$ in the current slot
 8:     IQ[$i$] = the request of VOQ $i$ in the last slot
 9:     IR[$i$] = Encode(H[$i$], K[$i$], IQ[$i$])
10:     $pout = (in + time) \bmod N$
11:     if L[$pout$] > 0 then
12:         send 1 to output $pout$
13:         send 0 to other outputs
14:     else
15:         send each IR[$i$] to the corresponding output
16:     end if
17:     H = K
18:     IQ = IR
19: end procedure
20: procedure Grant($out$, $time$)
21:     OQ[$i$] = the request from input $i$ in the last slot
22:     OR[$i$] = the request from input $i$ in the current slot
23:     $pin = (out + N - time \bmod N) \bmod N$
24:     if OR[$pin$] == 1 then
25:         send 1 to input $pin$
26:         send 0 to other inputs
27:     else
28:         $m = \arg\max_{i \in [0,N)}$ Decode(OQ[$i$], OR[$i$])
29:         send 1 to input $m$
30:         send 0 to other inputs
31:     end if
32:     OQ = OR
33: end procedure
34: procedure Accept($in$, $time$)
35:     K[$i$] = the rank of VOQ $i$ got from Request
36:     G[$i$] = the grant from output $i$
37:     $pout = (in + time) \bmod N$
38:     if G[$pout$] == 1 then
39:         accept the grant from output $pout$
40:     else
41:         $n = \arg\min_{\mathbf{G}[i]=1} \mathbf{K}[i] \neq 0$
42:         accept the grant from output $n$
43:     end if
44: end procedure
```

Algorithm 3 Encoding & Decoding

```
function Encode($last\_rank$, $cur\_rank$, $last\_req$)
    case $\{last\_rank, cur\_rank\}$
        when $\{0, 0\}$ then return 0
        when $\{0, 1\}$ then return 1
        when $\{0, 2\}$ then return 1
        when $\{1, 0\}$ then return 0
        when $\{1, 1\}$ then return 1
        when $\{1, 2\}$ then return 0
        when $\{2, 0\}$ then return 0
        when $\{2, 1\}$ then return 0
        when $\{2, 2\}$ then
            if $last\_req = 0$ then
                return 1
            else
                return 0
            end if
    end case
end function
function Decode($last\_req$, $cur\_req$)
    case $\{last\_req, cur\_req\}$
        when $\{0, 0\}$ then return 0
        when $\{1, 0\}$ then return 1
        when $\{0, 1\}$ then return 2
        when $\{1, 1\}$ then return 3
    end case
end function
```

VOQ of its preferred input is empty, the output grants the request with the highest rank. Notably, the strict granting order above may limit the performance of CHRF under some nonuniform and heavy traffic conditions. This is because for nonuniform traffic, rank-based arbitration is usually a better choice. But if the traffic is heavy, the embedded round-robin scheduler tends to dominate. Therefore, when the traffic is nonuniform and heavy, we may occasionally grant a request based on the highest rank first and the preferred relationship next. We leave this for possible future work.

In the accept phase, an input always accepts the grant from its preferred output first. If there is no preferred grant, the input accepts the grant with the highest rank.

E. TIME COMPLEXITY

Basic-HRF algorithm and HRF algorithm require each input to rank/sort N VOQs based on queue sizes (line 8 in Algorithm 1). The time complexity of common sort algorithms (e.g., quicksort) is $O(N \log N)$. In practice, the ranking process can be much simpler. This is because any input port receives and sends at most one packet in each time slot (due to the nature of input-queued switches). The ranked list can be stored in a balanced binary search tree [26]. When a new packet arrives at VOQ(i,j), its queue size is increased by 1, and other VOQs remain unchanged. Then the new rank of VOQ(i,j) becomes higher or remains unchanged. The time

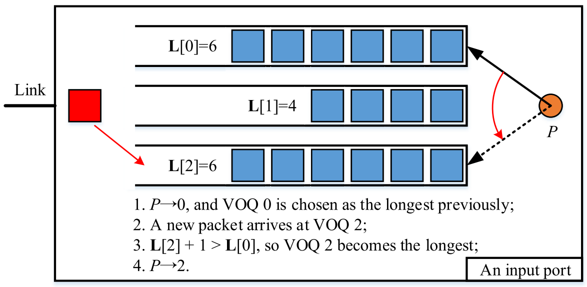
Link
L[0]=6
L[1]=4
L[2]=6
P
1. P→0, and VOQ 0 is chosen as the longest previously;
2. A new packet arrives at VOQ 2;
3. L[2] + 1 > L[0], so VOQ 2 becomes the longest;
4. P→2.
An input port

FIGURE 4. Fast algorithm for CHRF.

complexity for finding a new rank in a binary tree is $O(\log N)$. The same procedure can be adopted for packet departures. The overall time complexity of Basic-HRF or HRF is thus reduced to $O(\log N)$.

In CHRF algorithm, only three ranks (“empty,” “others” and “longest”) are to be differentiated. So the ranking process can be further simplified. In particular, each input only needs to maintain a pointer to the longest VOQ (line 4 in Algorithm 2). But finding the longest VOQ (in an unsorted list) has a worst-case time complexity of $O(N)$. To reduce its complexity, we can adopt the thinking of quasi-LQF [27]. As shown in FIGURE 4, a pointer P points to the VOQ which is chosen as the longest. On the arrival of a new packet at VOQ(i,j), the queue sizes of VOQ(i,j) and VOQ(i,P) are compared (in this example, $j = 2$ and $P = 0$). P is then updated to the longer one (i.e., VOQ 2). Note that there is only one VOQ chosen as the longest one, even if several VOQs have the same maximum size. As a result, it only involves one comparison and (at most) one pointer move when a packet is received. No comparison is needed at the departure of packets, because we only need to maintain the longest queue. The complexity of CHRF is thus reduced to $O(1)$. Of course, the quasi-LQF may produce a sub-optimal result due to mistracking the longest queue occasionally [27]. But CHRF can still achieve the same match size because the quasi-longest VOQ is always a nonempty one. It is worth reducing the complexity from $O(N)$ to $O(1)$ by using the quasi-LQF, as the value of N is becoming larger and larger.

V. PERFORMANCE EVALUATION

In this section, we quantitatively study the performance of Basic-HRF, HRF, and CHRF by simulations. We compare them with other single-iteration algorithms (including some multiple-iteration algorithms executed for one iteration):

- SRR [13]: the single-bit-single-iteration algorithm that employs the preferred input-output relationships.
