Weighted Fairness in Buffered Crossbar Scheduling

Nikos Chrysos†, Manolis Katevenis†
Institute of Computer Science - Foundation for Research and Technology - Hellas (FORTH)
ICS-FORTH, P.O. Box 1385, Vassilika Vouton, Heraklion, Crete, GR-711-10 Greece
http://archvlsi.ics.forth.gr/bufxbar/ - {nchrysos, katevenis}@ics.forth.gr

***Abstract*— The crossbar is the most popular packet switch architecture. By adding small buffers at the crosspoints, important advantages can be obtained: (1) Crossbar scheduling is simplified. (2) High throughput is achievable. (3) Weighted scheduling becomes feasible. In this paper we study the fairness properties of a buffered crossbar with weighted fair schedulers. We show by means of simulation that, under heavy demand, the system will allocate throughput in a weighted max-min fair manner. We study the impact of the size of the crosspoint buffers in approximating the weighted max-min fair rates and we find that a small amount of buffering per crosspoint (3-8 cells) suffices for the maximum percentage discrepancy, to fall below 5% for 32×32 switches.**

1 . INTRODUCTION

Switches, and the routers that use them, are the basic building blocks for constructing high-speed networks that employ point-to-point links. As the demand for network throughput keeps climbing, switches with an increasing number of faster ports are needed. At the same time, mechanisms are sought for higher sophistication in quality of service (QoS) guarantees. The crossbar is the simplest fabric for high-speed switches. It is the architecture of choice for up to several tens of ports, although for higher port counts, N, the order of the crossbar cost, $O(N^2)$, makes other alternatives more attractive. The hardest part of a high-speed crossbar is the *scheduler* needed to keep it busy.

With virtual-output queues (VOQ) at the input ports, the crossbar scheduler has to coordinate the use of $2N$ inter-dependent resources. Each input has to choose among N candidate VOQ's, thus potentially affecting all N outputs; at the same time, each output has to choose among potentially all N inputs, thus making all $2N$ port schedulers depend on each other. Known architectures for high-speed crossbar scheduling include [1] [2] [3]; their complexity and cost increases significantly when the number of ports rises, thus negatively affecting the achievable port speed.

An advanced form of quality of service (QoS) architecture uses *weighted round-robin (WRR)* scheduling –often in the form of *weighted fair queueing (WFQ)* [4]– which takes weight factors into consideration when determining "equality". This type of scheduling is needed when some customers pay more than others, or when each flow is an aggregate of a different number of sub-flows and we wish to treat sub-flows equally. The weight factors may be static (during the lifetime of connections), or they may change dynamically, e.g. in the case of varying aggregate membership, or when we want inactive sub-flows not to count toward the weight of their aggregate.

Existing crossbar schedulers either ignore QoS issues, or provide only static priorities and round-robin scheduling [2] [3]. *Weighted* round-robin behavior is very hard to achieve in crossbar schedulers while still maintaining high crossbar utilization (near-maximal matches) [5]: many iterations are needed to yield high-occupancy matches, thus severely limiting the port speed at which these schedulers can be used. The solution commonly used, today, is to provide significant *internal speedup:* the crossbar port rate is higher than line rate by a factor of f, considerably greater than 1.

While internal speedup is a good solution, it does incur significant cost: *(a)* the crossbar is more expensive (f times higher throughput), *(b)* the buffer memories are more expensive ($(1+f)/2$ times higher throughput), *(c)* the crossbar scheduler must run f times faster and *(d)* the number of buffer memories is doubled: besides input queues, output queues are needed as well [1].

An alternative solution, with the potential to yield both faster and less expensive switches, is to use *buffered crossbars*. By adding even small amounts of buffer storage at the crosspoints, the scheduling problem changes radically and is dramatically simplified: the $2N$ schedulers, N at the inputs and N at the outputs, work *independently* of each other, since each of them deals with only a single resource. The $2N$ schedulers are still coordinated but only over longer time-scales, through backpressure feedback from the crosspoint buffers. It has been shown that such buffered crossbars allow efficient *distributed scheduling* schemes[6][7][8].

