Fair and Smooth Scheduling for Virtual Output Queuing Switches Achieving 100% Throughput

Min Song[1], Sachin Shetty[1], Wu Li[2]

[1] Electrical and Computer Engineering Department
Old Dominion University
231 Kaufmann Hall
Norfolk, VA 23508
{msong,sshetty}@odu.edu

[2] Multidisciplinary Optimization Branch
NASA Langley Research Center
Mail Stop 159,
Hampton, VA 23681
w.li@larc.nasa.gov

Abstract. Cell scheduling has received extensive attention. Most recent studies, however, focus on achieving a 100% switch throughput under the uniform arrivals. As the demand for quality of service increases, two important goals are to provide predictive cell latency and to reduce the output burstiness. This paper presents a new scheduling algorithm, Worst-case Iterative Longest Port First (W*i*LPF), which improves the performance of the well-known scheduler Iterative Longest Port First (*i*LPF) such that both cell latency and output burstiness are well controlled. Simulation results are provided to verify how W*i*LPF outperforms *i*LPF.

1 Introduction

Due to the high price of output queuing switches and the low throughput of input queuing switches, modern switches mostly deploy virtual output queuing (VOQ) [3][4][6] technique. In VOQ-based technique, each input organizes the buffer to N logical queues that are each associated with an output (an $N \times N$ switch is assumed in this paper); queue $q_{i,j}$ contains cells arriving at input i and destined to output j. Therefore, at each time slot, a scheduler is needed to choose a maximum of N cells from the total N^2 logical queues to forward to N outputs. The objectives of cell schedulers for VOQ-based switches are to provide 1) a 100% switch throughput, 2) fair service to each queue/flow, and 3) smooth scheduling and therefore a reduced output burstiness. Meanwhile, the schedulers must be fast and simple to implement. We assume that data are transferred across the switch fabric in fixed sized cells and that time is slotted. A speedup of one is assumed for the switch fabric.

The rest of the paper is organized as follows. Section 2 introduces the cell scheduling techniques. Section 3 presents the new cell scheduling algorithm. The simulation analysis is given at Section 4. Section 5 provides the conclusions.

2 CELL SCHEDULING TECHNIQUES

For VOQ-based switches, the cell scheduling problem can be viewed as a bipartite graph-matching problem [7][8]. In essence, the scheduling algorithm needs to resolve both *output contention* and *input contention*. Conceptually, two scheduling levels exist: the port level and the queue level. The scheduling process can be performed in either a distributed or a centralized way. In distributed scheduling, the matching decision is made independently at each port. A handshake protocol between inputs and outputs is deployed to perform the following three operations [4]: *REQUEST*—each unmatched input broadcasts its requests to the outputs that it has cells to go; *GRANT*—each unmatched output selects one request independently and issues a grant to the corresponding input; and *ACCEPT*—each input independently chooses one grant to accept.

In centralized scheduling, a unique scheduler collects information from the entire switch and makes the scheduling decision. The information collected includes queue occupancy, the waiting time of head-of-line (HOL) cells, or the cell arrival rates. An example of a centralized scheduling process is the Iterative Longest Port First (*i*LPF) with running time complexity $O(N^2)$ [3]. It includes two steps: *SORT*—all inputs and outputs are sorted based on their port occupancies and their requests are reordered according to their port occupancies, and *MATCH*—for each output and input from largest to smallest, if a request is made and both input and output are unmatched, then the scheduler will match them. The port occupancy is calculated as follows:

$$R_i(n) = \sum_{j=1}^{N} l_{i,j}(n),\ C_j(n) = \sum_{i=1}^{N} l_{i,j}(n) \tag{1}$$

where $l_{i,j}(n)$ denotes the occupancy of queue $q_{i,j}$ at cell time n. The notations $R_i(n)$ and $C_j(n)$ are the input port occupancy and output port occupancy at cell time n, respectively. The *i*LPF algorithm is a well-known scheduling algorithm that achieves 100% throughput and is stable for all admissible independent arrival processes. It is also simple to implement in hardware.