- π-RGA [14]: a multi-bit-single-iteration algorithm aiming to reuse the match obtained in the previous time slots. (The name “pi-RGA” is used in FIGURE 5–FIGURE 8.)
- iSLIP-1: a single-bit-single-iteration algorithm by executing the classic iSLIP [17] for one iteration.
- iLQF-1: a multi-bit-single-iteration algorithm by executing iLQF [16] for one iteration.

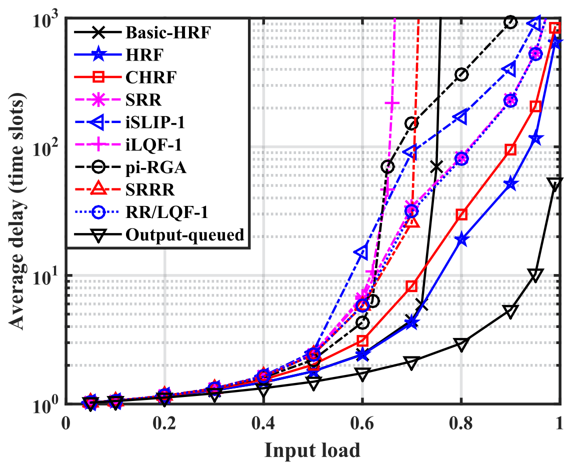

FIGURE 5. Delay vs. input load, under uniform balanced traffic.

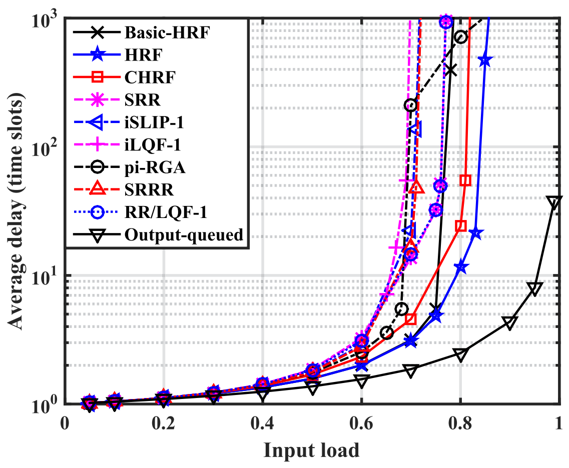

FIGURE 7. Delay vs. input load, under hotspot output traffic.

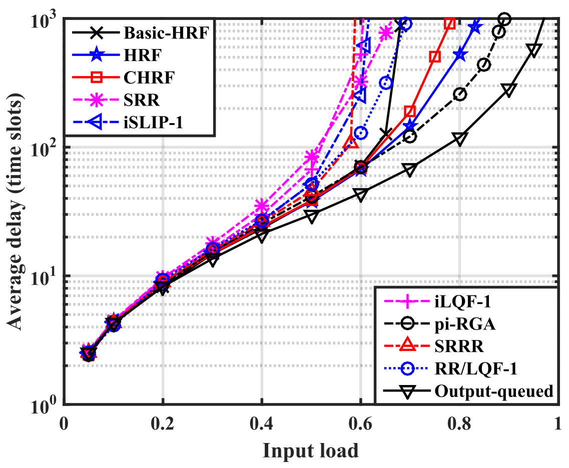

FIGURE 6. Delay vs. input load, under uniform bursty traffic.

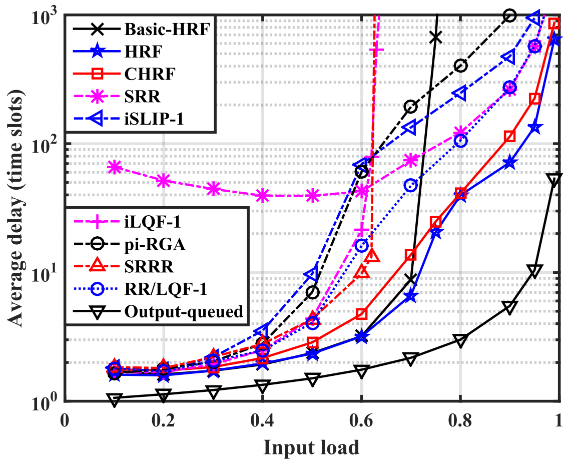

FIGURE 8. Delay vs. input load, under hotspot input traffic.

- SRRR [21]: a 4-phase algorithm where the extra phase is for an output to inform an input if the input is pointed to by the RR pointer at this output.
- RR/LQF-1: the newest single-bit-single-iteration algorithm by executing RR-LQF [24] for one iteration.
- Output-queued switch is also simulated for providing a lower bound on delay performance.

For CHRF, the request coding scheme with three ranks (see FIGURE 2 and TABLE 1) is adopted. Being a single-bit-single-iteration algorithm, we focus on comparing CHRF with four other single-bit-single-iteration algorithms SRR, iSLIP-1, SRRR and RR/LQF-1. Four types of traffic patterns [16], [17] are simulated, *uniform balanced* traffic, *uniform bursty* traffic, *hotspot output* traffic and *hotspot input* traffic. For any traffic, if the average number of packets destining to each output in each time slot is not more than 1 in the long run, we say it is admissible. Of course, if the traffic is not admissible, buffer overflow will happen no matter what scheduling algorithm is used, because each output can accept at most 1 packet in each time slot. Therefore, we only consider admissible traffics. We present the simulation results for a switch with size $N = 64$ below, and the same conclusions and observations apply to other sizes. The simulator warms up with 100,000 slots and runs another 100,000 slots for statistics. Note that the minimum delay of a packet is one time slot and input buffers are set large enough to avoid any instantaneous buffer overflow.

A. UNIFORM BALANCED TRAFFIC

Uniform balanced traffic is generated as follows. In each time slot, a packet arrives at each input with probability p (i.e., the input load) and destines to each output with equal probability. From FIGURE 5, we can see that Basic-HRF outperforms iLQF-1. The significant performance gain when $p > 0.5$ is due to rank-based arbitration being better than queue-based arbitration. When $p < 0.7$, HRF has almost identical (and the best) performance as Basic-HRF, which shows that highest rank first is an efficient mechanism. When $p > 0.7$, the embedded global round-robin scheduler (via the preferred input-output relationship) in HRF kicks in, which warrants HRF's best delay-throughput performance among all single-iteration algorithms.

As compared to HRF, CHRF cuts down on the request size from 7 bits (for $N = 64$) to a single bit at the cost of slightly increasing the delay. This confirms that (a) the fine granularity of full $(N + 1)$ ranks is not necessary, and (b) the request encoding/decoding scheme based on three ranks (in FIGURE 2 and TABLE 1) is efficient.