Of-course, the major cost of buffered crossbars is that N^2 additional buffers are needed, one at each crosspoint. For a 32×32 crossbar with 4 priority levels and two 64-byte cells of storage per crosspoint and priority level, the total buffer space in the crossbar is 8 K cells or 4 Mbits, which is clearly feasible in current ASIC technology. Up to a few years ago, this much memory was very expensive or even unrealistic for a single-chip implementation of the crossbar, and this was the reason why crossbars were usually bufferless, and the majority of research concerned such bufferless crossbars. Today however,

[1]Note that output queues are also needed for cell-to-packet reassembly, and for sub-port demultiplexing, when desired. However, buffered crossbar have the potential of eliminating cell-to-packet reassembly, because they can operate directly with variable-size packets.

† The authors are also with the Dept. of Computer Science, University of Crete, Heraklion, Crete, Greece.

this amount of SRAM is quite feasible in crossbar ASIC's. Furthermore much larger buffer memories are soon becoming feasible: for example, by dedicating 100 mm^2 of a 0.13 μm ASIC to SRAM, 40 Mbits of buffer memory will become available (at 2.5 μm^2 per bit [9]); this suffices for a 64 × 64 crossbar, with 4 cells per crosspoint and per priority level.

Concerning power consumption, although the number of memories is N^2, at most $2N$ of them are active at any given time, for unicast traffic. Also, power consumption in the buffer memories that are introduced will normally be lower than in the crossbar buses that already existed, because internal memory buses are much shorter than crossbar buses, while the throughput of both types of buses is the same (considering separate write and read buses in the memories).

In this paper we study a buffered crossbar with a WRR/WFQ scheduler at each input and output. First, we make the assumption of persistent sources, i.e. input-output connections with either constantly empty or constantly full VOQs. The assumption about persistent flows is what models the short-term behavior of a network under transient overload. In the long run, wide-area or end-to-end flow control will hopefully adjust the rates of individual flows so that the egress links of the switch are not oversubscribed. If these output links were never oversubscribed, the scheduling policies inside the switch would not matter and buffer memories would not be needed. However, short-term overloads do appear, due to the variability and unpredictability of traffic. Modern commercial switches have hundreds of megabytes of buffer storage, because they anticipate transient overload periods up to a fraction of a second. During such overload periods, it is the schedulers in the switch that allocate output bandwidth to the contending flows, thus determining the QoS that these flows receive. We model the behavior of the switch during the overload periods using persistent flows for the non-empty queues and inactive flows (or equivalently zero weight factors) for the empty queues. Using a simplified fluid model, we show that this system will allocate service in a weighted max-min (WMM) fair manner (section 2). Simulations verify this property and also examine the amount of buffering per crosspoint for sufficiently good approximation of the ideal WMMF allocation (section 3). We also study which weight factor combinations yield the best WMMF approximation, and the impact of the scheduling disciplines used at the inputs and the outputs. Finally, we present preliminary results for the case when some flows are source limited to a level just below their WMM fair share – that is some flows are not persistent. Section 4 compares our results to previous work.

2 . System

Figure 1 shows the model of the $N \times N$ switch system that this paper deals with. We consider one of the priority levels in a system; our results apply to any priority level in a real system, after subtracting from the link capacities the throughput already allocated to flows at higher priority levels. There are *virtual output queues (VOQ)* at the N inputs, containing fixed-size cells. The core of the system is an $N \times N$ crossbar containing N^2 small queues, one per crosspoint. Backpressure flow-control ensures that the crosspoint buffers will never overflow.

