Credit-based fair scheduling for Input-Queued Switches

Amir Gourgy, Honglin Wu, and Ted H. Szymanski
Department of Electrical and Computer Engineering
McMaster University, Hamilton Ontario L8S 4K1, Canada
amir@grads.ece.mcmaster.ca, wuh4@mcmaster.ca, and teds@mail.ece.mcmaster.ca

***Abstract*— We present a novel scheduling algorithm for internet routers with input-queued switches based on credit-based fair queueing. We present a flow-based iterative credit-based fair scheduler (iCBFS), for crossbar switches, that provides fair bandwidth distribution among flows at a fine granularity and achieves asymptotically 100% throughput, under uniform traffic. To reduce the implementation complexity of iCBFS, we present a port-based version of iCBFS that is tailored towards high-speed hardware implementation.**

Keywords-Input-Queued Switches, Scheduling, quality-of–service.

I. INTRODUCTION

There is a tremendous demand for Internet core nodes to provide quality-of-service (QoS) guarantees for multimedia services like VOIP, video-on-demand, and mission-critical applications, and to provide high switching capacity that makes uses of the virtually unlimited bandwidth of optical fibers. The Internet's success depends on the deployment of high-speed switches and routers that meet these two demands.

On the one hand, the demand of QoS guarantees can be met using output-queued (OQ) switches, which can provide optimal throughput. In addition, much research effort has been devoted to packet scheduling at output ports to support fair bandwidth sharing that provides delay bounds for regulated traffic (e.g., Weighted Fair Queueing (WFQ) family [10]). However, OQ for a $N \times N$ switch requires the switching fabric and memory to run up to N times faster than the line rate (i.e., speedup of N); unfortunately, for large N or for high-speed data lines, memories with sufficient bandwidth are not available.

On the other hand, the fabric and the memory of an input-queued (IQ) switch need only run as fast as the line rate. This makes input queueing very appealing for switches with fast line rates or with a large number of ports. Consequently, most high-performance switches and routers (e.g., [7] and [11]) use IQ-crossbar switches with unity speedup. These high-speed switches are built around a crossbar switch that is configured using a centralized scheduler designed to provide high throughput and they use a fixed-length cell as a transfer unit. Fixed-length switching technology is widely accepted for achieving high switching efficiency such that variable-length packets are segmented into fixed-length cells at the inputs and are reassembled at outputs. We assume fixed-length cell scheduling for the remainder of this paper[1].

Although several practical scheduling algorithms [13], [7], and [12] (described in next section) have been proposed for IQ switches to provide QoS guarantees, these algorithms provide bandwidth guarantees over coarse granularity (i.e., a frame) and exhibit unfairness over short time scales. Specifically, these schemes are fair only over timescales longer than a *frame size*, where the frame size is one round-robin of service over all backlogged queues in the switch in proportion to their reservations. Over timescales less than a frame, these schedulers do not serve flows in proportion to their reservations and flows can be served in any arbitrary order. Although the aggregate bandwidth received by a flow over the entire frame is in proportion to its reservation, within a frame time some flows may not get any service until the very end of the frame and bandwidth distribution over the frame time is nonuniform. Furthermore, as the switch size increases, the number of queues in the switch increases and frame size becomes larger; thereby, this unfairness leads to increased jitter, which is undesirable for multimedia services like VOIP. It is this problem that our proposed scheduler solves. We emphasize that this problem can not be solved by using a smaller frame size because the frame size is limited by the resolution of the minimum allocatable fraction of bandwidth per flow; for example, consider a future core router with link speeds of 100 Gbps. For a flow to reserve only 10 Mbps, or 0.01 percent of the link capacity, requires the frame size to be at least 10000 time slots. This problem will grow in significance as link speeds increase.

