FORTH-ICS / TR-309 April 2002

Weighted Max-Min Fairness in a Buffered Crossbar Switch with Distributed WFQ Schedulers: a First Report

Nikolaos Chrysos[1,2]

ABSTRACT

Switches play a pivotal role in the design of high performance networks; thus, they have to be efficient and inexpensive. Input buffered switches require the memories to run at just twice the line rate, thus demonstrating very good scalability. However, they also need a centralized scheduler that configures their crossbar so that, at any given time, non-conflicting packets are forwarded; the complexity of this scheduler increases considerably with switch size. Moreover, it is doubtful whether such an architecture can provide sophisticated QoS guarantees without sacrificing switching capacity. By contrast, the scheduling task is dramatically simplified if small buffers are included at each crosspoint of the crossbar. In this thesis we analyze such a buffered crossbar architecture which allows distributed scheduling: distinct servers at each input and output collectively but still independently schedule the set of flows through the interconnect. The input and output servers implement Weighted Round Robin(WFQ-like) scheduling and are loosely coordinated through backpressure signals from the crosspoint buffers. We study how close this system approximates the ideal weighted max-min fair allocation. We provide extensive simulations and preliminary analytical hints indicating that our scheme converges to such a fairness objective. In addition, we study convergence time and we quantify the unfairness during the convergence process. We also study saturation throughput under various assumptions on the form of traffic. Although even a buffer space of one cell suffices for the scheduling operation, a buffer size of 4 to 5 cells per crosspoint yields excellent performance, at least for switches up to 32x32.

Weighted Max-Min Fairness in a Buffered Crossbar Switch with Distributed WFQ Schedulers: a First Report

Nikolaos I. M. Chrysos[1,2]

Computer Architecture & VLSI Systems(CARV) Laboratory,
Institute of Computer Science(ICS)
Foundation for Research and Technology — Hellas(FORTH)
Science and Technology Park of Crete
P.O. Box 1385, Heraklion, Crete, GR-711-10 Greece
email: nchrysos@ics.forth.gr

Technical Report FORTH-ICS/TR-309 — April 2002

Work Performed as a Master's Thesis at the Depart. of Computer Science, Univ. of Crete.

Keywords: Crossbar Scheduling – Buffered Crossbar, Internal Buffers, Distributed Crossbar Scheduler, Weighted Round Robin Scheduling, WFQ Scheduling, GPS, Weighted Max-Min Fairness, Weighted Max-Min Fair Equilibrium, Convergence Time, Transient States, Saturation Throughput

[1] ICS-FORTH, P.O. Box 1385, GR-711-10 Heraklion, Crete, Greece. E-mail: nchrysos@ics.forth.gr

[2] Department of Computer Science, University of Crete, Heraklion, Crete, Greece. E-mail: nchrysos@csd.uch.gr

Acknowledgments

I would like to thank professor Manolis Katevenis for proposing the system and for his excellent guidance during the development of this work. I would also like to thank professor George Georgakopoulos for introducing me to the "world" of theoretical-computer-science and for reviewing the proofs that I include in this report, Mrs Paraskeui Fragopouloy for the insight in the formulation/characterization of the dependency chains, and Vassilis Siris for his useful network/simulation comments/tips. Of course I thank FORTH for its financial support during my graduate studies. I also thank prof. Pnevmatikatos, for introducing me to the world of systems and for supervising me, under other undergraduate and graduate projects. I should not forget Mr. Harteros and Mr. Sapuntzis for discussing the subject of this work with me, the University of Crete, Computer Science Department, for giving to me the basic computer science/system knowledge and the University of Manchester for providing the facilities to learn about the cognitive/human side of computer science. Finally, I would like to thank my parents for supporting me all time long; my mother especially, for studying me Greek philology/philosophy when I was a kid, and my father for implicitly introducing me to the cognitive world and for insisting that I should study computers. I also thank my friends, some especially for introducing me to the world of positive science, but all of them, for giving me the opportunity to escape from work from time to time. Finally, I find it useful here, to thank anyone who might ever try to study this report, despite its linguistic – and in some points technical/content – incompletness/inefficiencies/inaccuracies; for the language point, I must say, that I know (greek)English primarily through experience.