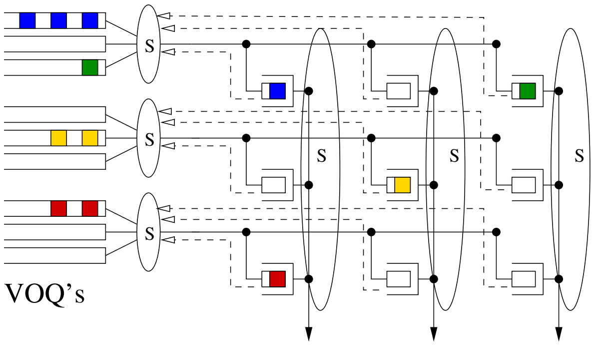

Fig. 1. System model assumed in this paper.

Flows are determined as input-output pairs. Each flow f corresponds to a crosspoint buffer and has a unique weight factor w_f. All schedulers use the Start-time Fair Queuing (SFQ) [10] scheduling discipline, which is a variant of WRR/WFQ. SFQ has complexity $O(logN)$, and can be implemented at high speed, e.g. using a tree of comparators exploiting bit-level parallelism [11]: the minimum of 256 twenty-four-bit numbers can be found in 4.5 ns in 0.18 μm CMOS technology. The backpressure information overhead is on the order of N bits per input and per time-slot.

2.1. Analysis under Persistent Sources

Consider that all N^2 VOQs corresponding to the N^2 flows are either continuously full (call the respective flow "active") or continuously empty (call the respective flow "idle"). Under this assumption, the only reason for an active flow to be ineligible at its input server is that backpressure is on. At the output server an active flow is ineligible if its crosspoint buffer is empty.

Distributed crossbar scheduling operates roughly as follows. Initially, when all crosspoint buffers are empty, each input scheduler serves each flow according to its fair share. The schedulers at different inputs operate independently; even if they happen to transmit cells to the same output in the same time slot, the crosspoint buffers keep the cells until the output scheduler reads them one by one. Output schedulers are initially forced to serve the few non-empty crosspoint buffers. As more and more buffers fill up, output schedulers start enforcing their fair shares.

The fair share of a flow at the output will, in general, differ from its fair share at the input. If the output fair share is higher, the output scheduler will attempt to read from the buffer more frequently than the input scheduler writes into it. As a result, the buffer will often be empty, and the flow will often be ineligible for the output scheduler; thus, the bandwidth of such a flow is dictated by the input scheduler allocation. On the other hand, if the output fair share of a flow is lower than its input counterpart, the buffer will gradually fill up, because the output reads it less frequently than the input writes into it. When the buffer fills up, backpressure will make this flow ineligible at the input, thus reducing its input service-rate until it equals the rate dictated by the output scheduler. In this way, in the long run, the service-rate allocated to each flow becomes the smaller of the rates that its input or output allocate to it. Because schedulers are work conserving, they will always serve a flow as long as there is at least one eligible. Thus, the bandwidth that remains unused by ineligible flows is distributed to the eligible ones according to the latter flows' fair share. Eventually, this redistribution will yield WMM fair allocations, as discussed below in more detail.

1) Fluid Model: The behavior of the system and its analysis are simplified when we replace discrete cells with an infinitely divisible fluid, and WRR/WFQ schedulers with ideal *generalized processor sharing (GPS)* servers [12]. In this fluid model, for each flow f, there is an input GPS server and an output GPS server. The rates that these two servers allocate to f may differ *only* during times when f's buffer is neither empty nor full; when the buffer fills up or is emptied, the higher of the two rates is forced to become equal to the lower one.

Theorem *1: Assume (a) a fluid model with GPS servers for a buffered crossbar; (b) finite size of buffer space at each crosspoint; and (c) all flows have either continuously empty or continuously full VOQs. Then, all flows will receive exactly their weighted max-min fair (WMMF) rates.*

The proof of this theorem, which can be found in an extended version of this paper [13], shows that flows reach their final stable state in the same sequence that the WMMF algorithm works, i.e. in the sequence of non-decreasing utility under WMM fairness ($U_f = \frac{rate_f}{weight_f}$). When a flow f stabilizes to its WMM fair rate, its crosspoint buffer becomes either full or empty. For a full buffer, f's output fair share limits its service rate (output share is smaller than input share, and f is bottlenecked at the output) and its remaining input share is transferred to the other flows that use the same input and deserve higher utility than f under WMM fairness. The result is the same when f's input fair share limits its rate. So, flows "find" their final input and output fair shares only after all input or output "neighbor" flows that deserve smaller WMM fair utility have stabilized to their WMM fair rates. This shows that a rate discrepancy from the WMM fair rate for a flow can be propagated to neighbor flows with greater WMM fair utility.