In this paper, we propose a scheduling algorithm for *IQ* switches based on credit-based-fair-queueing [2], called iterative credit based fair scheduling (iCBFS). Our simulation results show that iCBFS provides fair bandwidth distribution among flows bandwidth at a fine granularity, and solves the unfairness for timescales smaller than a frame size; thereby our algorithm provides better fairness than all existing schemes, with comparable hardware complexity. In addition, our schemes achieves asymptotically 100% throughput, under uniform traffic.

This paper is organized as follows. Section II provides a review of related work on scheduling for IQ

[1] The words *packet* and *cell* are used interchangeably for the remainder of this paper.

switches. Section III discusses fairness in IQ schedulers, presents our proposed flow-based scheduler (iCBFS), and compares its performance to other scheduling schemes. In section IV, we propose a port-based version of (iCBFS) that is tailored towards efficient high-speed hardware implementation. The hardware complexity of the proposed scheduler is described in section V. Finally, section VI concludes this paper and summarizes our contributions.

II. BACKGROUND AND RELATED WORK

IQ switches can suffer from head-of-line (HOL) blocking that limits the throughput to just 58.6% if each input maintains a single FIFO queue for all packets [6]. In virtual output queueing (VOQ) each input maintains a separate FIFO queue for each output to eliminate HOL. When VOQ is used in a crossbar switch, the scheduling algorithm configures the fabric during each packet time and decides which inputs will be connected to which outputs. In an $N \times N$ switch this requires the scheduler to examine the contents of N^2 virtual output queues and determine a conflict free match M between inputs and outputs. This is equivalent to finding a bipartite graph matching on a graph with N vertices [7]. Although this problem can be optimally solved using a maximum weight bipartite graph matching algorithm [7], [8], it requires a running time complexity of $O(N^3 \log N)$ on a sequential model, which makes the optimum algorithm prohibitively expensive to implement in hardware. Instead, most practical algorithms are based on simple heuristics that aim at maximizing the number of connections between inputs and outputs during each matching phase.

Anderson, et al. [1] designed an alternative to the optimal algorithm, called parallel iterative matching (PIM). PIM uses random selection to solve the contention in inputs and outputs. Input packets are first queued in VOQs. PIM uses an iterative three stages matching policy: request, grant, and accept stages. For simplicity, we call other schemes based on these stages as πRGA. Initially, all inputs and outputs are unmatched, and only those inputs and outputs that are not matched at the end of one iteration are eligible to participate in the next matching iteration. Specifically, PIM iterates the following three steps until either a *maximal matching* is found or a fixed number of iterations are performed. A maximal matching is one that adds connections incrementally, without removing connections made earlier in the matching process.

1. **Request stage:** Each unmatched input sends a request to every output for which it has a queued packet.
2. **Grant stage:** If an unmatched output receives multiple requests, it grants one by *randomly* selecting one of the requests. Each request has equal probability to be granted.
3. **Accept stage:** If an input receives multiple grants, it accepts one by randomly selecting an output among them.

Although finding a maximal matching using the probabilistic matching algorithm may, in the worst case, take N iterations, it was shown that under *uniform traffic*, the algorithm will converge to a maximal match in $O(\log N)$ iterations [1].

Mckeown proposed iSLIP [7] as an improvement over PIM. Instead of using random selection at both inputs and outputs, iSLIP uses rotating round-robin priority arbiters at both inputs and outputs. Under uniform Bernoulli i.d.d. traffic iSLIP arbiters adapt to a time-division multiplexing scheme, providing a perfect match and 100% throughput [7]. Mckeown [7] proposed weighted iSLIP (WiSLIP) as a variation of iSLIP that can allocate bandwidth nonuniformly to different inputs. The bandwidth from input i to output j is given by the ratio $f_{ij} = n_{ij} / d_{ij}$, where n_{ij} is the reservation for the $input_i - output_j$ pair, and d_{ij} is the aggregate reservation for output j. Instead of each arbiter maintaining an ordered circular list $S = \{1,...,N\}$ as in iSLIP, the list is expanded in WiSLIP at output j to be the ordered circular list $S_j = \{1,...,W_j\}$, where W_j = lowest common multiple of all the aggregate reservations for output j at all the input ports and input i appears $(n_{ij} / d_{ij}) \times W_j$ times; that is, the size of the circular changes based on the reservation values.