As compared to other single-bit-single-iteration algorithms, namely, SRR, iSLIP-1, SRRR and RR/LQF-1, CHRF gives the best delay-throughput performance. Note that the global round-robin scheduler (implemented via the preferred input-output relationship) is embedded in all three algorithms, i.e., SRR, RR/LQF-1 and CHRF. As a result, all of them give comparable heavy load performance (for $p > 0.95$). Due to the use of rank-based arbitration, CHRF outperforms both RR/LQF-1 and SRR. As an example, at $p = 0.8$ the average delay of RR/LQF-1 is 79.4 time slots, while that of CHRF is only 28.7 time slots, cutting down the delay by more than 2 times.

It is also worth noting that despite its complexity, π-RGA, a multi-bit-single-iteration algorithm, does not perform well under uniform balanced traffic, and its delay performance increases sharply when p is just above 0.6.

B. UNIFORM BURSTY TRAFFIC

Bursty arrivals are characterized by the ON/OFF traffic model, which is a special instance of the two-state Markov process. It has been shown that the composition of ON/OFF processes gives rise to long-range dependent (LRD) traffic. In the ON state, a packet arrival is generated in every time slot. In the OFF state, there are no packet arrivals. Packets of the same burst have the same destination output and the destination for each burst is uniformly distributed. Given average input load p and burst size s, the state transition probability from OFF to ON is $p/[s(1-p)]$ and that from ON to OFF is $1/s$. In our simulations, we set burst size $s = 30$ packets.

From FIGURE 6, we can see that, unlike the performance of uniform balanced traffic in FIGURE 5, the delay builds up quickly with input load for all scheduling algorithms. Nevertheless, Basic-HRF is still consistently better than iLQF-1, and CHRF always outperforms SRR and RR/LQF-1. When $p = 0.6$, the average packet delays of using SRR, RR/LQF-1 and CHRF are 83.9, 50.7 and 37.4 slots, respectively.

In FIGURE 6, CHRF outperforms π-RGA if $p \leq 0.6$. When $p > 0.6$, π-RGA performs better as it can more accurately target at scheduling bursty flows first. But π-RGA requires multi-bit requests and a more sophisticated arbitration mechanism.

C. HOTSPOT OUTPUT TRAFFIC

We assume that, in each time slot, packets arriving at each input port follow the same independent Bernoulli process with probability p. Hotspot outputs are generated as follows. For any input i, a packet goes to output $(i+N/2) \bmod N$ with probability 0.5, and other outputs with the same probability $1/[2(N-1)]$. Since each input has a "hotspot" output, this traffic pattern is unbalanced and nonuniform [16].

From FIGURE 7 and as expected, Basic-HRF yields much better performance than iLQF-1, and CHRF defeats SRR, iSLIP-1, SRRR and RR/LQF-1. This shows that the rank-based arbitration (in CHRF) is suitable for unbalanced traffic. Further note that CHRF outperforms π-RGA over almost all input loads simulated.

D. HOTSPOT INPUT TRAFFIC

Hotspot input traffic is generated as follows. One input port is always 100% loaded. Without loss of generality, let input 0 be the fully-loaded input port. The packet arrival rate to other inputs, or p, varies from 0.1 to 1. For each arrived packet, its destination is chosen randomly from all outputs. From the simulation results in FIGURE 8, we can see that, again, Basic-HRF outperforms iLQF-1 and CHRF outperforms SRR and RR/LQF-1. CHRF also outperforms the more sophisticated π-RGA under all traffic load. As an example, at $p = 0.8$, the packet delay obtained using π-RGA is 402 time slots, while that of CHRF is only 40 time slots, cutting down the delay 10 times.

It is interesting to note the "odd" delay performance of SRR: large delay (> 30 slots) when p is small, and the delay drops slightly as p increases toward 0.5. This is because input 0 is always fully loaded. When p (for other inputs) is small, the switch delay performance is dominated by the delay of packets from input 0. It is because SRR only allows each input to send at most one 1-request. When the preferred VOQ of input 0 is empty and the other VOQs of input 0 are backlogged, input 0 can (still) only send a single 1-request to its longest VOQ. But this 1-request will fail if the corresponding output has received a 1-request from its preferred input, or the output has randomly chosen another 1-request to grant. As a result, when p is small, the throughput of input 0 will be much lower than that of using an algorithm allowing multiple 1-requests, e.g., RR/LQF. Accordingly, the packets from input 0 have to wait for a longer time, resulting in higher overall delay when p is small.

In summary, under all four traffic patterns, *uniform*, *bursty*, *hotspot output*, and *hotspot input*, CHRF yields the best delay-throughput performance among all single-bit-single-iteration algorithms, namely, SRR, iSLIP-1, SRRR and RR/LQF-1. It even outperforms the multi-bit algorithm, π-RGA, under most traffic patterns.

VI. ANALYTICAL MODEL FOR DELAY PERFORMANCE

In this section, an analytical model is constructed to study the delay performance of HRF under uniform and low traffic load. We focus on HRF algorithm because CHRF algorithm has similar performance, yet is much more difficult to be modeled. In this section, we are dedicated to the analysis of delay. The analysis of throughput is provided in Appendix B.

When packets arrive uniformly and infrequently, it is reasonable to assume that all input ports are initially empty. Then if a VOQ receives a new packet, it becomes the longest queue of its input. Without loss of generality, suppose that at time slot 0, a packet A arrives at input i and destines to output j. VOQ(i,j) then becomes the longest queue at input i. Let P_t^A be the probability that packet A is sent at time slot t. From (1), VOQ(i,j) is preferred (by the embedded round-robin scheduler) with the highest priority exactly once every N slots. So t is upper bounded by N. Since the cut-through (packet arriving and leaving within the same time slot) is not allowed, t is lower bounded by 1. Therefore, t falls into the interval $[1, N]$ and

$$\sum_{t=1}^{N} P_t^A = 1. \tag{2}$$

The average delay of packet A can be written as

$$E(A) = \sum_{t=1}^{N} t \cdot P_t^A. \tag{3}$$

To calculate (3), we first follow HRF to derive the probability that packet A is transmitted at slot $t = 1$, i.e. P_1^A. When $t = 1$, packet A can be sent under two cases: (a) VOQ(i,j) is preferred with the highest priority for transmission; (b) VOQ(i,j) is selected as a non-preferred VOQ for transmission. The probability of case (a) is simply $1/N$ due to (1). The second case (b) occurs only when VOQ(i,j) is not preferred, output j's preferred VOQ (say, VOQ(k,j)) is empty and simultaneously output j grants input i. Note that VOQ(i,j) is not preferred with the probability $(1 - 1/N)$. At slot t, denote P_t^p the probability that output j's preferred VOQ(k,j) is empty, and Y_t the expected number of the highest rank-1 requests received by output j. Since output j randomly issues one grant among the highest rank requests, the probability that output j grants input i is $1/Y_t$. In case (b), packet A is sent with probability $(1 - 1/N)P_1^p/Y_1$.

