Transient Behavior of a Buffered Crossbar Converging to Weighted Max-Min Fairness

Nikos Chrysos and Manolis Katevenis
Institute of Computer Science - FORTH, and University of Crete, Greece
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 topology. By adding small buffers to each crosspoint, important advantages can be obtained: scheduling is dramatically simplified and weighted round robin (WRR) or weighted fair queueing (WFQ) becomes feasible; variable size packets can be switched; no internal speedup is needed; and, in many cases, no output buffer memories are needed. This paper studies the fairness properties of distributed WRR/WFQ scheduling in such a buffered crossbar. We provide arguments for why the system converges to weighted max-min (WMM) fairness, and we study the factors that affect stabilization delay after changes in offered load or weight factors. We simulate the system and observe how close real rates come to the theoretical WMM fair allocations: with buffer sizes of 2 to 5 cells per crosspoint, the average rate discrepancy is below 1%; the worst-case discrepancy falls below 4% with buffer sizes of 4 to 8 cells. Transient behavior simulations verified that stabilization delay is proportional to buffer size, and inversely proportional to the magnitude of the change in bandwidth allocation. In conclusion, buffered crossbars, which are technically feasible today, offer important advantages, which include excellent quality of service guarantees.**

Topics Keywords: Switches and switching, (Scheduling, QoS).

Methods Keywords: System design, Simulations.

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 topology for high-speed switches. It is the architecture of choice for up to at least 32 or 64 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.

1.1. Crossbar Scheduling, QoS, and Internal Speedup

With virtual-output queues (VOQ) at the input ports, the crossbar scheduler has to coordinate the use of $2N$ interdependent 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.

Quality of service (QoS) is an increasingly important concern in networks and switches. A simple form of QoS is based on static priorities: serve all high-priority packets before serving any lower-priority packet. This works well between some types of traffic with widely differing characteristics, but it is clearly inappropriate among flows of a similar type, and may even lead to starvation for the lower priority traffic. Another form of QoS is based on round-robin scheduling, and is appropriate for flows that are "equal" to each other.

An advanced form of QoS 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 subflows and we wish to treat subflows 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 subflows not to count towards the weight of their aggregate.

Existing crossbar schedulers either ignore QoS issues, or provide only priorities and/or round-robin-like 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] [6]: many iterations are needed to yield high-occupancy matches,

thus severely limiting the port speed at which these schedulers can be used; in addition, [6] that computes stable marriage matchings, needs a sorting operation per port and per time-slot.

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. In this way, *(a)* imperfect crossbar scheduling is acceptable, since an average utilization of $1/f$ for the crossbar outputs suffices for the egress lines to get fully utilized; *(b)* we can accommodate the rate increase that occurs when variable-size packets are segmented into fixed-size cells; and *(c)* the emphasis for QoS enforcement is shifted to the egress-line sub-system, since queues now tend to build up on the output side of the crossbar (*combined input-output queueing (CIOQ)*). Using the latter property, one can implement e.g. WFQ on the output queues, although, for traffic overloads higher than f, queues also build up on the input side, where crossbar schedulers cannot typically implement WFQ.

While internal speedup is a good solution, it does incur significant cost: the crossbar is more expensive (f times higher throughput), the buffer memories are more expensive ($(1+f)/2$ times higher throughput), and the number of buffer memories is doubled –besides input queues, output queues are needed as well. (Note that output queues are also needed for cell-to-packet reassembly, and for subport demultiplexing, when provided.) An alternative solution, with the potential to yield both faster and less expensive switches, is to use *buffered crossbars*.

1.2. Buffered Crossbars and their Advantages

The above discussion concerned crossbar switches with purely combinatorial crosspoint logic, i.e. without any storage at the crosspoints. By adding, however, 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, become *independent* of each other, and each of them deals with only a single resource. The $2N$ schedulers are still coordinated with each other, but only indirectly and over longer time-scales, through backpressure feedback from the crosspoint buffers. Hence, such *buffered crossbars* allow efficient *distributed scheduling* schemes. In turn, efficient scheduling eliminates one of the reasons for internal speedup.

Another important advantage of buffered crossbars is their capability to directly switch *variable-size packets*, without prior segmentation into fixed-size cells, given that schedulers are now independent, and do not have to operate in synchrony anymore. This eliminates the other reason for internal speedup. In turn, when the crossbar operates without internal speedup and without the need to reassemble packets from cells, the two main reasons for output queues have gone away, thus allowing significant cost savings[1]. In this (first) paper, though, we deal exclusively with fixed-size-cell traffic.

The amount of buffering needed per crosspoint is small –on the order of the line rate times the backpressure round-trip delay, which often amounts to one or a few cells. 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. 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).

1.3. Fairness in Distributed WFQ and Related Work

Given the above advantages of buffered crossbars, we expect that they should gradually displace unbuffered crossbars, now that IC technology makes such internal buffering feasible. Hence, it becomes important to study the distributed scheduling that becomes possible in buffered crossbars, especially given its capability to provide WRR/WFQ service, thus allowing the elimination of output queues.

In the eighties, Gallager [7] and Katevenis [8] independently proposed the use of per-flow buffering, per-flow backpressure, and round-robin scheduling for fairly allocating link capacity in arbitrary networks, and explained why this leads to *max-min fairness*. The fair rates of the flows are not explicitly computed –the distributed scheduling algorithm just finds them "by itself". Hahne [9] subsequently proved that this scheme indeed yields max-min fairness, although, in some pathological cases, very large buffers may be needed for that. This paper differs from the above 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; and *(c)* we study and simulate the transient behavior, specifically in crossbars, when flows or weights change.

[1] Sub-port demultiplexing, if needed, would still require output queues, unless the crossbar is modified to provide this capability, e.g. by partitioning each crosspoint buffer into per-subport queues, and making the output schedulers operate at subport granularity

In the late nineties, Stephens and Zhang [10] [11] studied and simulated buffered crossbars with WFQ/WRR schedulers, and proved their ability to provide delay bounds to properly policed flows. Chiussi and Francini [12] studied a similar distributed WFQ architecture in multistage networks with backpressure, and obtained analogous QoS guarantees. These delay bounds are based on the minimum rate guaranteed for each flow, which is the minimum of the following quantity over all links traversed by the flow: the ratio of the flow's weight over the sum of the weights of all flows traversing the link. Thus, these papers do not consider the allocation of the excess bandwidth that results when some flows are not able to use as much bandwidth as their weight indicates. The present paper differs from the above in precisely this point: we study the allocation of excess bandwidth, both during transient periods and in the long term. Recently, Javidi e.a. [13] examined buffered crossbars with longest-queue-first input schedulers and round-robin output schedulers, and showed full output utilization under some assumptions. Weighted max-min fairness (examined in this paper) also reaches full output utilization in the cases where the flows are not constrained at the inputs. By studying how close our system comes to weighted max-min fairness, we also study how close it comes to full utilization of oversubscribed outputs; however, we use a more general scheduling policy than [13], and we study fairness issues in addition to what that paper considers.

1.4. Contributions of this Paper

This paper studies the fairness properties of distributed scheduling in buffered crossbars. First, we examine bandwidth allocation assuming WFQ/WRR schedulers and persistent flows. Using a simple fluid model, section 3 shows that the system serves the flows according to *weighted max-min (WMM)* fairness; this is an extension of the arguments provided in [8] for the case of RR schedulers. We simulated the real system (fixed-size cells – not a fluid model) for various sizes of crosspoint buffers, various weight factor combinations, and various crossbar sizes (section 4.2). In each case, we measured the discrepancy between the actual flow rates and the rates predicted by WMM fairness; we find that these discrepancies are reduced to a few percents of the WMM fair rate with buffer sizes as small as a few cells per crosspoint. We also discuss which weight factor combinations yield nearly perfect results, and which combinations lead to larger discrepancies.