2) Discrepancies in a Non-Fluid System: A real system with discrete cells differs from the above ideal fluid model. The WRR/WFQ scheduler assumed in section 2 will precisely allocate rates according to the fair shares only in the long run, and only if the set of eligible flows stays fixed. For the set of eligible flows to stay fixed, crosspoint buffers have to be large enough so that normally-full buffers never empty, and normally-empty buffers never fill up. Consider a normally-full buffer: although its input scheduler is supposed to fill it more frequently than its output scheduler empties it, actual service is not perfectly smooth, and this fluctuation may cause the input scheduler to occasionally be late in refilling the buffer while the output scheduler may occasionally be early in emptying the buffer. As stated in the previous paragraph, an occasional rate discrepancy of a single flow can be propagated to other flows as well.

3 . SIMULATION RESULTS

3.1. Simulation Environment

Our simulator operates at time-slot (the time required to receive or transmit a cell) granularity. The simulator uses unit-delay backpressure: consider an input scheduler that decides, in time-slot t_i, to serve a flow f whose crosspoint buffer was empty; f becomes eligible for the decision made by its output scheduler in time-slot t_{i+1}. Similarly, consider an output scheduler that decides, in t_{i+1}, to serve a flow g whose crosspoint buffer was full; g becomes eligible for the decision made by its input scheduler in t_{i+2}. Under these assumptions, two cells worth of buffer space per crosspoint suffices so that a flow experiencing no competition at the input and at the output to be served at full link rate (1 cell per time-slot).

We use *Relative Error*, RE, as our metric to evaluate convergence accuracy to WMM fairness. Given a simulation interval, the Relative Error of the rate for each flow f is defined as:

$$RE_f = \frac{|ActualService_f - FairService_f|}{FairService_f}$$

where $ActualService_f$ is the number of cells of f that exited the switch during the simulation interval, and $FairService_f$ is the rate allocation to f according to WMM fairness, multiplied by the length of the simulation interval, i.e. it is f's expected service in number of cells. The RE_f measurements started 500K time-slots after the beginning of simulation, and extended as long as needed to reach a confidence interval of 0.04 with confidence 95%. We then extracted the *average* and the *maximum* (worst-flow) of the RE_f values over all active flows f. Each simulation was repeated 10 to 40 times, with different sets of (randomly chosen) weight-factors each time; we report the mean *average* and the mean *maximum* RE_f's of these runs.

In the following results, we configured each flow independently to be either active or idle with probabilities l and $1-l$ respectively. In all results except the one in section 3.6, all active flows are fed by persistent VOQ sources.

3.2. Effect of Weights Distribution

First we present the convergence accuracy under three different weight distributions for the active flows. In the configuration called *uniform*, all active flows have a random weight factor picked uniformly in the interval $[1, 1001]$. In the *skewed-j* configuration, the weight of a flow from input i to output j is chosen through the following random process $[1 + 10j + \text{unif\_rand}(0, 10j)]$, whereas in the configuration called *skewed-j^2* through $[1 + 10j^2 + \text{unif\_rand}\big(0, 10j^2\big)]$. We used skewed distributions in order to create imbalanced weight factors: a flow to an output with high index will probably have