Stiliadis [12] proposed weighted PIM (WPIM) that allocates output bandwidth among inputs based on reservations made during an admission control phase. In WPIM, the time axis is divided into frames with a fixed number of slots per frame (e.g., a frame is typically 1000 slots [12]). The reservation unit is slot/frame. Consequently, WPIM provides bandwidth guarantee at a coarse granularity of a frame size.

Ni and Bhuyan [3] proposed a fair scheduling algorithm for input-queued switches called iFS, which is based on virtual time. In iFS, each output link maintains a fair queueing engine, which assigns a virtual time to every incoming packet based on bandwidth reservation of the packet's flow. The incoming packet is then queued in an input buffer first-in-first-out on a per flow basis. The algorithm then executes a maximal matching scheme based on virtual time, where only the grant and accept stages are executed.

On the one hand, by using virtual-time stamps for every incoming packet, iFS [3] can honour bandwidth reservations at a very fine granularity better than most existing schemes; on the other hand, the cost of this algorithm is the increased complexity in implementing N virtual-time based fair queueing engines. A major problem with virtual-time-based approaches is the time stamp overflow. Because time stamp is an increasing function of the time that depends on a common virtual clock, which in turns reflects the value of the time tag of previously served packets, the virtual clock cannot be reinitialized to zero until the system is completely empty and all sessions are idle, which although statistically finite can be extremely long, given that most real-communication traffic exhibits self-similar patterns. This may easily cause an overflow in time tag unless special hardware algorithms are used [3]. Floating-point units are usually used in computing the virtual-time stamp. In addition, virtual-time-based approaches require that packets be sorted according to their time tags by the fair queueing engine. In iFS, every incoming

packet needs to be assigned a virtual time-stamp and inserted into a sorted list. Therefore, for practical implementation of iFS, a very high-speed fair queueing hardware engine needs to be designed to compute virtual time-stamps, perform sorting, and be able to process up to N packets during each time slot. These requirements are expensive to implement in hardware.

To overcome the complexity of using a virtual-time-based fair-queuing engine at each output, and assigning a virtual time-stamp to each incoming packet, Zhang and Bhuyan [13] proposed iDRR, a πRGA scheduling scheme based on deficit round-robin. In iDRR, each input and output maintain a circular list such that inputs and outputs are matched in round-robin based on a quota value assigned by deficit-round-robin engines, in proportion to their reservations. Each matched input-output pair may transfer packets until it uses its available quota or there are no more packets to transfer.

III. A FLOW-BASED FAIR SCHEDULING ALGORITHM

First, a definition of fairness in input-queued switch scheduling is presented. Second, we describe the architecture of our proposed flow-based iterative credit-based fair scheduler (iCBFS). Third, we present iCBFS algorithm in detail. Fourth, we evaluate the performance of iCBFS using various traffic models, and compare its fairness to WiSLIP, WPIM, and iDRR.

A. Definition of fair scheduling

We assume a work-conserving input-queued switch, and use a definition of fairness similar to [9] and [13]. Let $flow_k(i,j)$ denote the kth flow from input i to output j with bandwidth reservation S_k, and $W_k(t_1,t_2]$ be the amount of $flow_k(i,j)$ traffic served in the interval $(t_1,t_2]$. Two flows $flow_M(i_1,j_1)$ and $flow_N(i_2,j_2)$ are in contention if $i_1 = i_2$ or $j_1 = j_2$; that is, in an input-queued switch, there are essentially two shared resources: the crossbar, where flows at each input contend (input contention); and the bandwidth of each outgoing link, where flows destined to the same output link contend (output contention). For any two backlogged flows $flow_M(i_1,j_1)$ and $flow_N(i_2,j_2)$ that are in contention, a scheduling scheme is ideally fair in $(t_1,t_2]$ if