Second, we study the transient phenomena that occur when the weight factor of a flow changes, or when the active/inactive state of the flow changes. In section 3.3, we reason about the process that causes the system to restabilize at the new WMM fairness equilibrium, the factors that affect the delays in this process, and the chains of dependencies along which stabilization progresses. Section 4.3 presents our simulations of this transient behavior, which verifies our previous reasoning. The stabilization delay depends on the buffer size and the rate difference (new rate minus old rate); smaller buffers and larger rate changes incur faster stabilization. We did not notice any oscillations during the stabilization process along these chains.

To the best of our knowledge, both of the above contributions appear for the first time in the area of packet switching, as discussed in section 1.3 above, on related work. Our assumptions are summarized as follows. We assume buffered crossbars, motivated by modern IC technology and the resulting evolution of real crossbar switches that we foresee and support. Second, we assume WFQ/WRR schedulers, which are technologically feasible (see section 2.2) and yield advanced QoS architectures. Finally, 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. The transient phenomena that we study in this paper occur when the state of a queue changes between empty and non-empty.

2 . BACKGROUND & DEFINITIONS

This section reviews WMM fairness, and defines our crossbar and distributed WFQ/WRR scheduling models.

2.1. The Weighted Max-Min (WMM) Fairness Objective

Max-min fairness allocates as much bandwidth as possible to each flow, provided that this bandwidth is not "taken away" from a "poorer" flow. In other words, given

a max-min fair allocation, it is impossible to increase the bandwidth of any flow A without reducing the bandwidth of a flow B where B's allocation was inferior or equal to A's allocation[2]. Weighted max-min (WMM) fairness allocates *utility* in a max-min fair way, where utility of a flow is its bandwidth allocation divided by its weight factor [14]. Equivalently, if each flow with weight w is an aggregation of w microflows, then WMM fairness among flows is the same as plain max-min fairness among microflows.

A constructive algorithm for finding the WMM allocation can be deduced from the above definition. Let us call *weight* of a link the sum of the weights of the flows that traverse it, and *fair share* of a flow on a link the weight of the flow divided by the weight of the link. First, find the most congested link, i.e. the link with the largest weight. The allocation for the flows traversing the most congested link is precisely their fair share on that specific link. On the rest of the links, these flows cannot use all of their (local) fair share, because they are constrained to the (lower) allocation dictated by the most congested link; the portion of bandwidth that they leave unused is distributed to the rest of the flows. Consider the network that results from the original one when we remove the above most congested link and all the flows that traverse it. For this new network, compute link weights again, find the most congested link, allocate its (remaining) bandwidth to the (remaining) flows that traverse it, and so on.

2.2. System Model: Distributed WFQ Crossbar

Figure 1 shows the model of the $N \times N$ switch system that this paper deals with. There are *virtual output queues (VOQ)* at the N inputs, containing fixed-size cells. The core of the systems is an $N \times N$ crossbar; there is *no internal speedup* and there are *no output queues*. The crossbar contains N^2 small queues, one per crosspoint. A full system could contain separate queues per priority level; we analyze the behavior within one of the priority levels in this paper, while extension to multiple levels is rather straightforward. Backpressure flow-control ensures that the crosspoint buffers will never overflow.

There is a scheduler s per crossbar input, and a scheduler s per crossbar output. Each input scheduler chooses among its *eligible* VOQ's; a VOQ is eligible iff it is non-empty and its corresponding crosspoint buffer is non-full (backpressure is in the "go" state). Each output scheduler chooses among its *eligible* crosspoint buffers (a column in fig. 1); a crosspoint buffer is eligible iff it is non-empty. Output schedulers feed egress links directly, since there is no speedup and no output queues.

[2]Note that max-min fairness is different from maximum utilization; e.g., in a 2×2 crossbar with three active flows, $\lambda_{0,0}$, $\lambda_{0,1}$, and $\lambda_{1,1}$, max-min fairness is $\lambda_{0,0} = \lambda_{0,1} = \lambda_{1,1} = 0.5$, yielding aggregate throughput of 1.5, while maximum utilization is $\lambda_{0,0} = \lambda_{1,1} = 1.0$ and $\lambda_{0,1} = 0$, yielding aggregate throughput of 2.0.

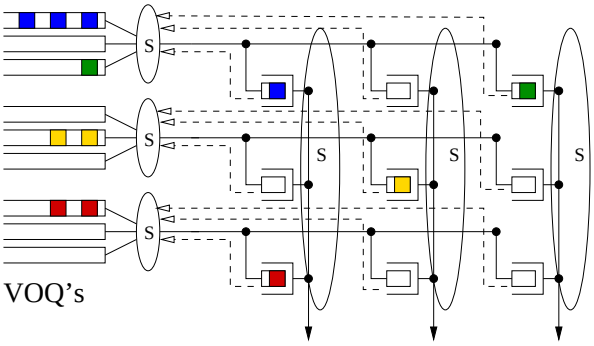

Fig. 1. System model assumed in this paper.

All schedulers use a common scheduling discipline, which is a WFQ/WRR variant. Flows are determined by input-output pairs, and correspond one-to-one to crosspoint buffers. Each flow f has a unique weight factor; schedulers use the inverses of the weight factors, called *service_interval*, SI_f, measured in arbitrary units (the same for all flows, though). Each scheduler maintains a *next_service_time* state variable, NST_f, for each of its flows; the NST of a flow at its input scheduler is independent from and unrelated to the NST of the same flow at the output scheduler (there is no attempt to enforce any single "system potential"). At each timeslot, t, a scheduler selects and serves an *eligible* flow g that has the minimum NST_g^t, and updates this NST_g^t to $NST_g^{t+1} = NST_g^t + SI_g$. Also, all *ineligible* flows h for which $NST_h^t < NST_g^t$ are *dragged* to the current scheduler "time" (potential): $NST_h^{t+1} = NST_g^t$; in this way, ineligible flows will not receive a burst of service when they become eligible again. This scheduling discipline can be implemented at high speed, e.g. using a tree of comparators exploiting bit-level parallelism [15]: the minimum of 256 twenty-four-bit numbers can be found in 4.5 ns in 0.18-micron CMOS technology.

3 . Convergence to WMM Fairness and Transient Behavior

This section shows why distributed WRR crossbar scheduling converges to WMM fairness, discusses how the convergence process works, and indicates which factors affect its delay. 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 a same output in the same time-slot, the crosspoint buffers will hold these cells until the output scheduler reads them out 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 service at the input until it gets equalized to the rate dictated by the output scheduler. In this way, over the long run, the service allocated to each flow becomes the smaller of the two rates that its input and its output can allocate to it. Because schedulers are work conserving, they will always serve a flow as long as there is at least one eligible flow; thus, bandwidth that remains unused by ineligible flows gets distributed to the remaining eligible flows, according to these latter flows' fair share. Eventually, this redistribution will yield WMM fair allocations, as discussed below in more detail.

3.1. Unbuffered Fluid Model

The behavior of our system and its analysis are simplified when we replace discrete cells with an infinitely divisible fluid, and WRR schedulers with ideal *generalized processor sharing (GPS)* servers [16].