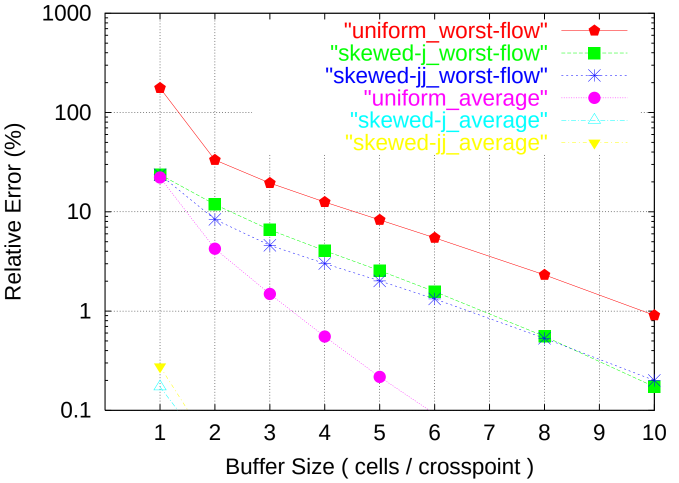

Fig. 2. Average and worst-flow RE for different weights distributions; 32×32 switch, all flows active.

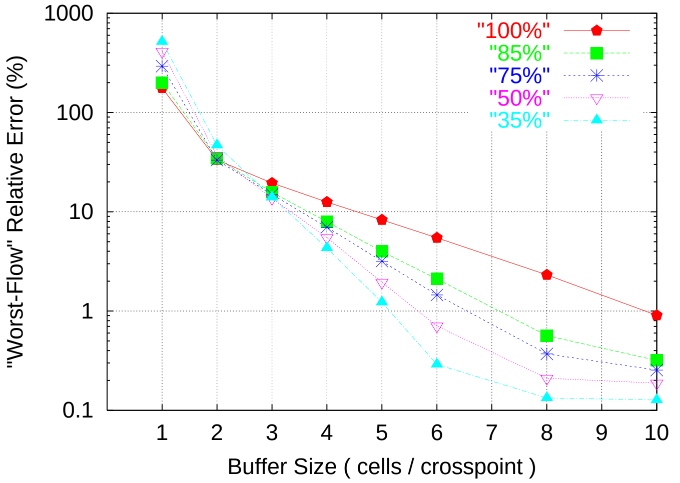

Fig. 3. Worst-flow RE over all flows, for various activity percentages; *uniform* weight-factor distribution, 32×32 switch.

a large weight, and thus a large input fair share. However, since most of the flows using the same high-index output have large weights, f's rate will be limited by its output fair share; hence a large portion of its input fair share must be redistributed to the other flows of the same input.

In figure 2 we plot both the average and the worst-flow RE for the three distributions discussed above, when all flows are active. Under all configurations, we see that a buffer size of 4 cells per crosspoint suffices to drive the average RE below 1 %. In contrast to our first intuition, the skewed distributions yielded much better accuracy than the uniform one. We hypothesize that this is due to all flows being bottlenecked at the outputs. A flow's approximation to WMM fairness is mostly affected by small discrepancies, when these occur at the flow's bottleneck server, since the rate of the flow is dictated by that server. So when all flows are bottlenecked at the outputs, occasional discrepancies in one flow's service, will only affect its output neighbors; input neighbors are not affected considerably, since they are bottlenecked at outputs. By contrast, in the uniform weights case, some flows are bottlenecked at the inputs and some at the outputs; occasional discrepancies in one server can propagate to more flows.

Besides their obvious importance for QoS accuracy, these results also show how well a buffered crossbar can sustain full output utilization for those outputs for which enough input demand exists. For the outputs for which the fair rates add up to 1, since the actual rates are within 1 % of the fair rates, it follows that utilization is 99% or better.

3.3. *Effect of Flow Activity*

Figure 3 plots the maximum value of RE over all active flows, with uniformly chosen weight factors, under four different activity probabilities, l = 100%, 85%, 75%, 50%, 35%. The average RE is not affected considerably by the percentage of idle flows, and we do not plot it.