$$\frac{W_M(t_1,t_2]}{S_M} = \frac{W_N(t_1,t_2]}{S_N}$$

That is, contending flows are served in proportion to their reservations. This definition of fairness, of course, holds only in an idealized fluid flow network. When the network is more realistic and serves the traffic flows by a nonnegligible quantum of variable size (packet by packet), the definition of fairness can be written as

$$\left| \frac{W_M(t_1,t_2]}{S_M} - \frac{W_N(t_1,t_2]}{S_N} \right| \le B$$

where B is a bound that gives a measure of fairness, also called fairness index. The smaller the fairness index, the fairer is the scheduling algorithm.

B. Architecture of iCBFS Switch

The basic architecture of iCBFS is shown in Figure 1; for each output link we maintain a credit-based fair queueing (CBFQ) arbiter and a separate queue is used for each flow at input ports. The scheduling algorithm is based on a πRGA policy.

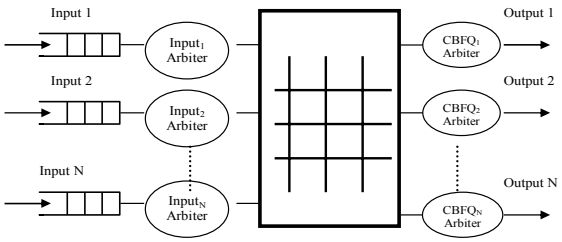


Figure 1. Architecture of a flow-based credit-based fair queueing switch.

The basic idea of iCBFS is to assign each flow a counter that gets incremented in proportion to the flow's reservation such that when the counter reaches a certain threshold value, its corresponding flow is flagged as a *candidate*, and is allowed to transmit a packet across the switch; subsequently, the counter is decremented after transmission. These counters are maintained by the CBFQ arbiters and are used to fairly resolve output contention as described in the next section.

In addition to the counters used by CBFQ arbiters, each input arbiter tracks the aggregate reservation from its port to all outputs, and uses a set of counters to track the aggregate number of packets transmitted to each output. These input arbiters' counters are used to fairly resolve input contention as described in the next section. All counters can be updated independently in parallel, which suits efficient hardware implementation.

Let the average packet arrival rate at input i for output j be λ_{ij}. The incoming traffic is called *admissible* if $\sum_{i=1}^{N} \lambda_{ij} < 1$, and $\sum_{j=1}^{N} \lambda_{ij} < 1$. We assume that flows' reservations are admissible.

C. Description of iCBFS Algorithm

In an $N \times N$ switch, for each $flow_k(i,j)$ going from input i to output j, the input arbiter i uses a separate queue q_{jk}, and the CBFQ_j arbiter maintains a counter K_{jk} and a bandwidth share S_{jk}. Let $Q_j = \{q_{j1}, q_{j2}, ..., q_{jJ}\}$ be the set of queues for flows $1,...,J$ that are destined to output j, with bandwidth shares $S_{j1} \ge S_{j2} ... \ge S_{jJ}$. Initially, all counters are set to zero. Each CBFQ_j engine updates the counters as follows: $K_{jk} = K_{jk} + S_{jk} / S_{j1}$; that is, the backlogged queue with the largest share, at each CBFQ engine, is chosen as the reference queue to calculate a pro-rated share of bandwidth each backlogged queue should receive. K_{jk} is the accumulated credit for $flow_k$ destined to output j. Each $flow_k(i,j)$ with $K_{jk} \ge 1$ and $|q_{jk}| > 0$ is marked as candidate and can be used during the iterative matching phase as described next.