Combining cases (a) and (b) together, the probability that packet A is transmitted at time slot 1 is

$$P_1^A = \frac{1}{N} + \left(1 - \frac{1}{N}\right)\frac{P_1^p}{Y_1}. \tag{4}$$

To get P_1^A, we next calculate P_1^p and Y_1. Recall that all VOQs are assumed to be empty at the beginning of slot 0. Then output j's preferred VOQ(k,j) will be empty at the beginning of slot 1 unless input k receives a new packet destining to output j at slot 0. P_1^p, the probability that VOQ(k,j) is empty at slot 1, is given by

$$P_1^p = 1 - \frac{\lambda}{N}, \tag{5}$$

where λ $(0 \leq \lambda \leq 1)$ is the offered load at input port k (as well as all other inputs under uniform traffic).

Although output j's preferred VOQ(k,j) is empty and the request coming from VOQ(i,j) is of the highest rank at slot 1, it is still possible that output j randomly grants other highest-rank request. This probability is determined by Y_1, the total number of highest rank requests received by output j at $t = 1$. Based on our assumption that a nonempty VOQ is in fact the longest queue of its input at slot 1, the number of highest rank requests can be simplified to be the number of nonempty VOQs. Then Y_1 is

$$Y_1 = \frac{\lambda(N-2)}{N} + 1, \tag{6}$$

where "$N-2$" is to disregard the confirmed empty VOQ(k,j) and nonempty VOQ(i,j) from the number of VOQs that may become nonempty from empty with probability λ/N, and "+1" is to compensate for the excluded but definitely nonempty VOQ(i,j). Substituting (5) and (6) into (4), we have

$$P_1^A = \frac{1}{N} + \frac{(N-1)(N-\lambda)}{N^2 + \lambda N(N-2)}. \tag{7}$$

Up to now, we can get the value of P_1^A from (7). But to calculate (3), we still need to follow HRF to derive the probability that packet A is transmitted at slot 2, i.e. P_2^A. If packet A is not transmitted at time slot 1 (with probability $1 - P_1^A$), at slot 2, it can be sent also under two cases that VOQ(i,j) is preferred or non-preferred by output j. Note that we have known that VOQ(i,j) is not preferred by output j at slot 1. (Otherwise, A would have been sent at slot 1 for sure.) Then VOQ(i,j) will be served exactly once in the subsequent $(N-1)$ slots due to (1). P_2^A is written as

$$P_2^A = \left(1 - P_1^A\right)\left[\frac{1}{N-1} + \left(1 - \frac{1}{N-1}\right)\frac{P_2^p}{Y_2}\right]. \tag{8}$$

To get P_2^A from (8), we next calculate P_2^p and Y_2. Without loss of generality, let output j prefer VOQ(m,j) ($m \neq i$) at slot 2. P_2^p is then the probability that VOQ(m,j) is empty at slot 2. To ensure that, firstly no packet should arrive at slot 1 (with probability $1 - \lambda/N$), and secondly there are two possible scenarios to consider: (a) no packet arrived (with probability $1 - \lambda/N$) at slot 0; or (b) a packet arrived (with probability λ/N) at slot 0 but left at slot 1. So

$$P_2^p = \left(1 - \frac{\lambda}{N}\right)^2 + \frac{\lambda}{N}\left(1 - \frac{\lambda}{N}\right)P_{1,1}^p, \tag{9}$$

where $P_{r,l}^p$ is defined as the joint probability that a packet has been buffered at the currently preferred VOQ for l slots and is sent at slot r.

To calculate P_2^p in (9), we first need to derive $P_{1,1}^p$. Since VOQ(m,j) is output j's preferred VOQ at time slot 2, it is not preferred by output j at slot 1 for sure. Then VOQ(m,j) has only one chance to be served at slot 1, i.e. as a non-preferred VOQ.

$$P_{1,1}^p = \frac{P_1^p}{W_1}, \tag{10}$$

where W_1 is the total number of highest-rank requests competing for output j's grant at $t = 1$. Note that W_1 above is not the same as Y_1 in (6). In particular, the confirmed empty VOQ(k,j), nonempty VOQ(i,j) and VOQ(m,j) should be removed from the number of VOQs that may become nonempty from empty with probability λ/N. Therefore,

$$W_1 = \frac{\lambda(N-3)}{N} + 1. \tag{11}$$

With (9), (10) and (11), the value of P_2^p can be derived. To get P_2^A from (8), we only lack Y_2. Let P_r^n denote the probability that the current non-preferred VOQ is empty at slot r. Then the probability that a non-preferred VOQ is not empty at slot 2 is $(1 - P_2^n)$. Like Y_1 in (6), we can write Y_2 as

$$Y_2 = (N-2)(1-P_2^n) + 1. \qquad (12)$$

P_2^n is the probability that the current non-preferred VOQ is empty at slot 2. To ensure that, first, no packet should arrive at slot 1 (with probability $1 - \lambda/N$), and second, there are two possible scenarios to consider: (a) no packet arrived (with probability $1 - \lambda/N$) at slot 0; or (b) a packet arrived (with probability λ/N) at slot 0 but left at slot 1. Define $P_{r,l}^n$ as the joint probability that a packet has been buffered in a currently non-preferred VOQ for l slots and is sent at slot r. Then

$$P_2^n = \left(1-\frac{\lambda}{N}\right)^2 + \frac{\lambda}{N}\left(1-\frac{\lambda}{N}\right)P_{1,1}^n. \qquad (13)$$

We can see that when $P_{1,1}^n$ is derived, P_2^n as well as Y_2 can be got. Without loss of generality, let VOQ(n,j) be output j's non-preferred VOQ at slot 2. It is still possible that VOQ(n,j) was preferred by output j at slot 1. Therefore, VOQ(n,j) has two opportunities to be served at slot 1, i.e., as a preferred or non-preferred VOQ. The probability that VOQ(n,j) is preferred by output j at slot 1 is $1/(N-2)$. Here "$N-2$" is to exclude VOQ(m,j) and VOQ(i,j) (not preferred by output j at slot 1 for sure) from the total number of VOQs that could be preferred with probability $1/N$. If VOQ(n,j) is not preferred by output j at slot 1, it may still be selected for transmission. It occurs only when output j's preferred VOQ(k,j) is empty and simultaneously output j grants input n. The probability of this case is determined by P_1^p and W_1. So

$$P_{1,1}^n = \frac{1}{N-2} + \left(1-\frac{1}{N-2}\right)\frac{P_1^p}{W_1}. \qquad (14)$$

Combine (8)-(14), and we have

$$P_2^A = \left(1-P_1^A\right)Q_2^A, \qquad (15)$$

where

$$Q_2^A = \frac{1}{N-1} + \frac{\frac{N-\lambda}{N-1}\left[1+\frac{\lambda}{N+\lambda(N-3)}\right]}{\frac{N^2(N-1)}{(N-2)(N-\lambda)} - N + \lambda - \frac{\lambda N}{N+\lambda(N-3)}}. \qquad (16)$$

We can see that as long as the switch size N and input loading λ are given, P_2^A can be derived from P_1^A. In the general case, P_t^A $(2 \leq t \leq N-1)$ can be derived recursively. We provide the general results in Appendix A. When we get all P_t^A $(1 \leq t \leq N-1)$, P_N^A can be obtained using constraint (3). The expected delay of Packet A, i.e., $E(A)$ in (3), is straightforward.