Theorem *1: Under the fluid model, GPS servers, no buffers at the crosspoints, and persistent sources, all flows will receive exactly their weighted max-min fair rate allocation.*

Proof: To prove this theorem, note that the constraints of the system are such that it has to operate according to the constructive algorithm of section 2.1 for the WMM fair allocation. Consider the most congested link (link with highest sum of flow weights). We argue that all flows on that link are always eligible. The proof will be by contradiction: assume that flow f on that link occasionally becomes ineligible. If this is an input link, given that the source of f is persistent, the only reason for f to occasionally become ineligible would be for f's output server to serve f at a rate lower than the input server rate. This is a contradiction, because the input link of f is the most congested link in the system, hence the input fair share of f is lower than its output fair share, and a GPS server (the output server in this case) will never serve a flow below its fair share "on its own initiative" –this could only occur if the flow were constrained to a lower rate at some other resource, which is not the case here. Similarly, if the most congested link is an output link, the only reason for f to occasionally become ineligible there would be for f's input server to serve f at a rate lower than the output server rate, which is a contradiction for the symmetric reason.

We just proved that all flows on the most congested link are always eligible, hence they receive precisely their fair share of service. For each of these flows, f, consider the other link that the flow goes through; now that we know the rate of f, we can deduce the maximum possible rate of all remaining flows on that link. Consider all other links in the system and all remaining flows on them, and consider the most congested link among them. The arguments go as above, until all system links have been exhausted.

3.2. Discrepancies in a Non-Fluid System

The real system with discrete cells obviously differs from the above ideal fluid model. The WFQ/WRR scheduler assumed in section 2.2 will allocate rates precisely 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 a normally-full buffer never empties and a normally-empty buffer never fills up. Normally-full buffers are the buffers of flows for which, under a stable state, the service at the input is higher than the service at the output, and conversely for normally-empty buffers. If the buffers are not large enough, it may happen that a normally-full buffer occasionally empties: although its input scheduler is supposed to fill it more frequently than its output scheduler empties it, actual service is not perfectly smooth. Such service fluctuation may cause the input scheduler to occasionally be late in refilling the buffer, and the output scheduler to occasionally be early in emptying the buffer. Section 4.2 presents our simulation results concerning the magnitude of these discrepancies.

3.3. Buffered Fluid Model and Dependency Chains

We now turn our attention to the transient phenomena when a flow enters or leaves the system, or equivalently when the weight factor of a flow changes (inactive flows are equivalent to zero weight factors). To study these phenomena in analytical terms we have to resort again to the

fluid model simplification. Unlike section 3.1, we obviously have to assume non-zero crosspoint buffers. In the model that we analyze here, there is a GPS server at each input and output of the crossbar, and a buffer is placed at each crosspoint, dedicated to one of the N^2 fluid flows, as shown in fig. 1 .

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. After a change in the system, the restabilization process corresponds to a chronological sequence of crosspoint buffers changing state. As each buffer in the sequence reaches its next (empty or full) state, the rate of the corresponding flow gets reduced either at the input or at the output. In turn, this increases the rates allocated by the corresponding GPS server to some of the other flows sharing the same port. We wish to study the *dependency chains* that may exist among flows, along which rate changes propagate until the system finds its new equilibrium. We will see that these chains are related to the sequence in which the constructive algorithm for the WMM fair allocation (section 2.1) computes the flow rates.

Theorem *2: Assume that a change in flow f causes the system to move from WMM fair allocation A to WMM fair allocation B. Then, any flow g whose rate under B differs from its rate under A is a flow for which either $U_g^A \geq U_f^A$, or $U_g^B \geq U_f^B$, or both (where U is the utility of a flow under an allocation, i.e. the rate of the flow divided by its weight).*

Proof:
The proof is by contradiction. Suppose that g had strictly smaller utility than f under both A and B allocations, and additionally g was affected by the change in f –that is the final rate of g (under B) is different from the starting rate of g (under A). Let V^A and V^B be the utility vectors of allocations A and B respectively, each of them ordered according to the sequence in which the constructive algorithm of section 2.1 determines the allocations –hence, each of them ordered by non-decreasing utility. If g reacted to f's change and $U_g^A < U_f^A$, $U_g^B < U_f^B$ then V^A and V^B will have the form:

$$V^A = \{U_{i_1}^A, U_{i_2}^A, \ldots, U_g^A, \ldots, U_f^A, \ldots, U_{i_N}^A\}$$
$$V^B = \{U_{i_1}^B, U_{i_2}^B, \ldots, U_g^B, \ldots, U_f^B, \ldots, U_{i_N}^B\}$$

Since the algorithm for computing the equilibrium fills this vectors from left to right (in groups of equal utilities), it means that when the WMM fairness algorithm computed the rate of flow g, this algorithm did not take into account f in both cases, so g cannot have been affected ◁.

GPS is a scheduling discipline that achieves WMM fair allocations [17, section 9.4.1]. Thus, a change in one flow f, served by a GPS at some input or output of the fluid buffered crossbar model, cannot affect a flow g, served by the same GPS server, that is "more congested" than flow f both before and after the change. So using theorem 2, we can tell which flows g are potentially affected by a change in flow f. We will use the following definitions:

- *Active flow:* a flow with non-empty input queue, or equivalently non-zero input rate, or equivalently non-zero weight factor.
- *Neighbor flows:* two active flows that share an input or an output port.
- *Interacting flows:* two neighbor flows, or two flows that both interact with a common third flow (i.e. the transitive closure of the neighborhood relations, considering only active flows).
- *Dependents of a flow f* in a WMM fair allocation A: all other flows whose rate may potentially be affected by a change in f's demanded rate or weight.

Theorem *3: In a stable state, A, of the fluid buffered crossbar model the dependents of a flow f are all included in the following set of flows: (a) f and all its neighbors; and (b) all flows that interact with the flows in (a) and that can be placed on a chain of flows, where the chain starts from a flow in (a) and contains flows in order of non-decreasing utility (under state A).*

The proof of this theorem can be found in of [18, section 4.4]. The theorem tells us that, first of all, all neighbors of f can get affected, regardless of the level of utility that they receive, e.g. when f changes from active to inactive or vice versa, or when the external input rate of f changes, or if f's weight changes. Next, each of these neighbors, including f, may propagate the change to other flows g, but only along paths of interactions composed of less and less congested flows. The increasing utility property ensures that the change can potentially propagate through a GPS server to neighbor flows.

3.4. Delay and Unfairness During Transient States

So far we have only described the (re)convergence process in terms of utility and interaction chains. We now turn to the delay of this convergence, and the magnitude of service unfairness during the process.