In addition to the counters used by CBFQ arbiters, each input arbiter i uses a set of N counters

$C_i = \{C_{i1}, C_{i2}, ..., C_{iN}\}$ such that C_{ij} indicates the current available quota of the $input_i - output_j$ pair. The quota is the number of reserved slots per frame for each $input_i - output_j$ pair. Let R_{ij} represent aggregate reserved bandwidth from $input_i$ to $output_j$. Each input arbiter i then assigns a quota to the counter values $\{C_{i1}, C_{i2}, ..., C_{iN}\}$ such that C_{ij} indicates the current available quota of the $input_i - output_j$ pair. These quotas can be either statically or dynamically reconfigured. In the static approach, a fixed minimum quota value ($q_{\min}$) is assigned to the minimum possible aggregate reservation $R_{\min}$. Subsequently, each aggregate reservation C_{ij} is assigned a quota $q_{\min} R_{ij} / R_{\min}$. In the dynamic approach, the value of $q_{\min}$ can be dynamically calculated based on the current flow with minimum reservation and the quotas of all other flows are calculated accordingly.

Initially, all inputs and outputs are unmatched. Then in each iteration:

1. **Request:** Each unmatched input sends a request to every output for which it has a queued packet.
2. **Grant:** If an unmatched output receives any requests choose any candidate flow that belongs to an unmatched input and send a grant to this flow at its corresponding input. Note that counters are updated if there are no candidate flows for any of the requests.
3. **Accept:** If an unmatched input receives any grants choose the flow with the largest quota for its C_{ij} counter. Note that selecting the flow with the largest quota resolves input contention in fair and simple manner.

In each time slot, for every selected flow, the switch transfers a packet of its head-of-line (HOL) queue. The input arbiter decrements the quota by 1 and the output arbiter decrements the flow's counter value by 1. The previous algorithm executes until either no more matches can be made or for a fixed number of iterations.

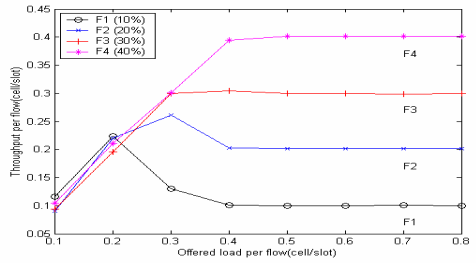
To circumvent flows from overusing or underusing their reservations, we require all quotas and counters be reinitialized after some period of time. For simplicity, we assumed a fixed frame size (e.g., 1000 slots) after which all the counters are initialized.

D. Simulation Results

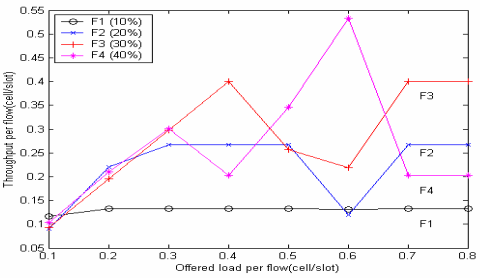
First, the switch is set such that different flows have different reservations and the throughput per flow is measured to evaluate the fairness of the scheduler. Second, the switch setting is such that all flows have equal reservations and the performance is measured for a 16×16 switch under uniform Bernoulli i.d.d. traffic. The number of iterations was fixed to 4. The performance of iCBFS is compared to WiSLIP, iDRR, and WPIM.

D1. QoS Traffic Model

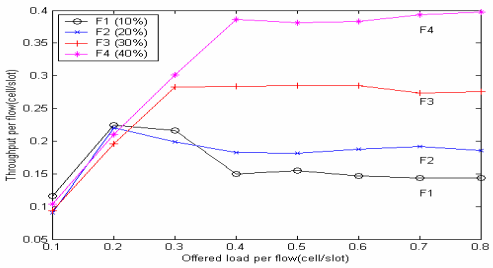
To illustrate the fairness of iCBFS in bandwidth allocation, a 4×4 switch was simulated such that each input has four flows, each going to a different output with a different bandwidth reservation. Let $f_k(i, j)$ represent flow k from input port i to output port j. In the simulated switch, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, $f_4(3,0)$ have reserved 10, 20, 30, and 40 percent of the bandwidth, respectively, but they always maintain the same actual arrival rate. Other flows have a load of 5 percent each. This traffic model has been used in [9] and [12]. We used equivalent switch settings for iCBFS, iDRR, and WPIM with equivalent frame size of 1000 slots. Figure 2 shows the throughput per flow using iCBFS, WiSLIP, iDRR, and WPIM after 750 time slots. The value of 750 represents 75 percent of the frame size and was chosen to illustrate the short-term unfairness problem present in other schemes and the superiority of iCBFS in solving this problem.