Recall that the input load λ is assumed to be light in our analytical model. Then packet A will be transmitted with a very short queuing delay. In other words, when $t \geq 4$, P_t^A is quite small and has negligible impact on $E(A)$. For example, when $N = 64$ and $\lambda = 0.1$, $P_1^A \approx 0.912$ and $P_2^A \approx 0.080$.

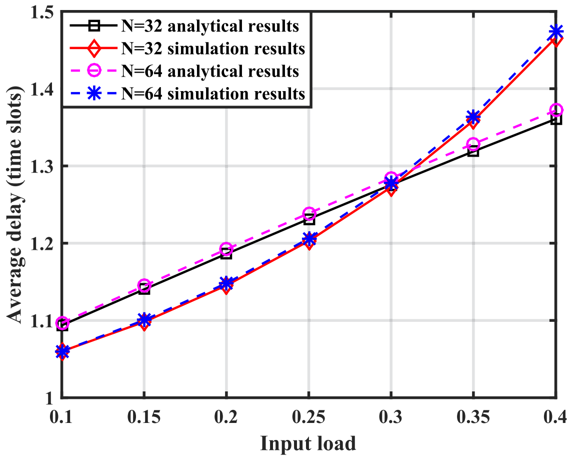

FIGURE 9. **Analytical results under low uniform offered load.**

Thus $\sum_{t=3}^{N} P_t^A = 1 - P_1^A - P_2^A \approx 0.008$ only. Motivated by this observation, we can simplify the calculation in (3) by

$$E(A) \approx P_1^A + 2P_2^A + 3(1 - P_1^A - P_2^A). \qquad (17)$$

Combining (7), (15), (16) and (17), the expected delay of packet A can be approximated by

$$E(A) \approx 3 - \frac{2(N^2-\lambda) + \lambda(N-1)^2 Q_2^A}{N^2 + \lambda N(N-2)}. \qquad (18)$$

To see how tight the value in (18) is, the analytical and simulation results are compared in FIGURE 9. We can see that, with light traffic, the simulation results almost overlap with the analytical results.

VII. CONCLUSION

In this paper, we first demonstrated the effectiveness of rank-based arbitration by a simple algorithm called Basic-HRF. In Basic-HRF, VOQs at an input port are ranked according to their queue sizes. The rank of a VOQ, coded by $\log(N+1)$ bits where N is the switch size, is sent to the corresponding output as a request. In both grant and accept phases, the request/grant with the highest rank is given the highest priority. We showed that such a rank-based arbitration outperforms the widely adopted queue-based arbitration. To improve the heavy load performance of Basic-HRF, the rank-based arbitration was integrated with an embedded global round-robin scheduler. The refined Basic-HRF algorithm, or HRF algorithm in short, performs better than almost all existing single-iteration algorithms. But its implementation complexity is higher than the class of single-bit-single-iteration algorithms due to the use of multi-bit requests. A novel request encoding/decoding mechanism was then designed to reduce the request size to a single bit. Unlike all existing iterative scheduling algorithms, a single-bit request is used to indicate the increase or decrease of a VOQ rank, rather than a VOQ is empty or not. Extensive simulation results showed that the resulting algorithm Coded Highest Rank First (CHRF) provides the best delay-throughput performance among all single-bit-single-iteration algorithms.

APPENDIX A
GENERAL RESULTS FOR PACKET DELAY

From Section VI, we can see that as long as the switch size N and input loading λ are given, P_2^A can be derived from P_1^A, P_3^A can be derived from P_2^A and P_1^A, and so on. In the general case, P_t^A ($2 \leq t \leq N-1$) can be recursively calculated as follows.

$$P_t^A = \left(1-\sum_{q=1}^{t-1} P_q^A\right)\left[\frac{1}{N+1-t}+\left(1-\frac{1}{N+1-t}\right)\frac{P_t^p}{Y_t}\right]. \tag{19}$$

Like (5) and (6), P_t^p is the probability that VOQ(k,j) is empty at slot t, and Y_t is the total number of highest rank requests received by the output at slot t. We have

$$Y_t = (N-2)\left(1-P_t^n\right)+1, \tag{20}$$

and

$$\begin{aligned}P_t^p &= \left(1-\frac{\lambda}{N}\right)^t + \frac{\lambda}{N}\left(1-\frac{\lambda}{N}\right)^{t-1}\sum_{r=1}^{t-1}\sum_{l=1}^{r} P_{r,l}^p \\ &+ \left(\frac{\lambda}{N}\right)^2\left(1-\frac{\lambda}{N}\right)^{t-2}\sum_{r=1}^{t-2}\sum_{l=1}^{r}\sum_{u=r+1}^{t-1}\sum_{v=1}^{u-r+l} P_{r,l}^p P_{u,v}^p \\ &+ \left(\frac{\lambda}{N}\right)^3\left(1-\frac{\lambda}{N}\right)^{t-3} \\ &\times \sum_{r=1}^{t-3}\sum_{l=1}^{r}\sum_{u=r+1}^{t-2}\sum_{v=1}^{u-r+l}\sum_{x=u+1}^{t-1}\sum_{y=1}^{x-u+v} P_{r,l}^p P_{u,v}^p P_{x,y}^p \\ &+ \ldots + \left(\frac{\lambda}{N}\right)^{t-1}\left(1-\frac{\lambda}{N}\right)\sum_{r=1}^{t-1} P_{r,1}^p. \end{aligned} \tag{21}$$

To calculate (21), we need the value of $P_{t,l}^p$. It is given by

$$P_{t,l}^p = \left(1-\sum_{v=1}^{l-1} P_{t-l+v,v}^p\right)\frac{P_t^p}{W_t}, \tag{22}$$

where

$$W_t = (N-3)\left(1-P_t^n\right)+1. \tag{23}$$

To calculate Y_t and W_t, we need to determine the value of P_t^n as follows.