Unfortunately, most scheduling algorithms are designed solely to achieve a 100% switch throughput. They only bound the expected values, such as queue length or waiting time, by using Lyapunov stability analysis. However, bounding the expected values is not sufficient to provide predictable latency to individual connection. For example, the *i*LPF algorithm performs well for uniform traffic, but not as well for non-uniform traffic and for switches working in an asymmetric mode. It causes the latency for some queues to be unpredictable.

Another issue addressed in this paper is the output burstiness. In packet switched networks, traffic patterns become increasingly irregular as packets are multiplexed and buffered at the intermediate nodes [1][2]. Schedulers should try to smooth the traffic as much as possible. Smooth scheduling helps networks accommodate more traffic, reduce the traffic burstiness, and provide a tight end-to-end delay bound in high-speed networks. Results from our preliminary study have been presented in [9].

3 THE NEW SCHEDULING ALGORITHM

We consider a switch that works under an asymmetric mode, i.e., the traffic distribution is non-uniform. As an example, Figure 1 shows a 2×2 VOQ-based switch, in which the arrival rates for queues are distributed as $\lambda_{1,1} = 0.89$, $\lambda_{1,2} = 0.1$, $\lambda_{2,1} = 0.1$, and $\lambda_{2,2} = 0.1$. The scenario in Figure 1 happens when the output link 1 is close to *hot-spot* servers.

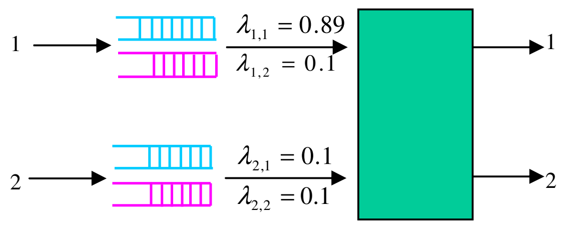


Fig. 1. A 2×2 VOQ-based switch works in asymmetric traffic mode

After the warm-up running τ slots, the following two inequities follow

$$l_{1,1}(\tau) > l_{1,2}(\tau) \text{ and } l_{2,1}(\tau) > l_{2,2}(\tau)$$

Then according to Equation (1), the following two port inequities follow

$$C_1(\tau) > C_2(\tau) \text{ and } R_1(\tau) > R_2(\tau)$$

Thus, according to *i*LPF, queues $q_{1,1}$ and $q_{2,2}$ continue receiving service until one of the two port inequities does not follow. Until then, queues $q_{1,2}$ and $q_{2,1}$ cannot receive any service. Thus, cells in queues $q_{1,2}$ and $q_{2,1}$, henceforth called *tagged queues*, experience significant delay. The reasons for this phenomenon are *1*) the *link-dependency*, i.e., whether a queue gets service or not depends on not only the traffic of its own link, but also the traffic from other links; and *2*) the *longest port preference*, i.e., *i*LPF gives preference to the queue with the longest port occupancy. There is no mechanism to guarantee a minimum amount of service to an individual queue. Consequently, tagged queues experience unexpected delays, and queues tend to receive batch service that increases the output burstiness. To alleviate these problems, we have designed a Worst-case Controller (Fig. 2) that monitors the queue occupancy and feeds back to the centralized scheduler to dynamically adapt the scheduling decision.

In particular, if WC finds that a nonempty queue $q_{i,j}$ has not received service, hence called a *worst-case queue*, for more than $w_{i,j}$ cell times, and both input i and output j are not yet matched, then the WC will match them. We call this process *worst-case matching*. If a conflict occurs among the worst-case queues, the one with the longest port occupancy gets service. Thus, a worst-case queue may need to wait, at maximum, for $2(N - 2)$ slots to get service. This deterministic bound in head-of-line-cell waiting time effectively guarantees that each queue (and thus its constituent flows) receives its reserved service share and that the service time to each queue spreads as evenly as possible.

```
$w_{i,j} = \left\lceil 1/\lambda_{i,j} \times \max\left(\sum_i \lambda_{i,j}, \sum_j \lambda_{i,j}\right) \right\rceil$

$\Delta l_{i,j}(n) = l_{i,j}(n) - l_{i,j}(n-1)$

*if* $[(\Delta l_{i,j}(n) = 1)$ *or* $((\Delta l_{i,j}(n) = 0) \wedge (l_{i,j}(n-1) = l_{i,j}(n-1)^{+}))]$
    *then* $\Lambda l_{i,j} = \Lambda l_{i,j} + 1$
*else* $\Lambda l_{i,j} = 0$
*if* $(\Lambda l_{i,j} \geq w_{i,j})$
    *then* mark queue $q_{i,j}$ as a worst-case queue, $\Lambda l_{i,j} = 0$
```