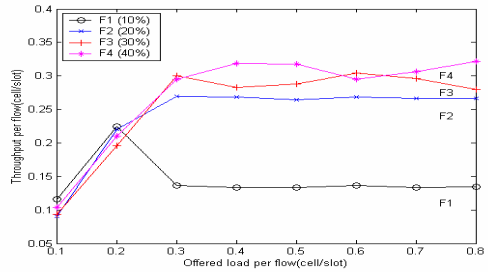
(a) Throughput per flow using iCBFS



(b) Throughput per flow using iDRR



(c) Throughput per flow using WiSLIP



(d) Throughput per flow using WPIM.

Figure 2. Plot of throughput per flow for iCBFS, iDRR, WiSLIP, and WPIM at 75 percent of the frame size

iCBFS vs. iDRR

Both iCBFS and iDRR were simulated with $q_{\min} = 50$ slots and $r_{\min} = 5\%$ with static counter initialization after 1000 slots. Although iDRR [13] avoids the complexity of the virtual-time approach used in iFS, it does that at the expense of other performance metrics such as delay and fairness. iDRR possesses all the deficiencies inherent in deficit-round-robin service, namely that is fair only over time scales longer than frame, and it has unbounded delay (the delay depends on local switch settings that can be arbitrarily large; see [5] p. 3). As shown in Figure 2(b), at the rightmost point of the graph, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, and $f_4(3,0)$ receive 13, 26, 40, and 20 percent of the bandwidth, respectively. Specifically, $f_4(3,0)$ receives only half of its reserved bandwidth share leading to a large delay and jitter. In contrast, iCBFS is able to precisely allocate the bandwidth among the flows in proportion to their reservation.

iCBFS vs. WiSLIP

As shown in Figure 2(c) at the rightmost point of the graph, WiSLIP does not precisely allocate bandwidth among flows in proportion to their reservations; $f_1(0,0)$ receives 15 percent of the bandwidth instead of its reserved 10 percent. Consequently, both $f_2(1,0)$, $f_3(2,0)$ receive only 18% and 27% instead of 20% and 30%, respectively. In contrast, iCBFS is able to precisely allocate the bandwidth among the flows in proportion to their reservation. We identify the unfairness in iSLIP and its variant WiSLIP as caused by the simple operation of the rotating round-robin priority arbiters—the output arbiters do not track precisely how much bandwidth each input port uses. Specifically, iSLIP and all its variants [7] use simple rotating round-robin priority arbiters at each output arbiter with a pointer g_i to the current highest priority input of the round-robin schedule. This pointer g_i is only incremented (modulo N) if, and only if, the grant signal is accepted in the first iteration of the algorithm. For all subsequent iterations, the pointer is not updated even if a granted input is accepted. Although this scheme elegantly eliminates starvation in both iSLIP and its variants, it leads to impreciseness in tracking the bandwidth allocated to each input port (see [7] for a detailed explanation regarding the pointer update and the starvation problem). In addition, as the switch size increases the number of elements at each output arbiter's circular list increases and these elements can be positioned in any order. Consequently, the time required to serve all elements in the list will increase and the short-term unfairness will manifest itself clearly.

iCBFS vs. WPIM

Although WPIM is fair over a time scale larger than the frame size (typically 1000 slots [12]), it is unfair over shorter time scales. As shown in Figure 2 (d) at the rightmost point of the graph, $f_1(0,0)$, $f_2(1,0)$, $f_3(2,0)$, and $f_4(3,0)$ receive 13, 26, 28, and 32 percent of the bandwidth, respectively. This unfairness is caused by the uniform random selection used at the output arbiters. In essence, all flows with available credit are treated equally until their credit is used up. Consequently, flows with higher bandwidth reservations than others receive their differential bandwidth share only at the end of a frame, whereas iCBFS distributes this differential bandwidth share uniformly over the entire time scale.