$$\begin{aligned}P_t^n &= \left(1-\frac{\lambda}{N}\right)^t + \frac{\lambda}{N}\left(1-\frac{\lambda}{N}\right)^{t-1}\sum_{r=1}^{t-1}\sum_{l=1}^{r} P_{r,l}^n \\ &+ \left(\frac{\lambda}{N}\right)^2\left(1-\frac{\lambda}{N}\right)^{t-2}\sum_{r=1}^{t-2}\sum_{l=1}^{r}\sum_{u=r+1}^{t-1}\sum_{v=1}^{u-r+l} P_{r,l}^n P_{u,v}^n \\ &+ \left(\frac{\lambda}{N}\right)^3\left(1-\frac{\lambda}{N}\right)^{t-3} \\ &\times \sum_{r=1}^{t-3}\sum_{l=1}^{r}\sum_{u=r+1}^{t-2}\sum_{v=1}^{u-r+l}\sum_{x=u+1}^{t-1}\sum_{y=1}^{x-u+v} P_{r,l}^n P_{u,v}^n P_{x,y}^n \\ &+ \ldots + \left(\frac{\lambda}{N}\right)^{t-1}\left(1-\frac{\lambda}{N}\right)\sum_{r=1}^{t-1} P_{r,1}^n. \end{aligned} \tag{24}$$

The joint probability that a packet has been buffered in a non-preferred VOQ for l slots and is sent at slot t is given by

$$\begin{aligned}P_{t,l}^n &= \left(1-\sum_{v=1}^{l-1} P_{t-l+v,v}^n\right)\left[\frac{1}{N-1-l}\right. \\ &\left. + \left(1-\frac{1}{N-1-l}\right)\frac{P_t^p}{W_t}\right]. \end{aligned} \tag{25}$$

APPENDIX B
THROUGHPUT AND STABILITY PROOF

If multiple iterations are allowed, with HRF algorithm, there is at least one match among all the unmatched inputs/outputs from the last iteration. Thus, maximal size matching will be achieved after N iterations at most. In other words, with a speedup 2, the multi-iteration version of HRF is stable if the traffic is admissible, no matter what the traffic pattern is [26]. In this appendix, we are devoted to analyzing the throughput of HRF with single iteration and no speedup, and provide the sufficient condition of stability for single-iteration HRF algorithm. If not specified, the switch buffer is assumed to be large enough.

A. A FLUID MODEL

For clarity, we use the term HRF to represent both HRF and CHRF. Following the approach in [28] and [29], we first construct a fluid model for HRF. Let λ_{ij} be the mean packet arrival rate to VOQ(i,j). A traffic pattern/matrix is admissible if

$$\sum_i \lambda_{ij} \leq 1, \sum_j \lambda_{ij} \leq 1. \tag{26}$$

Let $Z_{ij}(n)$ denote the number of packets in VOQ(i,j) at the beginning of time slot n. Further let $A_{ij}(n)$ and $D_{ij}(n)$ denote the cumulative number of packet arrivals and departures for VOQ(i,j) at the beginning of time slot n respectively. We have

$$Z_{ij}(n) = Z_{ij}(0) + A_{ij}(n) - D_{ij}(n), \tag{27}$$

where $n \geq 0$ and $i, j \in \{0, \ldots, N-1\}$.

Assume that the packet arrival process obeys the strong law of large numbers with probability one, i.e.,

$$\lim_{n\to\infty}\frac{A_{ij}(n)}{n} = \lambda_{ij}, \quad i, j \in \{0, \ldots, N-1\}.$$

The switch is, by definition, rate stable if

$$\lim_{n\to\infty}\frac{D_{ij}(n)}{n} = \lambda_{ij}, \quad i, j \in \{0, \ldots, N-1\}.$$

If a switch is rate stable for an admissible traffic matrix, then it delivers 100% throughput.

The fluid model is determined by a limiting procedure illustrated below. First, the discrete functions are extended to *right continuous* functions. For arbitrary time $t \in [n, n+1)$, we define

$$A_{ij}(t) = A_{ij}(n), \tag{28}$$
$$Z_{ij}(t) = Z_{ij}(n), \tag{29}$$

and

$$D_{ij}(t) = D_{ij}(n) + (t-n)(D_{ij}(n+1) - D_{ij}(n)). \quad (30)$$

Note that all functions are random elements of the set $\{0, 1, 2, \ldots\}$. We shall sometimes use the notation $A_{ij}(\cdot, \omega)$, $Z_{ij}(\cdot, \omega)$ and $D_{ij}(\cdot, \omega)$ to explicitly denote the dependency on the sample path ω. For a fixed ω, at time t, we have [28]

- $A_{ij}(t, \omega)$: the cumulative number of arrivals to VOQ(i,j);
- $Z_{ij}(t, \omega)$: the number of packets in VOQ(i,j);
- $D_{ij}(t, \omega)$: the cumulative number of departures from VOQ(i,j).

For each $r > 0$, we define

$$\bar{A}^r_{ij}(t, \omega) = r^{-1}A_{ij}(rt, \omega), \quad (31)$$
$$\bar{Z}^r_{ij}(t, \omega) = r^{-1}Z_{ij}(rt, \omega), \quad (32)$$

and

$$\bar{D}^r_{ij}(t, \omega) = r^{-1}D_{ij}(rt, \omega). \quad (33)$$

It is shown in [29] that for each fixed ω satisfying (27) and any sequence $\{r_n\}$ with $r_n \to \infty$ as $n \to \infty$, there is a subsequence $\{r_{n_k}\}$ and the continuous functions $(\bar{A}_{ij}(\cdot), \bar{Z}_{ij}(\cdot), \bar{D}_{ij}(\cdot))$, where $(\bar{A}^r_{ij}(t, \omega), \bar{Z}^r_{ij}(t, \omega), \bar{D}^r_{ij}(t, \omega))$ converges as

$$\bar{A}^{r_{n_k}}_{ij}(t, \omega) \to \lambda_{ij} \cdot t, \quad (34)$$
$$\bar{Z}^{r_{n_k}}_{ij}(t, \omega) \to \bar{Z}_{ij}(t), \quad (35)$$

and

$$\bar{D}^{r_{n_k}}_{ij}(t, \omega) \to \bar{D}_{ij}(t), \quad (36)$$

uniformly on compacts as $k \to \infty$ for any $t \geq 0$.