Contents

1 Introduction

The exponential growth of Internet is accompanied with continuously diverging demand for service. To keep up with the growth in demand and with the variations in service requirements, networks have to be made both efficient and clever enough to discriminate traffic. In addition, clusters and multi-processor systems, that highly utilize communication, have still not found the appropriate communication paradigm that will boost performance, applications and computing solutions. Switching elements play the most crucial role in all these efforts, since these are the points where “contenting”(or “sharing”) traffic is being multiplexed. Switches role is actually twofold. For one, their role is to forward data packets according to their destination and secondly to allocate internal and external resources. Forwarding means configuring the interconnect so that packets will be routed to their proper destination, while servicing means to decide, when each packet will be forwarded. A good service discipline, is one that can discriminate among different types of traffic, that provides fairness, protection and isolation while keeping the output links highly utilized.

Creating feasible and efficient switching fabrics which will be able to enforce QoS to user micro-flows, is one of the most triggering subjects in the networking area. The feasibility of an architecture is defined over hardware-economic terms, while efficiency is more usually subscribed to high output link utilization, fairness and compliance with quality of service guarantees-requirements. Most of the efficiency problems can be resolved, with a relative increase on the hardware-economic cost – by using for instance pure output queuing or a small internal speed-up. Here we consider that no speed up is actually feasible, as may actually be the case, for big scales or high throughput external links.

The problem that we face then, is dependent on what we will consider as fair and efficient, given the constrains that we place upon the operating speed of the switching-scheduling fabric. Following a black box conceptualization of a switching-system, the ideal is that only flows competing for the same output content for resources. This point is critical for network designers/operators, when they want to insert switching points without having to worry about the in-switch congestion and interaction of flows directed to different outputs. But this view is also simplistic, since pure output queuing switches that realize this objective, are highly expensive on hardware-economic terms, primarily due to memory-datapath requirements.

Contention mainly occurs where traffic is multiplexed. This happen on wide packet networks, since we cannot afford a direct point2point communication for all nodes, this also happens inside switching fabrics topologies. Input buffered switches, which represent the current favored trend, for both researchers and commercial vendors – due to low memory-datapath requirements –, suffer from internal contention. Even a non-blocking switch, that is a switch that can schedule any traffic pattern, as long as this traffic can be accommodated by the output links capacities, sometimes incurs internal contention that should be cleared out in a fair manner. It remains an open issue, how to create efficient input queuing systems(control requirements), that will fairly discriminate traffic and in the same time will allocate full capacity. On the other hand, the research in scheduling single resources, has many good results to exhibit. Per-flow queuing with packet GPS-like disciplines are proved to provide excellent service; this is the ideal solution for weighted-statistical-mutliplexing, if coupled with an output queuing, switch architecture – but this scheme is expensive(memory throughput requirements).

The problem, or our motives have been to create an input-queuing scheduling/switching architecture, with properties that reassembles that of GPS scheduling in the single resource case. That is, *given the hardware-economic constrains that restrict our design area and enjoin the practice of input-queuing to be adopted, that implicitly adds contetion points, what is the best that you can do, to provide service similar with GPS, still without requiring an scheduler that cannot scale well?* It seems that there exists an architecture that can be constructed under reasonable costs and retains the weighted max-min fair scheduling property of GPS servers. In this thesis we present this switch architecture, which can be "easily" constructed and offers fair allocation of the internal resources to distinct input/output pairs – sessions. If each input/output pair is assigned a weight that represents the relative priority of the traffic using this path, then the proposed switch-scheduling architecture will allocate resources in a manner that closely approximates weighted max-min fairness.

So the contribution of this thesis is twofold. For one, we propose an input-buffered switch architecture that solves the problem of providing efficient discrimination in input buffered switches, under reasonable incremental costs. In this view we present the rationale that directed us to specific design choices. The other contribution is the experimental-theoretical analysis on various properties of the architecture, that can give some insight for the selection of critical parameters, that relate to the emerging, cost/performance trade-offs.