When all flows use their reserved credits, WPIM reduces to PIM and all unreserved bandwidth is distributed equally among all inputs [12], whereas iCBFS distributes unreserved bandwidth among all inputs in proportion to their reservations.

In summary, iCBFS provides fair bandwidth among flows in proportion to their reservations. iCBFS provides significantly better fairness than WiSLIP, WPIM, and iDRR over time scales less than a frame size. We emphasize the as the switch size increases, the frame size required to serve all the input ports increases proportionally and the short-term unfairness problem manifests itself clearly in increased jitter. The simple case of a 4×4 switch was only used to simplify the presentation. In addition, as the link speed increases the frame size would also increase.

D2. Uniform Traffic

In addition to providing fair bandwidth among flows in proportion to their reservations, we evaluated the performance of iCBFS when all flows have equal reservations. Figure 3. shows the average delay of iCBFS compared to other scheduling schemes under uniform i.d.d. Bernoulli traffic. Similar to other scheduling schemes, iCBFS is capable of achieving asymptotically 100% throughput under uniform traffic. Under uniform traffic, all schedulers behave similarly. However, this traffic model is not realistic for internet routers.

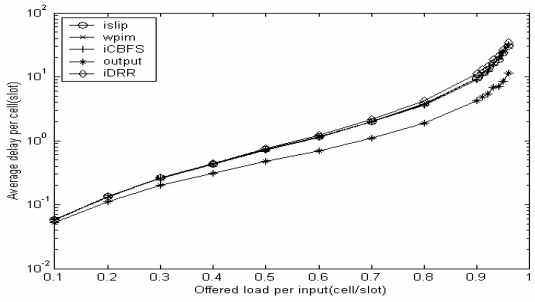


Figure 3. Performance of iCBFS, iSLIP, WPIM, and Output queueing under uniform traffic.

IV. A PORT-BASED FAIR SCHEDULING ALGORITHM

There is a trade-off in the design of high-speed switches between fairness among flows and simplicity of hardware design required for high-speed implementation. On the one hand, flow-based-scheduling guarantees fairness among flows by isolating non-conforming flows and provides bandwidth guarantee to individual flows; on the other hand, it makes hardware design relatively complex, and does not scale well as the number of flows grows. Port-based scheduling [12] allows simple hardware implementation at the cost of a coarse

granularity of bandwidth guarantee. Rather than tracking *individual* flows at each input port, a port-based scheduler tracks the *aggregate* bandwidth reservation at each input port. Consequently, the complexity of a port-based scheduler is proportional to the switch size instead of the number of flows, which can be significantly larger. Thus, port-based scheduling can reduce the complexity of the scheduler considerably.

We propose to divide scheduling into two layers: CBFQ per virtual queue at the input side (labelled $V_{CBFQ}(i,j)$ for packets at input port i destined to output j), and a port-based scheduler, PCBFQ_j, at each output port j. $V_{CBFQ}(i,j)$ can be implemented in software using DRAM, and PCBFQ_j can be easily implemented in hardware. Intuitively, using this hierarchical scheduling scheme, the complexity of the original CBFQ_j engine at each output j is distributed among all the input ports and the PCBFQ_j only deals at the abstraction of port-based scheduling; thus simplifying the design considerably.

The PCBFQ_j, at each output port j, maintains a counter K_{ij} and a bandwidth share S_{ij} for the aggregate bandwidth reservation from input i to output j. Similar to iCBFS, each input i arbiter uses a set of counters C_{ij} to track the aggregate quota for each $input_i - output_j$ pair. A VOQ_{ij} becomes candidate if $K_{ij} \geq 1$. The port-based of iCBFS, called iPCBFS would execute the request, grant, and accept stages as described in the previous section.