Fig. 2. Worst-case Controller (WC) [1]

We call this property *fair and smooth scheduling*. The combination of WC and *i*LPF is called Worst-case Iterative Longest Port First (W*i*LPF) (Fig. 3), where the WC is embedded at the end of the *SORT* step of *i*LPF algorithm. This process ensures that the worst-case matching has a higher priority than the normal matching. Similar to *i*LPF, the two steps in W*i*LPF can be pipelined to keep the running time complexity of $O(N^2)$. It should be noticed that the WC effectively functions as a traffic shaper or rate controller [5].

```
*Step I: Sorting & Worst-case Matching*
  1.  Sort inputs and outputs based on their port occupancies
  2.  Reorder requests according to their input and output port occupancies

  3.  Run WC for each output and input from largest to smallest
          *if* (queue $q_{i,j}$ is a worst-case queue) and (both input and output unmatched)
          *then* match them

*Step II: iLPF Matching*
  1.  *for* each output from largest to smallest
  2.      *for* each input from largest to smallest
  3.          *if* (there is a request) and (both input and output unmatched)
                  *then* match them
```

Fig. 3. The W*i*LPF algorithm

[1] $l_{i,j}(n)$ and $l_{i,j}(n^{+})$ represent the queue $q_{i,j}$ occupancy at the beginning and the end of cell slot n, respectively.

4 SIMULATION ANALYSIS

To validate the W*i*LPF algorithm, we conducted the simulation using a discrete-event simulator written for the purpose[2]. Simulations were run by using a 3×3 VOQ-based switch. The arrival rates for the three queues at links 1 and 2 are fixed as 0.79, 0.1, and 0.1, respectively. For link 3, the arrival rates for the first two queues are both fixed as 0.1; the arrival rate for the third queue is a variable from 0.1 to 0.7. All simulation runs have been fixed at one million cell times. Both Bernoulli traffic and Bursty traffic ($E[B]$ = 16) are used. Fig. 4 shows the Markov transition diagram for the cell arrival process at link 2, where states 1, 2, and 3 represent arriving cells for $q_{2,1}$, $q_{2,2}$, and $q_{2,3}$, respectively; $p_1 = \frac{\lambda_{2,1}}{\lambda_{2,1}+\lambda_{2,2}+\lambda_{2,3}}$, $p_2 = \frac{\lambda_{2,2}}{\lambda_{2,1}+\lambda_{2,2}+\lambda_{2,3}}$, and