Definition 1: Any function obtained through the limiting procedures in (34), (35) and (36) is said to be a *fluid limit* of the switch. The fluid model equation using HRF is

$$\bar{Z}_{ij}(t) = \bar{Z}_{ij}(0) + \lambda_{ij}t - \bar{D}_{ij}(t), \quad t \geq 0. \quad (37)$$

Definition 2: The fluid model of a switch operating under a scheduling algorithm is said to be weakly stable if for every fluid model solution $(\bar{D}, \bar{Z})$ with $\bar{Z}(0) = 0$, $\bar{Z}(t) = 0$ for almost every $t \geq 0$.

B. STABILITY PROOF

From [28], a switch is rate stable if its corresponding fluid model is weakly stable. Our goal here is to prove that for every fluid model solution $(\bar{D}, \bar{Z})$ using HRF, $\bar{Z}(t) = 0$ for almost every $t \geq 0$. Specifically, we will use **Fact 1** from [29].

Fact 1: Let f be a non-negative, absolutely continuous function defined on $\mathbb{R}^+ \cup \{0\}$ with $f(0) = 0$. Assume that for almost every t, $f(t) > 0$ and $f'(t) \leq 0$. Then $f(t) = 0$ for almost every $t \geq 0$.

Note that $\mathbb{R}^+$ is the set of positive real numbers and $f'(t)$ denotes the derivative of function f at time t.

In the following theorem, we show the sufficient condition for 100% throughput of HRF.

Theorem 1: (*Sufficiency*) When $\lambda_{ij} \leq 1/N$ (for all $i, j \in \{0, 1, \ldots, N-1\}$), HRF can achieve 100% throughput.

Proof: Define $\mathbf{B} \triangleq \{m : \bar{Z}_{im}(t) > 0\}$. Let $G_i(t)$ denote the joint queue occupancy of all nonempty VOQs at input port i. We have

$$G_i(t) = \sum_{m \in \mathbf{B}} \bar{Z}_{im}(t). \quad (38)$$

Because $\bar{Z}(t)$ is a non-negative and absolutely continuous function, from (38), $G_i(t)$ is also non-negative and absolutely continuous. Without loss of generality, assume all VOQs are initially empty, i.e., $\bar{Z}(0) = 0$. Then $G_i(0) = 0$ and the derivative of $G_i(t)$ is

$$G'_i(t) = \sum_{m \in \mathbf{B}} \bar{Z}'_{im}(t).$$

Combine the above equation with (37), and we get

$$G'_i(t) = \sum_{m \in \mathbf{B}} \lambda_{im} - \sum_{m \in \mathbf{B}} \bar{D}'_{im}(t).$$

From the condition $0 \leq \lambda_{ij} \leq 1/N$ (for all $i, j \in \{0, 1, \ldots, N-1\}$),

$$G'_i(t) \leq \frac{h}{N} - \sum_{m \in \mathbf{B}} \bar{D}'_{im}(t), \quad (39)$$

where $h = \|\mathbf{B}\| \geq 0$.

Suppose that $G_i(t) > 0$ when $t > 0$. This implies that $\forall m_1 \in \mathbf{B}$ and $\forall m_2 \notin \mathbf{B}$, $\bar{Z}_{im_1}(t) > 0$ and $\bar{Z}_{im_2}(t) = 0$. Then $\bar{Z}_{im_1}(t) - \bar{Z}_{im_2}(t) > 0$. By the continuity of these functions, $\exists \delta$ such that

$$\min_{t' \in [t, t+\delta]} \bar{Z}_{im_1}(t') - \bar{Z}_{im_2}(t') > 0, \quad \forall m_1 \in \mathbf{B}, \ \forall m_2 \notin \mathbf{B}.$$

Let

$$q = \min_{\substack{m_1 \in \mathbf{B} \\ m_2 \notin \mathbf{B}}} \min_{t' \in [t, t+\delta]} \{\bar{Z}_{im_1}(t') - \bar{Z}_{im_2}(t')\}.$$

Thus for a large enough k, we have $\bar{Z}^{r_{n_k}}_{im_1}(t') - \bar{Z}^{r_{n_k}}_{im_2}(t') \geq q/2$, where $\forall m_1 \in \mathbf{B}, \forall m_2 \notin \mathbf{B}$, and $t' \in [t, t+\delta]$. Also for a large enough k, we have $r_{n_k} \cdot q/2 \geq 1$. Thus $Z_{im_1}(t') - Z_{im_2}(t') \geq 1$, where $\forall m_1 \in \mathbf{B}$, $\forall m_2 \notin \mathbf{B}$, and $t' \in [r_{n_k}t, r_{n_k}(t+\delta)]$. This means that in the long time interval $[r_{n_k}t, r_{n_k}(t+\delta)]$, any nonempty VOQ at input port i belongs to the set $\mathbf{U} \triangleq \{\text{VOQ}(i, m) : m \in \mathbf{B}\}$, and any VOQ that belongs to set $\mathbf{U}$ is nonempty [28], [29]. Since HRF always gives the highest priority to the preferred input-output pairs calculated by (1), during the same time interval, each nonempty VOQ sends at least one packet per N slots. Then in the long time interval $[r_{n_k}t, r_{n_k}(t+\delta)]$, input i sends at least $h = \|\mathbf{B}\|$ packets per N slots. In other words,

$$\sum_{m \in \mathbf{B}} \left[D_{im}(r_{n_k}t') - D_{im}(r_{n_k}t)\right] \geq Lh, \quad (40)$$

where $L \in \mathbb{Z}$, $NL \leq r_{n_k}t' - r_{n_k}t < NL + N$. So we have

$$L > \frac{r_{n_k} \cdot (t'-t)}{N} - 1. \quad (41)$$

Combining (40) with (41), we have

$$\sum_{m\in\mathbf{B}}\left[D_{im}(r_{n_k}t') - D_{im}(r_{n_k}t)\right] > \frac{h\cdot r_{n_k}\cdot(t'-t)}{N} - h.$$

Since $h = \|\mathbf{B}\|$ is within $[0, N]$, its impact is insignificant for fluid limit [28]. Dividing the above equation by r_{n_k} and letting $k \to \infty$, the fluid limit is obtained as:

$$\sum_{m\in\mathbf{B}}\left[\bar{D}_{im}(t') - \bar{D}_{im}(t)\right] > \frac{h\cdot(t'-t)}{N}.$$

Further dividing the above equation by $(t'-t)$ and letting $t' \to t$, the derivative of fluid limit is

$$\sum_{m\in\mathbf{B}}\bar{D}'_{im}(t) > \frac{h}{N}. \tag{42}$$

Combine (39) and (42), and we get

$$G_t'(t) < 0.$$

Based on **Fact 1**, $G_i(t) = 0$ for almost every $t \geq 0$. Due to (38), $\bar{Z}_{im}(t) = 0$ for almost every $t \geq 0$. Then HRF is weakly stable. We proved **Theorem 1** that when $\lambda_{ij} \leq 1/N$ (for all $i, j \in \{0, 1, \ldots, N-1\}$), HRF achieves 100% throughput.