When a flow f changes state or demand, multiple dependency paths may be stimulated in parallel. When two such paths reconverge at some input or some output, their effects may be additive or subtractive; also, a flow f may experience increased congestion at its input and decreased congestion at its output. This sequence of events can produce a rate increase and afterwards a rate decrease (or

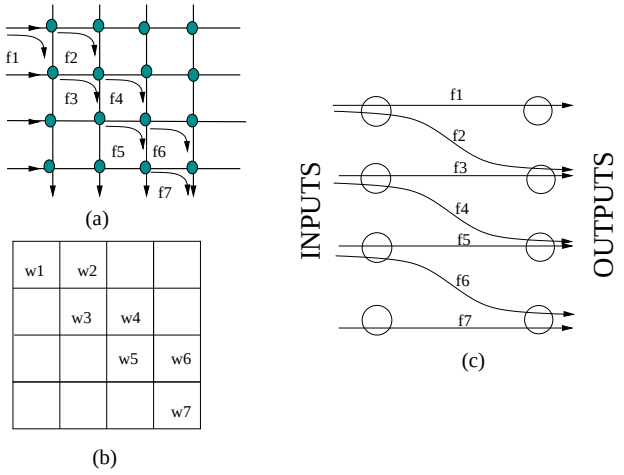

Fig. 2. (a) Seven interacting flows (b) their weight matrix, and (c) the bipartite graph of resources(nodes) and requests(edges).

vice-versa) before f's rate stabilizes, e.g. first a congestion decrease reaches f's input and latter a congestion increase reaches f's output. In other cases, the multiple dependency paths produce changes of the same sign and accelarate convergence. Although dependency paths reconverging upon a flow may have subtractive results, the latest one of them will determine the final rate, hence we turn our attention to how long one, single dependency chain can be. We believe that depedence chains can never be circular, because they are always formed along paths of decreasing utility; also, in our simulations we have never seen circular dependencies. We are currently in the process of verifying that this statement is indeed always true.

A rate increase propagates instantly an increase in demand, from the flow's input server to the flow's output server or vice versa, regardless of the state of the buffer. On the other hand, a rate reduction may incur a delay in propagating the change. For example, a reduction in input rate of a flow whose buffer was full will only propagate to the output after the buffer gets emptied; if, however, the buffer was already empty, the change will propagate instantly. Conversely, a reduction in output rate will incur a delay if the buffer was empty and can now be filled, while it will propagate instantly if the buffer already full.

The delay parameters are: the crosspoint *buffer size*, since these may need to get filled or emptied before a rate change propagates; the *magnitude of the rate change*, which is usually equal to the input-output rate difference that determines how fast the buffer gets filled or emptied; and the *dependency chain length*, which gives an indication of the number of buffers that must get filled or emptied.

The longest dependency chain in the crossbar has

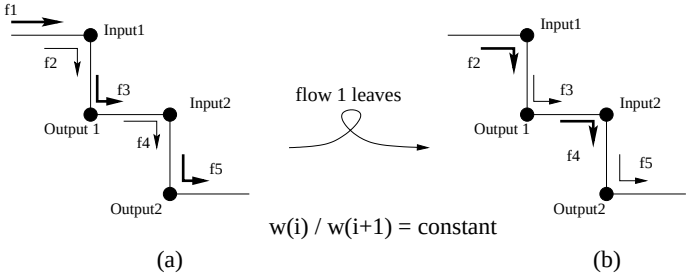

Fig. 3. Dependence chain of the first few flows in fig. 2 .

length $2N$, because it visits all inputs and outputs precisely once –if some port occurred twice, we would have to reconvergent dependency chains (see the discussion above). Thus, there can be at most N buffers that must change state before all flows restabilize correctly.

Figures 2 and 3 show an example where all buffers along the chain have to switch states. To create this example we used decreasing weights that satisfy the relation $\forall i : \frac{w_i}{w_{i+1}} = c$. Consider that $\frac{w_i}{w_{i+1}} = 2$ and that initially flow 1 is active, as in fig. 3 . In the starting equilibrium state, flow 2 will be bottlenecked at the input, and will receive a rate of 0.33. Flow 3 will be taking advantage of f_2's inability, and will use up the remaining rate of 0.67; this flow can receive that excess bandwidth, since it corresponds to its fair rate at the input ($\frac{w_3}{w_2} = 2$). Now, when f_1 becomes inactive, f_2 will benefit from the absence of f_1 and will increase its rate to 0.67. This event will be instantly propagated to f_3, which will drop its output rate to 0.33; this change will be propagated to f_4 after the crosspoint buffer of f_3 is drained. At that point, f_4 will instantly raise its rate to 0.67, and so on.

The above scenario of serial changes represents the worst-case with respect to delay and unfairness rate. We can estimate the delay of stabilization along such a dependency chain, for a node at distance $2D$ from the originating flow, as:

$$Delay \ \leq D \cdot \frac{B}{min(oldRate_f - newRate_f)} \quad (1)$$

The unfairness rate during the transient states is proportional to the $min(oldRate - newRate)$, so we can derive an unfairness bound measured in bits that does not depend on the old and new rates:

$$UnFairBits \ \leq D \cdot B \quad (2)$$

Section 4.3 presents our simulation results, which verify the general form of the above relations.

Worst case scenarios like the above have a very specific form, and are thus quite improbable to occur in practice. In practical situations, weight factors may look like random numbers. Then, it is quite likely that dependency

chains have a small average length, and that multiple reconvergent chains exist; these cannot delay the rate stabilization compared to the slower dependency path, whereas in many other times the reconvergent paths produce additive effects, which speed-up convergence.

Another effect to take into consideration, in practical situations, is that several flows may be inactive (their VOQ at the input is empty). Inactive flows do *not* help in building *multiple* reconvergent chains, but they do *not* help in building *long* dependency chains, either. The net effect is that accelerated convergence is also expected in the cases where many inactive flows exist.

4 . Simulation Results

4.1. Simulation Environment

We implemented a simulator in C++ to evaluate the performance of the distributed WFQ buffered crossbar. The simulator assumes fixed-size cell traffic, and operates at cell-time granularity. The input and output schedulers operate according to the algorithm described in section 2.2. Active flows are fed by persistent (always non-empty) VOQ sources, while inactive flows receive no incoming traffic (empty VOQ sources); as discussed in section 1.4, persistent flows model the short-term behavior of the switch under transient overload, for as long as this overload lasts, and provide a fixed set of demands to the switch, under which the system can stabilize to its equilibrium state. For the transient behavior measurements, we made one of the flows change between active and inactive state at a precisely known instant in time. The simulator uses unit-delay on/off backpressure: assume that an input server decides in time-slot t_i to server a flow f; this decision becomes known to f's output server in time-slot t_{i+1}; the cell is actually transmitted in time-slot t_{i+1} while in the same time-slot t_{i+1} the output server may decide to serve this same cell in time-slot t_{i+2}. If an output decides to read a cell from a full crosspoint buffer during time-slot t_j, this buffer becomes eligible for its input server in time-slot t_{j+1}. Under these assumptions, two cells worth of buffer space per crosspoint suffices for a flow experiencing no competition at the input and at the output to be served at full line rate.

4.2. Accuracy of Convergence to WMM Fairness

As discussed in section 3, the ideal fluid model converges to WMM fairness, while for a real, discrete cell system a number of inaccuracy sources exist. We study by simulation the magnitude of these inaccuracies. The metric that we use is the *Relative Error*, RE, with respect to the ideal WMM fair rate allocation. Given a simulation interval, for each flow f the relative error of f's rate is defined as:

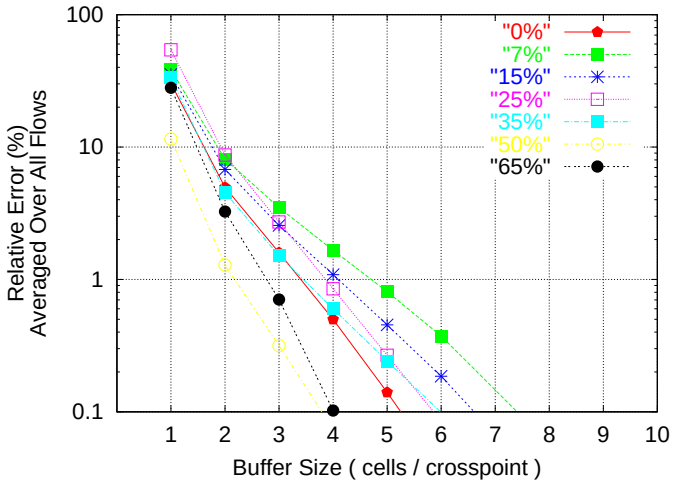

Fig. 4. Average RE versus buffer size, for various percentages of inactive flows; under uniform weights, 32×32 switch.