Our proposed scheme, also looks alike some recent proposal for distributed scheduling [DGPS][ChQoS]. While theirs and our work where done independently, we came up with very similar architectures. But in their proposals, they investigate the architecture with concern to specific QoS requirements – making several assumptions on the demand and the nature of flows, and probably under-estimating the switching capacity of the system. Here we investigate the core of the architecture, in an other view. The contribution of this thesis, is the analysis of the architecture in issues of fairness, when flows are utilize all the available capacity. More specifically, we have verified through simulations – and we are close to the target of analytically proving –, that the service provided by the switch core, will approximate arbitrary close the weighted max-min fairness objective, when all flows – input/output pairs – are continuously backlogged. In addition we discuss various other critical parameters in the process of reaching this equilibrium at the first place, and on the time it will take to stabilize in a new equilibrium, after conditions marginal changes. Our work, makes no assumptions on the nature of weights and we discuss the performance of the system in the general case of random weights assignments. These issues are critical also for the very-good work, done in [DGPS]; since they cannot allocate all available capacity for guaranteed services, the distribution of remaining/excess capacity issue, is of major importance. Finally we analyze through simulations, the saturation throughput of the system and compare the available capacity("good-put") of the switch core, under various scheduling alternatives and hypothesis on the process of flows arrivals.

The remaining of this thesis is organized as follows. On section one, we discuss on network objectives and advocate for the adoption of weighted max-min fairness, for which we present different definitions and algorithms that can be found in literature. On section two, we quickly describe background work, in the area of scheduling one resource and on the crossbar scheduling problem – adhering on critical in our view points. We describe our architecture on section three, arguing on important decisions. In the same section we derive results for the expected behavior of our scheme, under various objectives. On section three we present simulation results, which demonstrate the operation of the proposed architecture and we study the impact of the internal buffer space to the overall performance of the scheduling discipline. Furthermore, we present results, comparing various alternatives in the design spectrum for the same core architecture. Two expensive schemes for scheduling a bufferless crossbar are described and analysed at the end of this section. Finally, we reach the end of this thesis with discussions on section five and we conclude at section six, discussing open issues and future work.

2 Network objectives – policies, techniques and efficiency

Network traffic engineering, is the discipline concerned with the construction of effective data networks. By these means, network traffic engineering studies the problems of network topology, resource allocation and network demand, in the view of an objective which translates into criteria for evaluating network configuration. Down to the bottom, demand is the subject of the discussion whereas *both* topology and allocation are the design parameters that build upon and aim to serve demand.

Most *effective* networks, can be formulated only when all the above sub-issues are considered together and inter-dependently. Of course economic-social[1] and complexity reasons sometimes force the independent examination of each sub-problem. In the context of each sub-problem a different objective is often implicitly or explicitly used to formalize efficiency. One opportunity to consider *both* views together is when we examine simple network elements like switches, which usually enforce the allocation disciplines and by the same time, provide the means for constructing efficient, or demand-oriented wider network topologies. When designing these switches, the topology and the allocation problem are usually considered together in the view of a single objective[2]. Again a discipline may be proved to provide this objective but through methods which are either unimplemented so far, or very costly. In this case you might for instance try to find a discipline-method pair, which approximates the objective close enough, with additional cost constrains. These constrains are usually in the form of logic, buffers, links and technology.[3].

Technical network objectives may vary in various sub-areas, since each area needs – and builds upon – its own axioms of what is good or ideal. For example, when designing switching fabrics the non-blocking degree of the fabric stands as an axiom of the switch performance. This technical view can only take substance if the non-blocking idiom is translated (implicitly or explicitly)into better service for the flows using the network. When designing scheduling disciplines the delay of the flows, or the compliance with the requirements of the flows are regarded as the metrics to evaluate disciplines. Again technical concepts need to stand upon political positions on the nature flows and their relative importance. Even when administrative actions are taken to leverage, for instance, the congestion at downstream nodes, this is again based upon the political view that this will eventually be for the good for all the flows. Since the common place of most of all ideas for efficiency stand upon political positions on the way that the "shared" but disposable goods are actually "shared", the most primitive efficiency metrics can be nothing else than political. This is the first point in viewing efficiency and the final target of any network configuration. If it is to build a network-society we

[1] by means of installed equipment

[2] Of course local properties do not always translate into global, especially in the case of nearly chaotic infrastructures like the Internet

[3] with special dynamically and application-related, cost-weights on each of-course

must first locate the units – the members in the sharing process – that constitute this society and furthermore we need to institute the first-axioms that we want to dictate the sharing process between the members. We have to differentiate among members in a way that will enable us to define the absolute equality in the sharing process. Then we can discuss on fairness and effectiveness(although the term is mostly technical) of any technique.