We see that buffer sizes of 6 to 8 cells yield worst-case errors bellow 5 % for any activity ratio. The system better

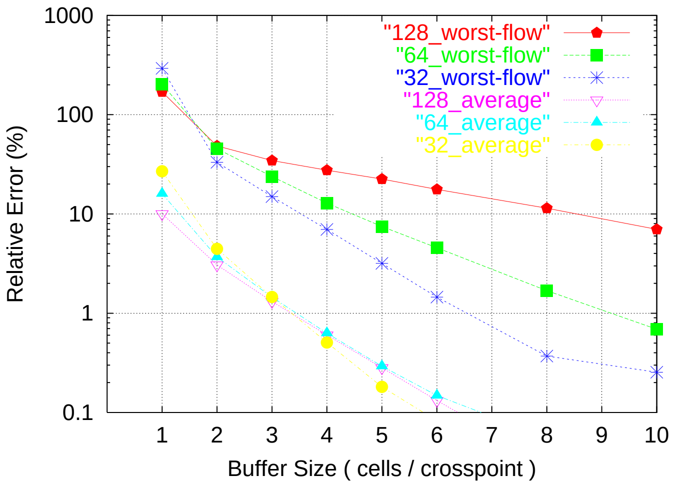

Fig. 4. Average and worst-flow RE for various switch sizes; *uniform* weight-factor distribution, 75% of flows active.

approximates fair allocations when there are less active flows. This tendency can be attributed to the smaller number of flows that each WRR/WFQ scheduler has to consider when there are more inactive flows; fewer flows per scheduler result in less jitter in their service time, therefore smaller probability for a normally-full buffer to empty or a normally-empty buffer to fill up.

3.4. *Effect of Switch Size*

Figure 4 plots the relative error (average and worst-flow) for various switch sizes: 32×32, 64×64, and 128×128. We observe that larger switches yield worse RE for the worst flow. On the other hand, the average RE is rather insensitive to switch size. We conclude that larger switches have a few flows with reduced accuracy and many flows with good accuracy in the process of finding WMM fairness. The reduced accuracy of some flows can be attributed to larger jitter in the WRR/WFQ schedulers (due to more flows in each scheduler).

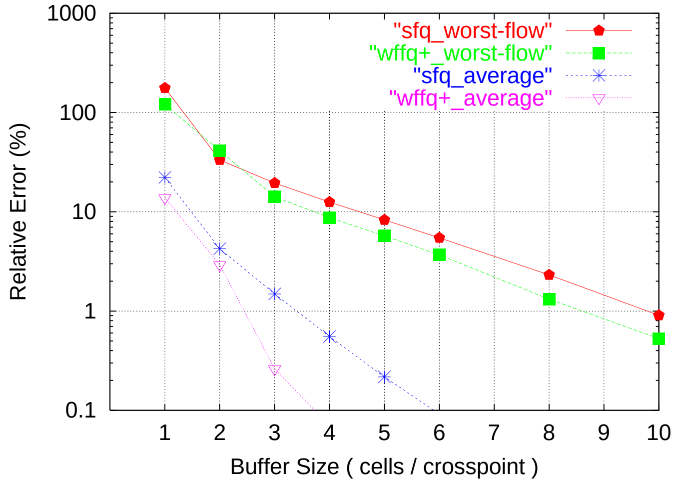

Fig. 5. Worst-flow and average RE over all flows under WF^2Q+ and SFQ scheduling disciplines; *uniform* weight-factor distribution, all flows active, 32 × 32 switch.