$$p_3 = \frac{\lambda_{2,3}}{\lambda_{2,1}+\lambda_{2,2}+\lambda_{2,3}}$$

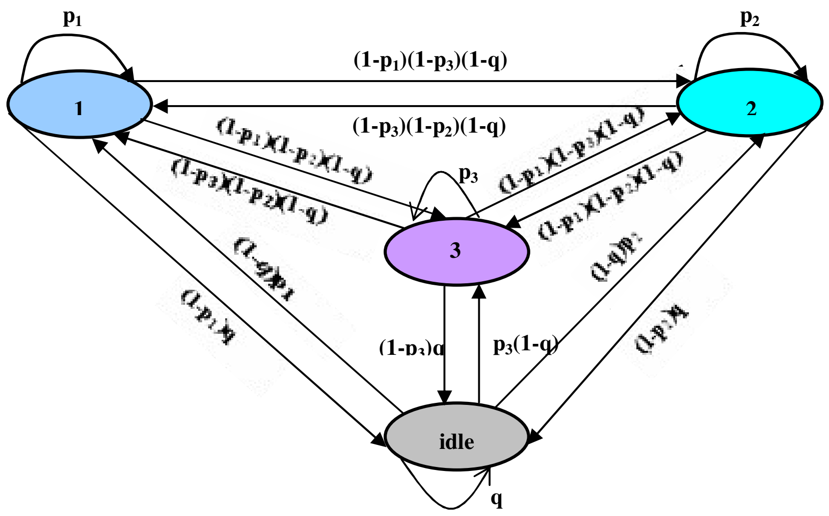


Fig. 4. Markov chain transition diagram for the cell arrival process at link 2

Tables 1 and 2 provide the average cell latency for each queue as a function of the utilization of link 3 under *i*LPF and W*i*LPF algorithms for Bernoulli traffic. Although the latencies of queues $q_{1,1}$, $q_{2,2}$, $q_{2,3}$, and $q_{3,3}$ in W*i*LPF are increased at a maximum 15 cell times, the latencies for all other queues in W*i*LPF are decreased with a maximum 42 cell times. All queue latencies are upper bounded by the expected waiting time of an $M/D/1$ queue, in which $d = \lambda / 2\mu(\mu - \lambda)$.

[2] The simulator was designed based on the simulator used in paper [3].

Table 1. Average cell latency (in cell times) for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.3	0.4	0.5	0.6
$q_{1,1}$	8.4/20.2	7.6/19	8.6/20.8	8.2/20.5
$q_{1,2}$	20.8/21.3	21.4/20.6	24.6/22	27.2/22.8
$q_{1,3}$	40.9/29.1	39.5/28.5	41/29.9	40.3/29.2
$q_{2,1}$	54/51.9	51.6/46.9	61.2/52.2	63.5/50.6
$q_{2,2}$	6.6/3.5	6/3.4	6.1/3.6	5.6/3.5
$q_{2,3}$	5.1/2.8	5.1/2.9	5.4/2.9	5.6/3.1
$q_{3,1}$	103/60.7	93.5/53.8	102.5/61	96.8/57.6
$q_{3,2}$	5.3/5.2	5.5/5.2	5.3/5.1	5.8/5
$q_{3,3}$	0.7/0.3	0.8/0.4	0.9/0.5	1.2/0.6

Table 2. Average cell latency (in cell times) for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.7	0.8	0.9
$q_{1,1}$	8.6/20.9	8.75/20.8	10.7/25.3
$q_{1,2}$	30/23.1	35.8/23.6	48.6/27.2
$q_{1,3}$	40/28.3	41.2/27.7	42.9/27.9
$q_{2,1}$	69.6/50.6	77.4/52.8	101/58.9
$q_{2,2}$	5.4/3.5	5.2/3.5	5.4/3.6
$q_{2,3}$	5.9/3.3	6.7/3.8	9.9/6.1
$q_{3,1}$	95.6/57	93.1/55.9	98.9/60.2
$q_{3,2}$	6.2/5.1	7.1/5.2	8.9/5.7
$q_{3,3}$	1.6/0.8	2.6/1.3	5.5/3.4

Tables 3 and 4 provide the average cell latencies for Bursty traffic. The average cell latencies for queues $q_{1,2}$, $q_{1,3}$, $q_{2,1}$, and $q_{3,1}$ in W*i*LPF are reduced for maximum six cell times and for queues $q_{1,1}$, $q_{2,2}$, $q_{2,3}$, and $q_{3,3}$ the latencies are increased for maximum four cell times.