2.1 Network objectives – the utility applications perspective

More commonly links utilization and the fairness of the allocations have been considered as the desired properties. Only recently the ability of complying with QoS contracts to flows has been added to this set of objectives, an introduction which merely has changed the networking paradigm [ShNetObj]. Most of these objectives can be formulated into a single maximization problem, which maximizes a function over the utility that each flow (defined as a grouping of network packets that share routing-quality of service) of traffic gains during any time interval.

What follows is a general formulation of an utility-based objectives class that has similar structure to that presented in[ShNetOnj]. This is based on the notion of application utility, a conceptual manufacture that captures the application degree of satisfaction through the service provided by the network[4]

2.1.1 A generic/ideal model for defining efficiency

If we partition network time into discrete elementary time intervals, then we might accept that the marginal utility that a flow f gains during one elementary time interval t_i, can have the form:

$$\mathcal{MU}_{t_i}^{f} = \mathcal{B}_{t_i}^{f} \ / \ \mathcal{P}_{t_i}^{f}$$

if flow f has a packet for transmission on the beginning of time slot t_i, or

$$\mathcal{MU}_{t_i}^{f} = \mathcal{O}$$

if there are no packets for transmission on the beginning of time slot t_i.

Hypothesis That is the marginal utility of each flow of traffic, is a function of the instance bandwidth $\mathcal{B}_{t_i}^{f}$ allocated to this flow, divided by the instantly formulated priority index $\mathcal{P}_{t_i}^{f}$. In case that you find this hypothesis here somehow abrupt, you can skip to section 8 , for a presentation of the hidden reasoning that drove us to consider dynamic(or per packet) weights. The basic stand here, is the assumption that a flow with high priority needs extra bandwidth to get the same utility with a flow that has small priority index. Now given a finite discrete time interval $\mathcal{T}$, the utility that a flow perceives during this internal, will be simply the sum of the marginal utilities gained at each time slot within this interval.

[4] Shenker interpreters utility as the willingness to pay. Of course willingness to pay is subscribed to the humans that are the final acceptors.

$$\mathcal{U}^f_{\mathcal{T}} = \sum_{\mathcal{T}} \mathcal{MU}^f_{t_i}$$

In this model we assume that priorities are dynamically assigned to flows in response to the previous service that the flows experienced, in combination with the inherent characteristics of the flows and the applications that uses them. In contrast with the simplistic view that applications (and by so flows) have constant need of bandwidth, the utility view reveals the fact that not all marginal bandwidth changes have the same impact on the application well-being[ShNetObj]. Utility - bandwidth plots(*Fig. 1*) that can be found in [UtMaxMin] project this fact. Although these plots are static, since these stand upon an average behavior characterization, these manifest the first step needed in the political action of differentiating between flows.

We argue here that flows are not only elastic or only inelastic but can be all these together, since all that a flow basically needs is this service that will make the application work better. The idea is that if we get down to the bottom of the details, a dynamically changing priority index, can serve the purpose of reflecting the dynamically changing *urgency* for service in a dynamic execution environment. Here we propose dynamic weights as a technicality that can capture better the differentiation needed to allocate the absolute fairness in a best-service network. We do not build upon the specific techniques that will make the idea to work – should a flow receiving small service upgrade this urgency of service or place smaller demand – but we grant it as a general rule in a general problem.