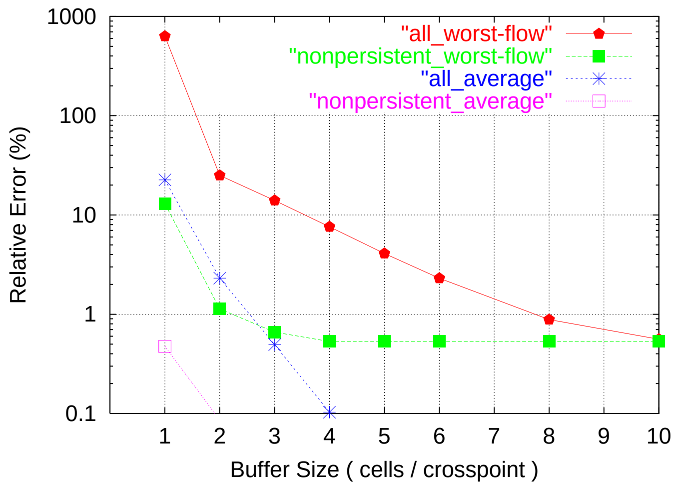

Fig. 6. Average and worst-flow RE for all_active and the non_persistent ones in separate; *uniform* weight-factors distribution, active percentage 75%, 32 × 32 switch

3.5. Effect of WRR/WFQ Schedulers Accuracy

While the SFQ service discipline that we used so far is cost effective, other alternatives for WRR/WFQ scheduling have proved to be more accurate. Figure 5 plots the average and worst-flow RE for SFQ versus WF^2Q+ [15] schedulers at the inputs and outputs. WF^2Q+ provides better accuracy, due to the smaller jitter in the allocated service; however, the improvement is not impressive.

3.6. Non-Persistent Sources

All simulations presented so far use persistent sources, i.e. sources that request more service than their WMM fair share. In our (ongoing) research, we are studying the transient behavior of the system when sources rates change between levels above and below their fair share [17]. We are also studying the effect of sources with average rate below their fair share but with probabilistic cell arrival times; in this context, we configured one flow ($f_{i->j}$) at each input (i) to be fed by a Bernoulli i.i.d VOQ source, while the VOQs of the remaining active flows are continuously full. The average incoming rate of non persistent flows is configured slightly below their WMM fair share [2]. In Fig. 6, we plot the worst-flow and the average RE separately, for all active flows and for the non-persistent ones. We see that with 3 cell buffers per crosspoint, flows fed by non-persistent sources receive service rate approximately equal to their WMM fair rate (*nonpersistent_worst-case* RE is less than 1%). With 5-6 cells per crosspoint the overall capacity of the switch is allocated in a WMM fair manner (*all_worst-case* RE is less than 5%, *all_average* RE is less than 0.1%). This means that the excess capacity, left-over by the flows with low incoming rate, has been fairly redistributed to the persistent ones.

4 . Previous Work & Contribution

Hahne [16] proved that per-flow buffering, per-flow back-pressure, and round-robin scheduling indeed yields max-min fairness, although, in some pathological cases, very large buffers may be needed for that. The present paper differs from [16] in that *(a)* we consider *weighted* rather than plain round-robin and *weighted* rather than plain max-min fairness; *(b)* we simulate small-buffer effects.

Stephens and Zhang [6] studied and simulated buffered crossbars with WRR/WFQ schedulers, and proved their ability to provide delay bounds to properly policed flows. These delay bounds are based on the minimum rate guaranteed for each flow, which is the minimum, over all links traversed by the flow, of the ratio of the flow's weight over the sum of the weights of all flows traversing the link. They do not consider the allocation of the excess bandwidth that results, when flows' minimum rate guarantees do not fully occupy the capacity of the switch. The present paper differs from the above, in that we study the allocation of excess bandwidth and how well it approximates WMM fairness. Recently, Javidi e.a. [7] examined buffered crossbars with longest-queue-first input schedulers and RR output schedulers, and showed full output utilization under some assumptions. Also Chao e.a. [8] studied the throughput of a buffered crossbar with RR input and output schedulers, under uniform and non-uniform input loads. They examine the improvement of using additional buffering at the crosspoints and also prove that with a small internal speed-up, 100% throughput can be obtained even under non-uniform loads. We consider their work as complementary to ours: we use weighted rather than plain round-robin scheduling, and we study the fairness properties under heavy load.