Note that the simulation results for iPCBFS are identical to the simulation results for iCBFS described section III.D.

V. HARDWARE COMPLEXITY

We assume a CRCW PRAM model such that all executed operations are $O(1)$ time, and estimate the complexity of iCBFS for an $N \times N$ switch. In the iCBFS algorithm, each time an output arbiter selects a flow to send a grant signal or an input arbiter selects an output to send the accept signal the computations performed are $O(N)$.

The priority sort of the flows to select the flow with the largest share at each output arbiter changes only at the burst level timescale. That is, each flow's share is allocated upon the admission of a new flow and does not change during its lifetime. Consequently, the sorting consists only of extracting the pre-ordered list of active flows from a static list. Note that iDRR also maintains a pre-ordered list of active flows such that the flow with the smallest reservation is always used in calculating the quota for other flows.

During the grant stage of iCBFS, the counter values do not need to be sorted according to their values. Consequently, we only need to compare each counter's value to 1. We point that all the counters' update and comparison operations can be implemented using integers.

In iCBFS, each output arbiter needs to maintain a counter for each flow, whereas in iPCBFS the number of counters is fixed and equals N. Similar to all algorithms based on πRGA policy, both iCBFS, and iPCBFS may require up to N iterations in the worst case and an average of $O(\log N)$ iterations for uniform traffic.

VI. CONCLUSION

We proposed iCBFS, a flow-based fair scheduling algorithm for internet routers with input-queued switches. We showed through simulation that iCBFS can fairly allocate bandwidth in proportion to flows' reservations and provide considerably better fairness over short-time scales compared to all other schemes; thereby, iCBFS reduces the jitter and delay for multimedia services like VOIP and video-on-demand. In addition, the algorithm achieves 100% throughput for uniform traffic. To simplify the implementation complexity of iCBFS, we proposed a port-based version of iCBFS, iPCBFS, which is simpler to implement in hardware.

REFERENCES

[1] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High Speed Switch Scheduling for Local Area Networks," ACM Trans. Computer Systerms, vol. 11, no. 4, pp. 319-352, Nov. 1993.

[2] B.Bensaou, D.H.K.Tsang, King Tung Chan, "Credit-based fair queueing (CBFQ): a simple service-scheduling algorithm for packet-switched networks," IEEE/ACM Trans. Networking, vol. 9, no. 5, pp. 591 –604, October 2001.

[3] H. J. Chao, Y. R. Jenq, X. Guo, and C. H. Lam, ``Design of Packet Fair Queuing Schedulers Using a RAM-based Searching Engine," in IEEE J. Select. Areas Commun., pp. 1105-1126, June 1999.

[4] H. J. Chao and N. Uzun, "A VLSI Sequencer chip for ATM traffic shaper and queue manager," IEEE J. Solid-State Circuits, vol. 27, no. 11, pp. 1634-1643, Nov. 1992.

[5] P. Goyal, H. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet-switching networks," in Proc. SIGCOMM, 1995, pp. 157-168.

[6] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division switch," IEEE Trans. Commun., vol. 35, pp. 1347–1356, Dec. 1987.

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

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

[9] Ni Nan, L.N Bhuyan, "Fair scheduling in Internet routers," IEEE Trans. Computers, vol. 51, no. 6, pp. 686 –701, June 2002.

[10] A. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," IEEE/ACM Trans. Networking, vol. 1, no. 3, pp. 344-357, June 1993.

[11] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. McCallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. D. Troxel, D. Waitzman, and S. Winterble, "A 50-Gb/s IP router," IEEE/ACM Trans. Networking, vol. 6, pp. 237–248, June 1998.

[12] D. Stiliadis and A. Varma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," Proc. IEEE INFOCOM '95, pp. 960-968, Apr. 1995.

[13] Xiao Zhang, and Laxmi N. Bhuyan, " Deficit Round Robin Scheduling for Input-Queued Switches," IEEE J. Selected Areas in Comm., vol. 21, no. 4 , pp. 584 –594, May 2003.