Now that we have expressed a formula for the utility a flow perceives during one time interval, we can formalize a general class of network objectives. If it was possible, we would require from a network to distribute equal utilities to all flows. But since we cannot expect flows demand to be perfect-uniformly distributed on the finite resources, this is not always desired and possible. It would be simply inefficient, to bound the utilities of all flows, to the lowest affordable – probably that of the flows using the most congested points. It follows that efficient networks, will apply diverged utilities/good to flows.

$$\mathcal{EFFECTIVENESS}^{network}_{\mathcal{T}} = \mathcal{OBJ}_{f \in F}(\mathcal{U}^f_{\mathcal{T}})$$

where F is the set of flows competing for network resources during the time interval $\mathcal{T}$.

So this meta-class of networking objectives can be perceived as maximizing a *function* on the utilities that the flows using the network, are allocated. It can be seen as having two parameters, which when specified, direct to a specific objective. The one is the priorities mechanism and the other is the choice of the above function. There is no insight, on which should be considered first or second. Note though, that dynamic priorities can differentiate application as having different motives at different times(bandwidth,latency), while the objective function can differentiate between groups of application, as “sharing” and by these means

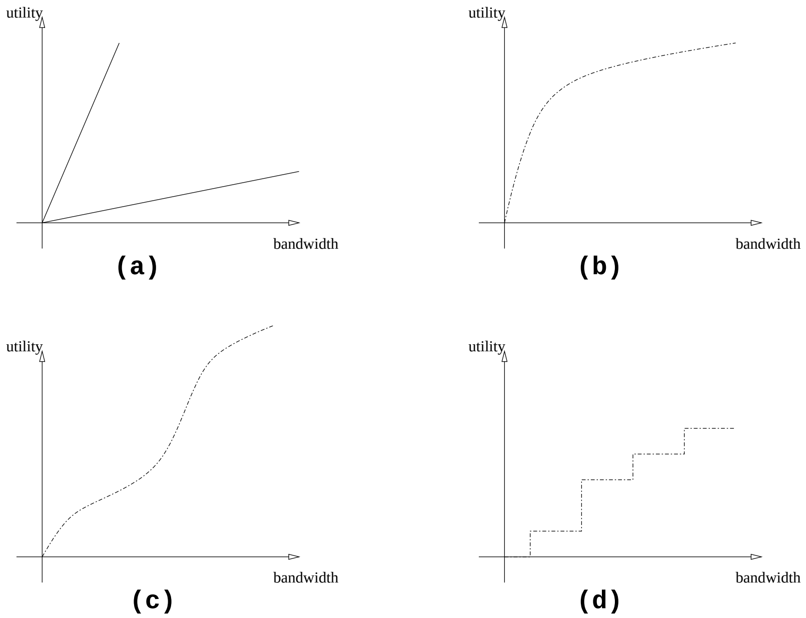


Figure 1: Utility bandwidth figures. Figure a, correspond to constant priorities and linear utility functions. Figure b,c correspond to rate elastic applications, while figure c is a stepwise utility function, that could be subscribed to delay sensitive applications.

be allocated resources, with different inter-intra-politics on groups. More basically we can differentiate upon the notion of the application alone-an application (is a a packet, a flow of packets, an aggregation of flows or an institute?). Differentiated priorities and/or policies may coexist and form as a whole, a single objective.

2.1.2 Model instances and weighted max-min fairness

We can conceptually distinguish between objective function, although the criteria are not always absolute. Some objective functions, can be technical, in terms of defining efficiency on the technical issues(capacity), other can be operational(traffic smoothness) and other can be seen as providing the means for the social welfare, in terms of fairness issues. A simple class objective anticipated could be for example that the sum of the utilities should be as max as possible for any time interval[ShNetOnj]:

$$\mathcal{EFFECTIVENESS}_{\mathcal{T}}^{network} = \sum_{f\in F}(\mathcal{U}_{\mathcal{T}}^{f})$$

Clearly this objective is very difficult to capture and as many papers correctly comment, this objective may not always drive fair disciplines. Note that the definition for network objectives that we have just presented differs with this presented by Shenker, in that it does not necessary maximizes the sum of the “perceived” utilities. The objective function can be any arbitrary function upon the flows utilities field. Maximizing the sum of the utilities – although in some sence directs to maximum total social welfare – does not ascertain any fairness in the distribution of resources. It can easily be the case that the maximization results to some flows-applications consuming almost all the utility, where others take nothing.