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

where $ActualService_f$ is the number of cells of f that exited from the simulated switch during the simulation interval, and $FairService_f$ is the rate allocation of 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. We report the *average* and the *maximum* of the RE_f's over all active flows f. For the reports below, the measurement interval started safely after any initial transients of the system (many tens of thousands of cell times after the beginning of simulation), and extended as long as needed to reach a confidence interval of 0.04% with confidence 95%[3].

1) Effect of Weight Factors and Flow Activity: We used three different types of weight factor distributions to flows. In the configuration called *uniform*, all active flows have a random service interval variable ($SI_f = \frac{1}{w_f}$), picked uniformly in the interval $[1, 1001]$. In the *skewed-j^2* distributions, SI_f for the flow from input i to output j is randomly chosen through the following random process $[1 + 10j^2 + unif\_rand\,(0, 10j^2)]$. Finally in the distribution named *mixed*, each flow from input i to output j has $SI_f = 8$ or 6 with equal probability, if $i + j$ is odd and $SI_f = 4$ or 3 with equal probability, if $i + j$ is

[3]The confidence is extracted as follows: every 10,000 cell times, we compute the average and maximum RE over all active flows. Then we extract a new average-estimate of the average and maximum RE values seen so far, and we stop when the *true average* of the average and maximum RE lies in an interval of width 0.04% around the current average-estimate with probability > 0.95.

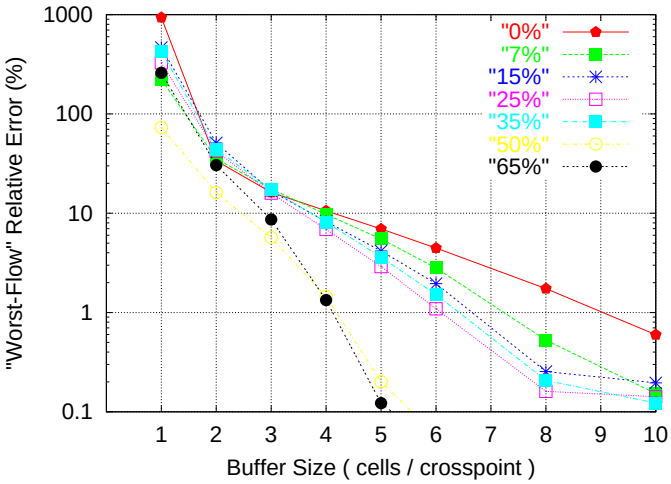

Fig. 5. Worst-case RE over all flows; other parameters as in fig. 4 .

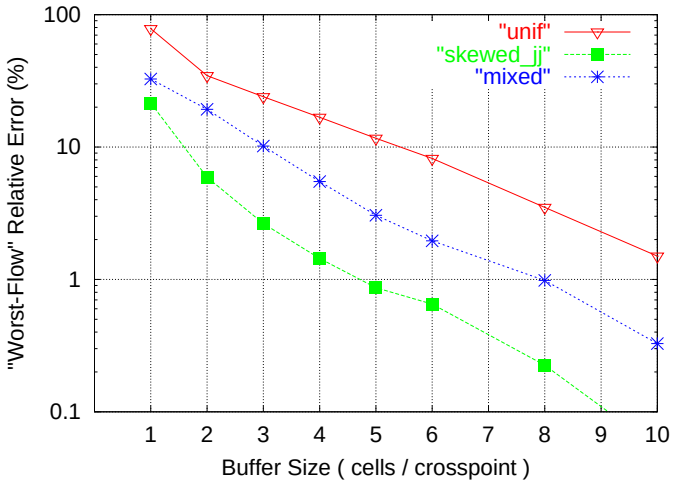

Fig. 7. Worst-case RE over all flows, under various weight-factor distributions; other parameters as in fig. 6 .

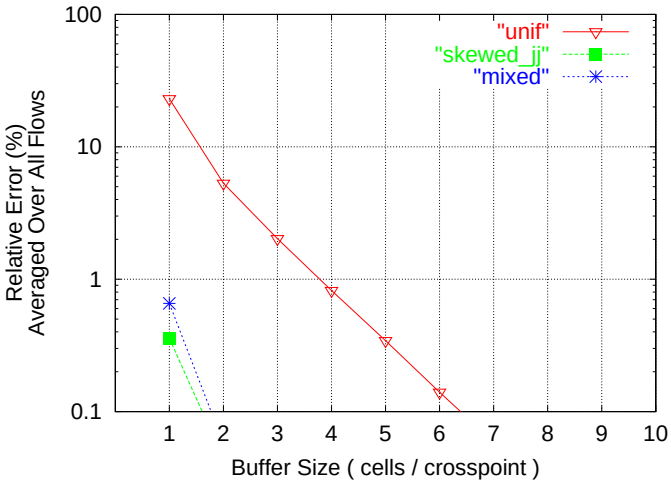

Fig. 6. Average RE under various weight-factor distributions; all flows are active; 32×32 switch .

even. The skewed distribution was used in an effort to create imbalanced weight factors: the small-index outputs of the switch are in much higher demand (smaller SI hence larger weight) than the large-index outputs. The mixed distribution was used in order to create multiple dependencies among flows, since neighbor flows are likely to have weights similar to the scenario presented in fig. 3 . In all configurations, we randomly decide with probability l if a flow will be inactive.

Figure 4 plots the average RE of the active flows, with uniformly chosen weight factors, under four different inactivity probabilities, l = 0%, 7%, 15%, 25%, 35%, 50%, 65%. We see that a buffer size of 5 cells per crosspoint suffices to drive the average error below 1 %.

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. Given that the fair rates for such outputs add up to 1.0, and given that the actual rates are within 1 % of the fair rates, it follows that utilization is 99 % or better for these outputs.

Figure 5 plots the maximum value of RE over all active flows, for the same simulations as in fig. 4. We see that buffer sizes of 6 to 8 cells each –depending on the ratio of inactive flows– yield worst-case errors of 5 or less percent.

There is a general tendency to better approximate the fair allocations when there are more inactive flows. This is well pronounced for the maximum error, and less clear for average error. This tendency can probably be attributed to the smaller number of flows that each WFQ/WRR scheduler has to consider when there are more inactive flows; fewer flows in the schedule means less jitter in their service time, so less opportunities for a normally-full buffer to empty or a normally-empty buffer to fill up. Another reason may be the shorter dependency chains when more inactive flows exist.

Figures 6 and 7 plot the relative error (average or worst-flow) for the various weight factor distributions discussed above. Although we created the skewed distribution with the intention to drive the system into difficult operation, it turns out that relative error gets smaller under the skewed distributions, the difference being more pronounced in error averages over all flows. Under most runs that we have tried, the uniform distribution produces the worst convergence conditions for the buffered crossbar.