Table 3. Average cell latency (in cell times) for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.3	0.4	0.5	0.6
$q_{1,1}$	5.1/7.3	5.3/7.8	5.7/8.5	6.3/9.7
$q_{1,2}$	12.6/10.4	13.9/11.4	15/12	17.1/13.9
$q_{1,3}$	8.8/8.6	9.6/9.6	10.5/10.8	12.5/12.7
$q_{2,1}$	26.6/20.2	23.9/18.8	23.9/17.9	24/18.2
$q_{2,2}$	5.5/6.8	5.5/7	5.5/7.4	5.8/7.9
$q_{2,3}$	4.8/5.1	6.5/6.5	8.4/7.8	11.5/9.3
$q_{3,1}$	23.6/20.7	22.7/18.9	22.3/18.3	23.2/17.8
$q_{3,2}$	7.6/7.6	8.9/8.1	10.4/8.5	12.3/10.4
$q_{3,3}$	3.9/5.8	3.9/5.1	4.3/5	4.7/6.5

Table 4. Average cell latency (in cell times) for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.7	0.8	0.9
$q_{1,1}$	7.1/11.1	8.1/13	10.3/15
$q_{1,2}$	20/15.2	22.5/17	28/19
$q_{1,3}$	14.7/14.3	17.7/16	24.8/21
$q_{2,1}$	24/18	24.2/19.3	25/20
$q_{2,2}$	6.4/9	7.2/10.4	9.4/16
$q_{2,3}$	15.2/12.1	18.9/14.8	27.5/20.2
$q_{3,1}$	24.9/18.2	26/18	29.2/21.4
$q_{3,2}$	14.7/12.4	17.6/15	22.8/20.4
$q_{3,3}$	5.5/7.7	6.5/9.4	9.2/15.9

The most significant performance improvement in W*i*LPF can be seen in the HOL cells holding time as shown in Tables 5 and 6.

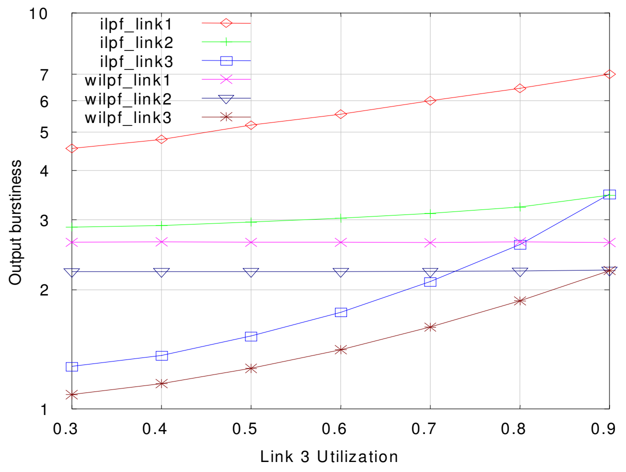
Table 5. Maximum HOL cells holding time for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.3	0.4	0.5	0.6
$q_{1,1}$	39/3	31/3	46/3	34/3
$q_{1,2}$	159/10	145/10	179/10	165/10
$q_{1,3}$	199/10	180/10	187/10	230/10
$q_{2,1}$	287/11	275/11	301/11	307/11
$q_{2,2}$	26/4	27/4	35/ 4	25/4
$q_{2,3}$	59/11	67/11	85/11	81/11
$q_{3,1}$	154/10	168/10	169/10	160/10
$q_{3,2}$	106/11	93/11	113/11	92/11
$q_{3,3}$	15/4	16/4	20/4	22/ 4

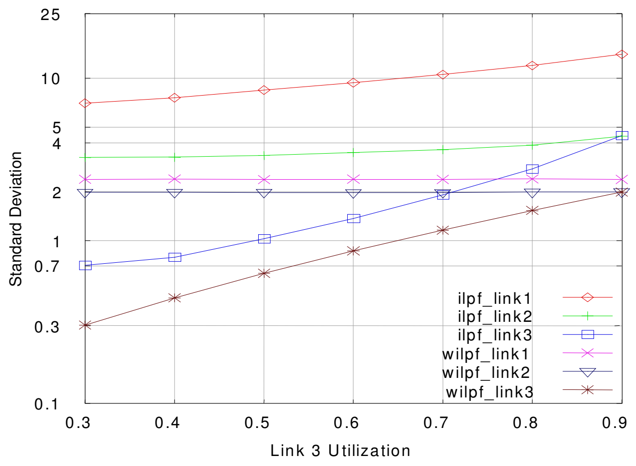
Table 6. Maximum HOL cells holding time for each queue as a function of the utilization of link 3 (data are in the format of *i*LPF/W*i*LPF)