Max-min fairness is this objective(belongs in the formulated class), which makes sure that the minimum utility received by a flow, is as maximum as possible. That is it suffices to change the objective function of the above general form to the minimum set operation.

$$EFFECTIVENESS_T^{network} = MIN_{f \in F}(\mathcal{U}_T^f)$$

So this criterion ranks networks and allocation disciplines, based on the minimum utility they allocate to flows. A weighted max-min fair discipline will be the one, that assigns the highest minimum possible utility over all the flows. As we shall see, this objective can be realized by some discipline and actual networks. An Other "fair" oriented objective is the proportional fairness. It can been shown that in the case of a single bottleneck, the proportional fairness criterion can be reduced to a maximization problem, where the utility functions are the logarithms of the assigned rates[5]([FaCongCont]).

The basic argument against max-min fairness, is that it may lead to performance degradation for all member[6], but this again is more alike a technical argument and not a political one. It should be obvious that the shared medium should be capable to sustain a minimum performance for all fair demands. If this is not the case, we cannot see who will decide who will get access to the sharing process and who will be left out. On the other hand proportional fairness takes the position that the more links you use, the less bandwidth you get. It has a very good-intuitive background on fairness and efficiency, since this basically tells us that the more you content with, the less you "deserve" to receive from each one. But in addition it is very pessimistic to realize and accept that when the distance of the communication path grows the performance degrades. In my view, since this is not how want the global interconnect to look like on higher levels – the target on the application-human level is to wipe out the distance factor – we should not conceal it on its groundings.

2.2 Weighted Max-Min Fairness

.

2.2.1 Intuitive description of the objective

We have seen that max-min fairness is this instance of the above class of network objectives, which makes sure that the flow with the minimum utility allocation gets as more utility as possible. This intuitively – and literally – means that if one different schedule existed, which would in the same time interval give more utility

[5]Generalization that include weights are possible

[6]This is the same argument that favors admission control

to a flow, then this allocation would have "steal" this extra utility from a flow, with equal or less utility in the first schedule. It again means that if a schedule could carry more traffic from a flow, without risking other flows utility, then it probably should cause otherwise it is not fair in the max-min view. Max-min fairness also provides some kind of protection/isolation, since it ensures that given the worst-case "appetite" of others, every flow will be protected, since every flow will sustain a minimum fair share.

A simple and intuitive method that assigns to each flow its fair share, is the following: *start giving to all flows infinitesimal bandwidth chips, proportional to their weights, until some flow is out of demand - in which case you stop serving this flow*[7] *-, or some resource has been exhausted - in which case you put aside this resource and all flows using it*[8]*. Continuing till all flows have been satisfied, or all resources have been consumed, you get to target.* This constructive but imaginary algorithm, best describes the fairness grounding of this objective.

2.2.2 Different but equivalent constructive/situational definitions

Formal definitions of max-min fairness have been independently proposed in [HaydenMaxMin][JaffeMaxMin]. Here we have made a small change to introduce the utility generalization to max-min fairness, similar to what done in [UtMaxMin].

Lets consider for each realized allocation S, out of all possible/feasible allocations S^*, a vector of dimension equal to the number of flows under consideration. Each entry in this vector represents the utility that a S assigned to one flow. If this vector is ordered in terms of utility values we call it, an utilities-allocation vector.

$$V^S = (U^S_{f_1}, U^S_{f_2}, , , U^S_{f_n}) : U^S_{f_1} \leq U^S_{f_2} \cdots \leq U^S_{f_n}$$

Now an allocation is weighted max-min fair, if the utility-allocation vector that it allocates to flows, is the lexicographically greatest possible.

$$S \text{ is max-min fair} \equiv \qquad V^S >_{lex} V^x \qquad\qquad \forall x \in S^*$$

That is, the minimum utility is as max as possible and subject to this we allocate the next minimum utility and so on. This problem can be formulated with linear programming([HaydenMaxMin]) and given the assumption that utility is an increasing function over bandwidth(finite resource in general), it solves into a single equilibrium.In a single resource case, this definition of weighted max-min fairness reduces to the following contructive algorithm. Starting from zero, all flows receive equal incremental utilities, until some flow(s) has no more demand or some resource capacity is exhausted. When the first event happens the excess capacity is normalized-equally distributed to other, through the same procedure. In the case of M resources, we iterativelly find the flow with the smallest possible utility assignment on some single resource, we allocate this utility to this flow, we remove

[7] a satisfied flow is said bottleneck at source

[8] these flows found their bottleneck – becomed bottlenecked –, on this resource

it from the competing set, and continue until all flows are removed.