2) Effect of Switch Size: Figures 8 and 9 plot the relative error (average or worst-flow) for various switch sizes: 32×32, 64×64, and 128×128; weight factors are drawn uniformly, and 25 % of the flows are inactive. We observe that average RE becomes better (smaller) for larger switches; this may possibly be due to a larger number of

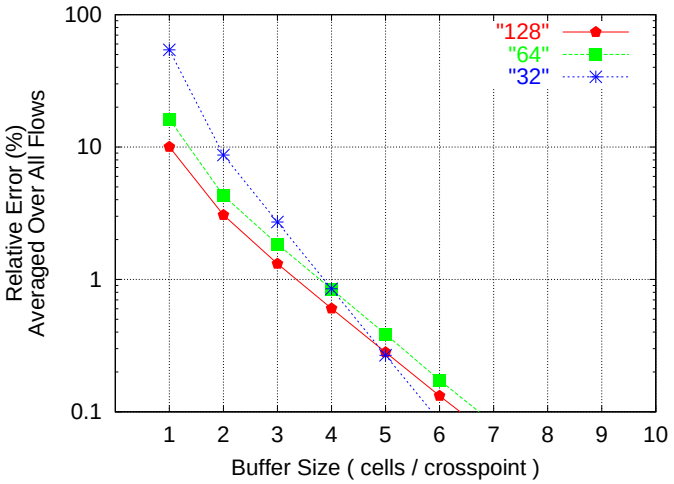

Fig. 8. Average RE for various switch sizes; uniform weight-factor distribution, 25% inactive flows .

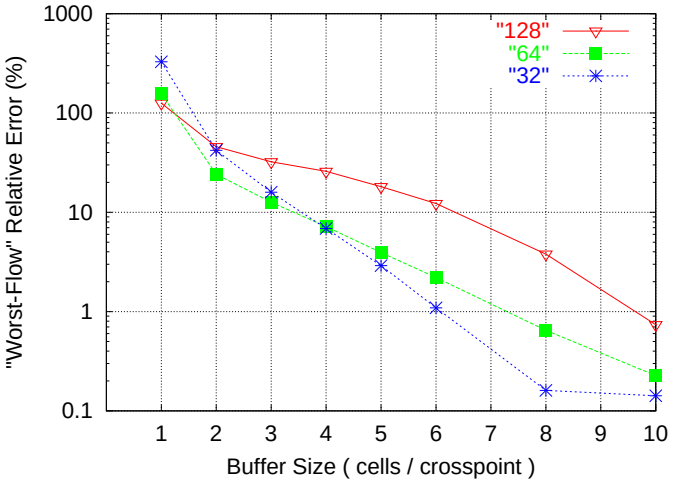

Fig. 9. Worst-case RE over all flows, for various switch sizes; other parameters as in fig. 8 .

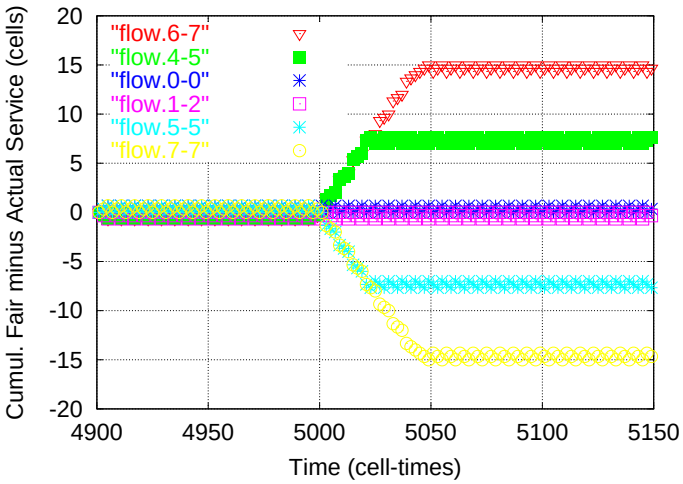

Fig. 10. Service difference versus time; 4-cell buffers per crosspoint, 8×8 switch, fig. 2 flows, weight ratio = 2. The configuration shown in fig. 2 ($\frac{w_i}{w_{i+1}} = 2$) .

flows receiving accurate service, which brings the average down. On the contrary, larger switches yield worse (larger) RE for the worst flow. 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 may possibly be attributed to larger jitter in the WFQ/WRR schedulers (due to more flows in each scheduler) and to longer dependency chains.

Concluding, the system stabilizes very close to the WMM fair shares, under all demand distributions that we have tested, with small amount of buffering per crosspoint (3 to 10 cells).

4.3. Transient Delays and Unfairness Magnitude

We simulated the system's transient behavior, when a flow's activity or weight changes, in order to verify our results of section 3.4. We present the results of the simulation of a dependency chain like the one shown in figure 2. We initially configured all flows across the chain to be active (persistent VOQ's), we run the system until it stabilizes, and then we changed one flow to inactive (empty VOQ), examining which other flows are affected, and their restabilization delay. For the flows of interest, f, we plot the difference of the cumulative service (number of cells) that f received during the simulation, as a function of time, from the cumulative service that ideal WMM fairness would allocate to f; for computing the latter (a piecewise linear function) the WMM fair rate is assumed to change instantly at the moment when the original affecting flow changes to inactive. When the above metric for flow f becomes parallel to the time axis, it means that f's rate has stabilized to its new WMM fair rate.

First we verified that flows with lower utility than the changing flow remain unaffected by the change, like they should (Theorem 3). Figure 10 shows what happens when flow f_{3-3} turns from active to inactive on cell time 5000. We see that only flows receiving greater utility than f_{3-3} are affected: $f_{6-7}, f_{7-7}, f_{4-5}, f_{5-5}$). Flows f_{0-0} and f_{1-2} are not affected, since these received lower utility than f_{3-3} in the allocation before the event.

Then we verified the impact of the distance in the dependence chain. In fig. 11 we make f_{0-0} inactive, as in fig. 3 , at cell time 5000. All input neighbor flows ($f_{i\to j}$, $f_{i\to j+1}$) must switch rates from 0.67 to 0.33 and vice versa. For flows f_{3-4} and f_{4-4} to stabilize 3 buffers must switch state; for flows f_{6-7} and f_{7-7} to reach their new rate, 6 buffer changes are needed. The rate at which buffers switch state here is 0.33 (= old_rate- new_rate =

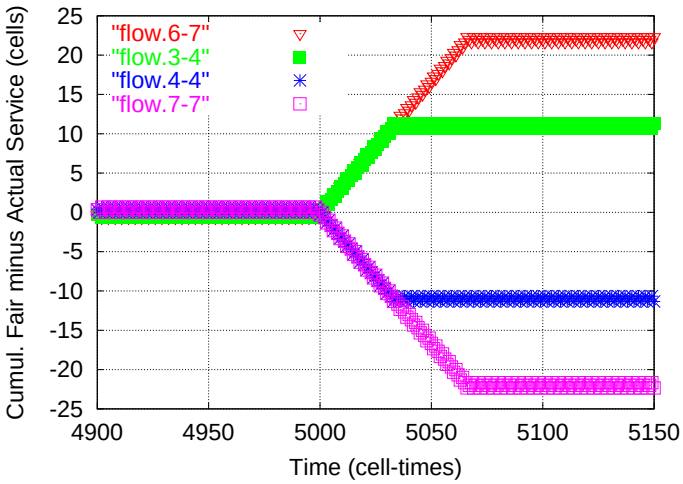

Fig. 11. Service difference versus time for flows at increasing distance from the affecting flow; parameters as in fig. 10 .