REFERENCES

[1] E. Zahavi, I. Keslassy, and A. Kolodny, "Distributed adaptive routing convergence to non-blocking DCN routing assignments," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 88–101, Jan. 2014.

[2] Z. Cao and S. S. Panwar, "Efficient buffering and scheduling for a single-chip crosspoint-queued switch," *IEEE Trans. Commun.*, vol. 62, no. 6, pp. 2034–2050, Jun. 2014.

[3] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Progressive congestion management based on packet marking and validation techniques," *IEEE Trans. Comput.*, vol. 61, no. 9, pp. 1296–1310, Sep. 2012.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[5] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, 2008.

[6] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, 2009.

[7] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, 2012, p. 17.

[8] A. Singh *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[9] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Trans. Commun.*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.

[10] Y. Tamir and G. L. Frazier, "High-performance multiqueue buffers for VLSI communication switches," in *Proc. 15th Annu. Int. Symp. Comput. Archit.*, May 1988, pp. 343–354.

[11] J. Chao, "Saturn: A terabit packet switch using dual round robin," *IEEE Commun. Mag.*, vol. 38, no. 12, pp. 78–84, Dec. 2000.

[12] G. Damm, J. Blanton, P. Golla, D. Verchère, and M. Yang, "Fast scheduler solutions to the problem of priorities for polarized data traffic," in *Proc. Int. Symp. Telecommun.*, 2001, pp. 1–4.

[13] A. Scicchitano, A. Bianco, P. Giaccone, E. Leonardi, and E. Schiattarella, "Distributed scheduling in input queued switches," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2007, pp. 6330–6335.

[14] S. Mneimneh, "Matching from the first iteration: An iterative switching algorithm for an input queued switch," *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 206–217, Feb. 2008.

[15] C. Gauthier. (2011). Overcoming 40G/100G SerDes Design and Implementation Challenges. EE Times. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1279194

[16] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Univ. California Berkeley, Berkeley, CA, USA, 1995.

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

[18] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, "High-speed switch scheduling for local-area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, 1993.

[19] E. Leonardi, M. Mellia, M. A. Marsan, and F. Neri, "Stability of maximal size matching scheduling in input-queued cell switches," in *Proc. IEEE Int. Conf. Commun.*, vol. 3. Jun. 2000, pp. 1758–1763.

[20] K. Xi, Y.-H. Kao, and H. J. Chao, "A petabit bufferless optical switch for data center networks," in *Optical Interconnects for Future Data Center Networks*. New York, NY, USA: Springer, 2013, pp. 135–154.

[21] D. Lin, Y. Jiang, and M. Hamdi, "Selective-request round-robin scheduling for VOQ packet switch architecture," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2011, pp. 1–5.

[22] B. Hu, K. L. Yeung, and Z. Zhang, "An efficient single-iteration single-bit request scheduling algorithm for input-queued switches," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 187–194, 2013.

[23] B. Hu, K. L. Yeung, and Z. Zhang, "Minimizing the communication overhead of iterative scheduling algorithms for input-queued switches," in *Proc. IEEE Global Telecommun. Conf.*, Jun. 2011, pp. 1–5.

[24] B. Hu, K. L. Yeung, Q. Zhou, and C. He, "On iterative scheduling for input-queued switches with a speedup of $2 - 1/N$," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3565–3577, Dec. 2016.

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

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

[27] Y.-S. Lin and C. B. Shung, "Quasi-pushout cell discarding," *IEEE Commun. Lett.*, vol. 1, no. 5, pp. 146–148, Sep. 1997.

[28] M. S. Berger, "Delivering 100% throughput in a buffered crossbar with round robin scheduling," in *Proc. Workshop High Perform. Switching Routing*, Jun. 2006, pp. 403–407.

[29] T. Javidi, R. Magill, and T. Hrabik, "A high-throughput scheduling algorithm for a buffered crossbar switch fabric," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2001, pp. 1586–1591.

BING HU (S'06–M'10–SM'16) received the B.Eng. and M.Phil. degrees in communication engineering from the University of Electronic Science and Technology of China in 2002 and 2005, respectively, and the Ph.D. degree from the Department of Electrical and Electronic Engineering, The University of Hong Kong, in 2009. He joined the College of Information Science and Electronic Engineering, Zhejiang University, in 2010, where he is currently an Associate Professor. His current research interests include next-generation Internet, high-speed packet switch/router design, and data center networks.

FUJIE FAN (S'16) received the B.Eng. degree in electronic information engineering from the Ocean University of China, China, in 2014. He is currently pursuing the Ph.D. degree with the College of Information Science and Electronic Engineering, Zhejiang University. His research interests include routing algorithms for data center networks, traffic engineering, and scheduling algorithms for high-speed switches.

KWAN L. YEUNG (S'93–M'95–SM'99) was born in 1969. He received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong in 1992 and 1995, respectively. He joined the Department of Electrical and Electronic Engineering, The University of Hong Kong, in 2000, where he is currently a Professor. His research interests include next-generation Internet, packet switch/router design, all-optical networks, data center networks, and wireless data networks.

SUGIH JAMIN received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, in 1996. He is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. He co-founded a peer-to-peer live streaming company, Zattoo, Inc., Ann Arbor, MI, USA, in 2005. He received the National Science Foundation CAREER Award in 1998, the Presidential Early Career Award for Scientists and Engineers in 1999, and the Alfred P. Sloan Research Fellowship in 2001.