We are currently working on transient periods that occur when flows switch state from active to idle and vice-versa, or when the weight of a flow changes. Our preliminary results

[2]the minimum of their guaranteed service rate at the input and the output servers,

$$r_{f_{i->j}} = \min\left(\frac{w_{f_{i->j}}}{\sum_{k=1}^{N} w_{g_{i->k}}}, \frac{w_{f_{i->j}}}{\sum_{k=1}^{N} w_{g_{k->j}}}\right).$$

Because excess bandwidth exists, this rate is a bit lower than their fair share.

concerning delay and unfairness during such transient periods can be found in [17].

References

[1] T. Anderson, S. Owicki, J. Saxe, C. Thacker: *"High-Speed Switch Scheduling for Local-Area Networks"*, ACM Trans. on Computer Systems, vol. 11, no. 4, Nov. 1993, pp. 319-352.

[2] R. LaMaire, D. Serpanos: *"Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues"*, IEEE/ACM Trans. on Networking, vol. 2, no. 5, Oct. 1994, pp. 471-482.

[3] N. McKeown: *"The iSLIP Scheduling Algorithm for Input-Queued Switches"*, IEEE/ACM Trans. on Networking, vol. 7, no. 2, April 1999, pp. 188-201.

[4] A. Demers, S. Keshav, S. Shenker: *"Design and Analysis of a Fair Queueing Algorithm"*, Proc. of ACM SIGCOMM Conf.,Texas USA, Sep. 1989, pp. 1-12.

[5] N. Ni, L. N. Bhuyan: *"Fair scheduling for Input Buffered Switches"*, citeseer.nj.nec.com/482342.html.

[6] D. Stephens, H. Zhang: *"Implementing Distributed Packet Fair Queueing in a scalable switch architecture"*, Proc. INFOCOM'98 Conf., San Francisco, CA, March 1998, pp. 282-290

[7] T. Javidi, R. Magill, and T. Hrabik: *"A High-Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric"* Proc. IEEE Int. Conf. on Communications (ICC'2001), Helsinki, Finland, June 2001, vol. 5, pp. 1586-1591.

[8] R. Rojas-Cessa, E. Oki, and H. Jonathan Chao: *"CIXOB-k: Combined Input-Crosspoint-Output Buffered Switch"*, Proc. GLOBECOM '01, vol. 4, pp. 2654-2660

[9] www.tsmc.com/

[10] P. Goyal, H. M. Vin and H. Cheng: *"Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks"*, Proc. of ACM-SIGCOM 96, pages 157-168, Palo Alto, CA, August 1996.

[11] K. G.I. Harteros: *"Fast Parallel Comparison Circuits for Scheduling"*, Master of Science Thesis, University of Crete, Greece; Technical Report FORTH-ICS/TR-304, Institute of Computer Science, FORTH, Heraklio, Crete, Greece, 78 pages, March 2002; http://archvlsi.ics.forth.gr/muqpro/cmpTree.html

[12] A. K. Parekh and R. G. Gallager: *"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The single Node Case"*, IEEE/ACM Transactions on Networking, vol. 1, no. 3, June 19.

[13] N. Chrysos and M. Katevenis: *"Weighted Fairness in Buffered Crossbar Scheduling – Extended Version"*, http://archvlsi.ics.forth.gr/bufxbar/wf_bufxbar_extended.ps

[14] Z. Cao, E. W. Zegura: *"Utility Max-Min: An application-Oriented Bandwidth Allocation Scheme"*, Proceedings of IEEE INFOCOM 99, New York, NY, March, 1999.

[15] J. C. R. Bennett, H. Zhang: *"Hierarchical Packet Fair Queueing Algorithms"*, IEEE/ACM Transactions on networking, Vol. 5., No. 5, October 1997

[16] E. Hahne: *"Round-Robin Scheduling for Max-Min Fairness in Data Networks"*, IEEE Journal on Selected Areas in Communications, vol. 9, no. 7, September 1991.

[17] N. Chrysos and M. Katevenis: *"Transient Phenomena of a Buffered Crossbar Converging to Weighted Max Min Fairness"*, http://archvlsi.ics.forth.gr/bufxbar/bufxbar_chrys_02-08.ps