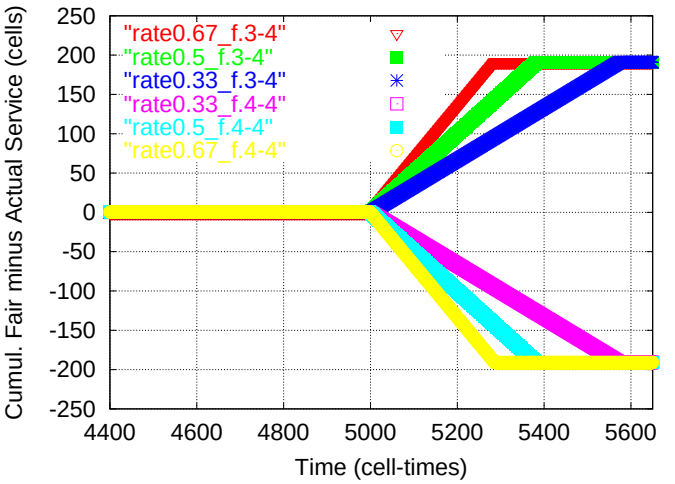

Fig. 13. Service difference versus time for varying sizes of rate change; 64-cell buffer space/crosspoint.

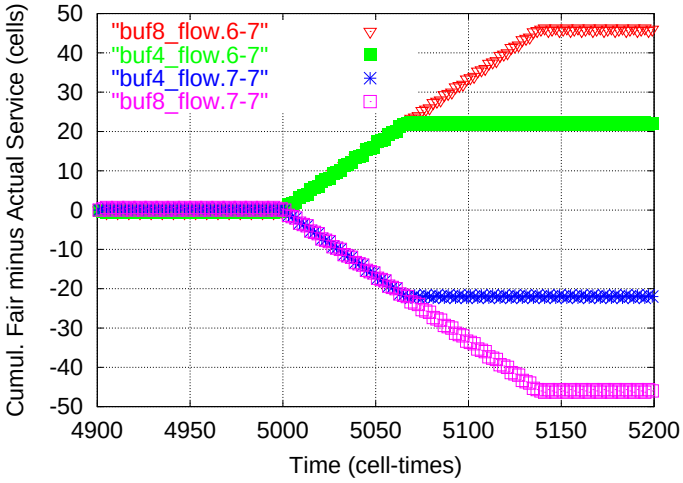

Fig. 12. Service difference for the same change in two different crossbars, one with 4-cell buffers per crosspoint and one with 8-cell buffers; other parameters as in fig. 10 .

$0.67 - 0.33$), so the estimation for the stabilization delay is 3 buffers $\times$ 4 cells/buffer $\times$ 0.33 cells/cell-time = 36 cell times for the first couple, and similarly 72 cell times for the second; we see that fig. 11 verifies that. The unfairness that accumulates during the transient phase, measured in number of cells, is again proportional to the distance along the dependence chain, and is almost doubled as seen in fig. 11 for the second pair of flows, which is at twice the distance relative to the first pair.

Next, we examined the effect of the crosspoint buffer size. Based on the same reasoning as above, we expect the delay and the unfairness magnitude to double, for a given flow, when we go from a crossbar with 4-cell buffers to a switch with 8-cell buffers per crosspoint. Figure 12 shows that this is indeed the case.

The final parameter that affects the delay of the stabilization at the new state is the magnitude of the change. Here we examined how the magnitude of rate change affects the delay and the cumulative service unfairness. In the configuration of fig. 2 , we modify the weights ratio $\frac{w_i}{w_{i+1}} = c$, which affects the flow rates, before and after the change. When this weights ratio is 5, flow rates switch between 1/6 and 5/6, hence the magnitude of the rate change is 0.67; with weights ratio of 3 the rate change is 0.5, and with a ratio of 2 the "speed" of change is 0.33 cells per cell-time. Figure 13 shows the effect on flows f_{3-4} and f_{4-4} when f_{0-0} turns inactive at cell time 5000, for the above three different values of the weights ratio. We see that indeed, the affected flows stabilize faster to their new rate when the weights ratio is 5 (rate change of 0.67), while ratio 2 yields the slowest stabilization (rate change = 0.33, and about twice the delay compared to rate change of 0.67). On the other hand, the amount of cumulative unfairness, measured in number of cells, remains the same in all three cases, confirming our corresponding conclusion in section 3.4. In fig. 13 the cumulative unfairness is just below 200 cells, when the theoretically predicted number is 192: 3 buffers on the dependence chain have to change state, times 64 cells per buffer, equals 192 cells of cumulative unfairness.

Finally, we examined two other scenarios that are related to dependency chains. The first is the equivalence of an active to inactive change (or vice versa), with a change in weight. This equivalence is better understood when the weight is simply the number of active sub-flows that constitute an aggregation; a weight increase is equivalent to a number of sub-flows becoming active, whereas a weight decrease is equivalent to a number of sub-flows becoming inactive. In fig. 14, flow f_{4-0} increases its weight from 1 to 256 and as a result all flow rates change, despite the

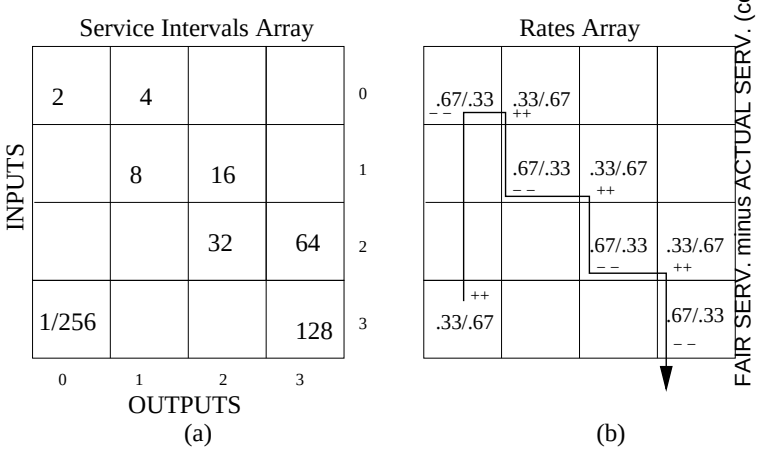

Fig. 14. (a) The service intervals (=1/weight) table of the flows before and after the change (old/new). (b) The stabilized rates before and after the change (old/new).

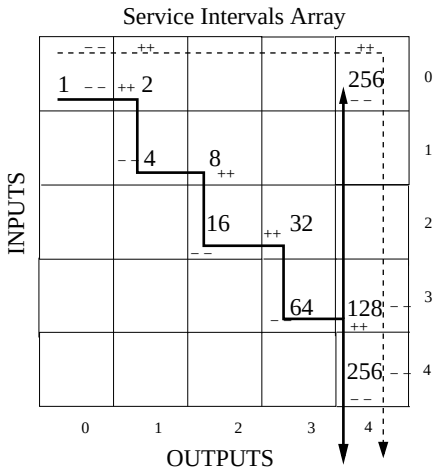

Fig. 15. Presentation of the two reconvergent dependency chains on the service intervals ($SI_f = \frac{1}{w_f}$) table of a $5x5$ crossbar.

fact that f_{4-0} was receiving the highest utility before the change. Since flow f_{4-0} can affect its neighbor flow f_{0-0}, it turns out that this flow can affect all other flows in the dependency chain that starts from f_{0-0}. This is what theorem 3 captures by including neighbor flows independent of previous utility, in the characterization of the dependency paths[4].