Queues	0.7	0.8	0.9
$q_{1,1}$	44/3	38/3	40/3
$q_{1,2}$	157/10	222/10	217/10
$q_{1,3}$	176/10	196/10	188/10
$q_{2,1}$	293/11	280/11	314/11
$q_{2,2}$	35/4	38/4	32/4
$q_{2,3}$	55/11	72/11	95/11
$q_{3,1}$	213/10	221/10	304/10
$q_{3,2}$	83/11	83/11	109/11
$q_{3,3}$	21/4	23/4	31/4

Because the W*i*LPF algorithm spreads the service time to queues evenly, both the output burstiness and its standard deviation (Fig. 5) are exceedingly reduced.



(a) The output burstiness



b) The standard deviation of output burstiness

Fig. 5. The output burstiness (a) and standard deviation (b) as the function of link 3 utilization

5. CONCLUSIONS

To achieve deterministic cell latency, smooth output traffic, and a high switch throughput, we have designed a new cell scheduling algorithm, W*i*LPF, for

VOQ-based switches. W*i*LPF has two components: a worst-case controller and a centralized scheduler. The worst-case controller monitors the queue behavior and feeds back to the centralized scheduler. The worst-case controller, which is unique to W*i*LPF, can be easily embedded into the centralized scheduler without increasing the overall running time complexity of $O(N^2)$. Analysis and simulation results suggest that W*i*LPF reduces the overall cell latency and significantly smoothes the output traffic, and keeps the same switch throughput and same running complexity as of *i*LPF. Similar to *i*LPF, the two steps in W*i*LPF can be pipelined to reduce the running time. This means that the matching algorithm operates on weights that are one slot out of date. However, it is still stable for all admissible independent arrival processes.

References

[1] A. Raha, S. Kamat, X.H. Jia, and W. Zhao, “Using Traffic Regulation to Meet End-to-End Deadlines in ATM Networks,” *IEEE Trans. on Computers*, Vol. 48, Sept. 1999, pp. 917–935.

[2] L. Georgiadis, R. Guérin, V. Peris, and K.N. Sivarajan, “Efficient Network QoS Provisioning Based on per Node Traffic Shaping,” *IEEE/ACM Trans. on Networking*, Vol. 4, Aug. 1996, pp. 482–501.

[3] A. Mekkittikul and N. McKeown, “A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches,” *Proc. of the IEEE INFOCOM*, Vol. 2, April 1998, pp. 792–799.

[4] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, “High Speed Switch Scheduling for Local Area Networks,” *ACM Trans. on Computer Systems*, Vol. 11, Nov. 1993, pp. 319–352.

[5] D. Stiliadis, and A. Varma, “Rate-Proportional Servers: A Design Methodology for Fair Queuing Algorithms,” *IEEE Trans. on Networking*, Vol. 6, April 1998, pp. 164–174.

[6] N. McKeown, B. Prabhakar, and M. Zhu, “Matching Output Queuing with Combined Input and Output Queuing,” *IEEE Journal on Selected Area in Comm.*, Vol. 17, June 1999, pp.1030–1038.

[7] A. C. Kam, and K.Y. Siu, “Linear-Complexity Algorithms for QoS Support in Input-Queued Switches with No Speedup,” *IEEE Journal on Selected Area in Comm.*, Vol. 17, June 1999, pp. 1040–1056.

[8] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, “Achieving 100% Throughput in an Input-Queued Switch”, *IEEE Trans. on Commu.*, Vol. 47, No. 8, August 1999, pp. 1260-1267.

[9] M. Song, and M. Alam, “Two Scheduling Algorithms for Input-queued Switches Guaranteeing Voice QoS,” *Proc. of IEEE Global Telecommunications Conference*’01, Texas, November 25-29, 2001, Vol. 1, pp. 92–96.