The second scenario is the reconvergence of multiple dependency chains, that normally occur with common/random weights configurations. The slowest of these

[4]We decided not to show the actual plot of the flows, but only the final states produced, due to space limitations.

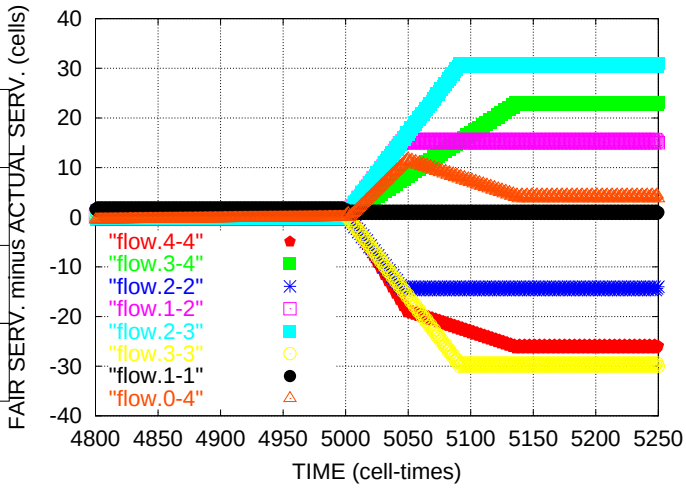

Fig. 16. Service difference versus time for the configuration presented in fig. 15 ; 16 cells worth of buffer space/crosspoint.

paths will determine the stabilization time.

In fig. 15 , there are two dependency paths that start from flow f_{0-0} and reach flows f_{3-4} and f_{4-4}; one path goes through flow f_{0-4} and the other path goes along the main diagonal of the crossbar table. When flow f_{0-0} becomes inactive at time-slot 5000, flow f_{0-1} takes almost all remaining capacity from input 0 ($\simeq$ 0.99 cells/time-slot), whereas its output service rate is confined to 0.67 cells/time-slot. When its crosspoint buffers fill (at time $\simeq$ 5050), this flow's input rate adapts to its output fair share (0.67 cells/time-slot) and flow's f_{0-4} input service rate increases –increasing demand and service rate at output 4–, thus potentially affecting (decreasing) flows f_{3-4} and f_{4-4}. Flow f_{3-4} is not affected since at this time this is constrained in the input and the potential service decrease at output 4 does not exceed its demand stemming from input 3, whereas flow's f_{4-4} service rate decreases. This is the time (time-slot $\simeq$ 5050) that the first dependency path affects flows f_{3-4} and flow f_{4-4}, but since the second dependency chain has still not reached them, the knee-like effect is produced for flows f_{0-4} and f_{4-4} at time $\simeq$ 5050 (fig. 16). Later, when the second and slower dependency chain reaches output 4 at time $\simeq$ 5130 (flow f_{3-4} stops being constained at the input, since f_{3-3} reduced its demand at input 3 to 0.33 cells/time-slot), all flows, including f_{0-4}, stabilize correctly at their fair shares. Note that the stabilization of flow f_{0-4} at this time cannot affect flow f_{0-1} (neighbors at input 0), since this flow receives less utility at input 0[5].

5 . Conclusions

Current IC technology allows the integration of small crosspoint buffers into crossbar chips. The resulting

[5]If it did, the system could oscillate along circular dependencies.

buffered crossbar architecture offers significant advantages: *(a)* variable size packets can be switched; *(b)* no internal speedup is needed; *(c)* output buffer memories can often be eliminated; *(d)* scheduling gets simplified because it becomes distributed, and can easily yield full output utilization; and *(e)* WRR/WFQ scheduling becomes feasible, thus enabling sophisticated QoS architectures.

In view of the expected rising commercial importance of buffered crossbars, we studied the *fairness* properties of distributed WFQ/WRR scheduling in such switches. These properties become important for QoS every time a crossbar output gets oversubscribed. Even if higher-level flow control ensures the absence of output congestion in the long run, short-term output overloading will often occur due to traffic variability. During such periods, which can last for fractions of a second in modern networks, it is important to offer strong QoS guarantees. Weighted round robin serves that purpose; weight factors are desirable both in order to provide service differentiation, and in order to support flow aggregation where different aggregates consist of different and variable numbers of flows.

We showed how distributed WFQ/WRR scheduling in buffered crossbars yields weighted max-min (WMM) fair rate allocations under a simplifying fluid model. We simulated the system under discrete cell traffic and measured the relative discrepancy of the actual rate allocations from the WMM fair allocations; we showed that small crosspoint buffer sizes (single-digit number of cells) suffice for excellent approximation of ideal WMM fairness (to within less than a few percent). This also shows that full utilization of oversubscribed outputs is approximated to the same excellent degree. We studied extensively the transient behavior of the system, when flows come and go or when weight factors change, and concluded and verified by simulation that restabilization at the new fair rates occurs within a delay time that is faster when buffers are smaller and when rate changes are larger; the amount of service unfairness during transients, when expressed in bytes, depends mostly on buffer size, and not on the magnitude of rate changes. In conclusion, buffered crossbars, which are technically feasible today, offer important advantages, including excellent quality of service guarantees.

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; http://tiny-tera.stanford.edu/∼nickm/papers/ToN_April_99.pdf

[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] A. C. Kam, K.-Y. Siu *"Linear complexity algorithms for QoS supper in input-queued switches with no speedup"*, IEEE Journal on Selected Areas in Communications, vol. 17, no. 6, June 1999, pp. 1040-1056.

[7] E. Hahne, R. Gallager: *"Round Robin Scheduling for Fair Flow Control in Data Communication Networks"*, Proc. IEEE Int. Conf. on Communications, June 1986, pp. 103-107.

[8] M. Katevenis: *"Fast Switching and Fair Control of Congested Flow in Broad-Band Networks"*, IEEE Journal on Selected Areas in Communications, vol. 5, no. 8, October 1987, pp. 1315-1326.

[9] 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; http://citeseer.nj.nec.com/hahne91roundrobin.htm

[10] 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; http://www-2.cs.cmu.edu/ hzhang/papers/INFOCOM98b.pdf

[11] D. Stephens *"Implementing Distributed Packet Fair Queueing in a scalable switch architecture"*, Master Thesis at Carnegie Mellon University, Department of Electrical and Computer Engineering, May 1988; http://www.andrew.cmu.edu/user/donpaul/research/myps/ms-thesis.ps

[12] F. Chiussi, A. Francini: *"A Distributed Scheduling Architecture for Scalable Packet Switches"*, IEEE Jour. Sel. Areas in Communications, vol. 18, no. 12, December 2000, pp. 2665-2683.

[13] 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; http://www.eecs.umich.edu/ taraj/iccCorrections.doc

[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. http://citeseer.nj.nec.com/cao99utility.html

[15] Kostas 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

[16] Abhay K. Parekh and Robert 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.

[17] S. Keshav: *"An Engineering Approach to Computer Networking"*..

[18] Nikos Chrysos *"Weighted Max-Min Fairness in a Buffered Crossbar Switch with Distributed WFQ Schedulers: a First Report"*, Master of Science Thesis, University of Crete, Greece; Technical Report FORTH-ICS/TR-309, Institute of Computer Science, FORTH, Heraklio, Crete, Greece, 150 pages, April 2002; http://archvlsi.ics.forth.gr/bufxbar