A different formulation of the same problem can be seen, if we introduce the term of bottleneck. A resource (link) is said to be a bottleneck for a flow, if this flow gets the maximum utility at this resource-over all other flows using it-and this resource is fully occupied. Then an allocation will be max-min fair if every unsatisfied flow[9], has at least one bottleneck. To see this, notice that if this was not the case, then we could have assigned more utility to some flows – those without bottlenck –, without risking the utility of flows with less or equal utility, so the utility vector was not the lexicographically largest possible.

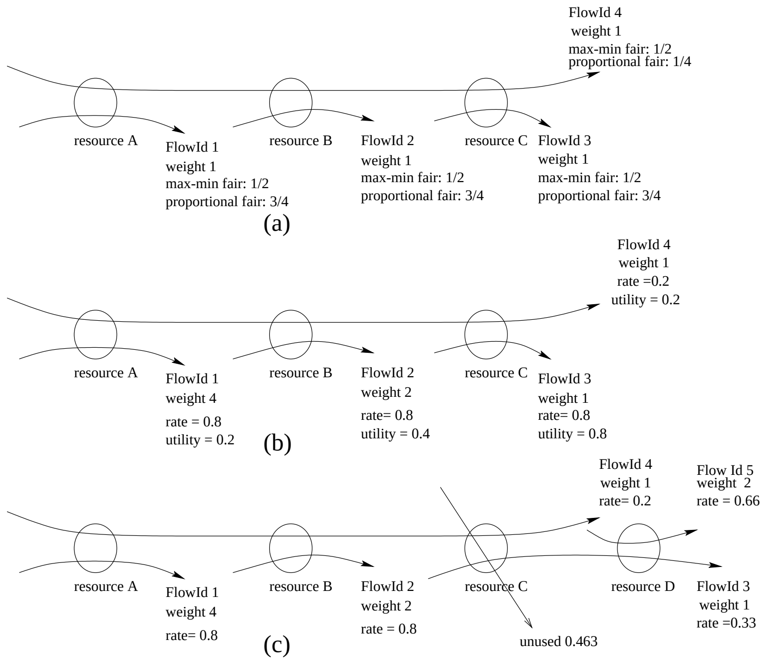


Figure 2: In this examples we have four flows not constrained in source, contenting for three resources(links) A, B,C of the same capacity. Flow 1 uses only resource A, flow 2 uses resource B, flow 3 utilizes resources C and flow 4 contents for all resources.

In *Fig. 2* we examine the fair rate allocation for a simple network configuration, under different weights assignments for the flows using it. In (a) all flows have the same weight, so the max-min fair rate is half the capacity of one resource for each flow[10]. With proportional fairness, the rates assignment would be 1/4 for flow 4 and 3/4 for all other flows, since this maximizes the $\log(rate^1) + \log(rate^2) + \log(rate^3) + \log(rate^4)$ under the resources finite capacity contrains. In (b) the most bottlenecked resource is resource A – since a fair allocation on this node would assign equal utilities for flow 4 and 2 equal to $1/6$, when the fair allocation in all other resource is greater. So we must certainly assign to flows 1 and 4 their fair share at this resource, that is $1/5$ to flow 1 and $4/5$ to flow 4. Note again that this

[9]said bottlenecked at source

[10]If it was to give more capacity(rate=utility under equal weights) to some flow, we would have to take it away from some flow with less or equal capacity

rate assignment, results in equal utilities for these two flows. Continuing at the second most congested resource – which is in general depend-ant to the bandwidth reservation we made so far –, we reach node B. We observe that we can give all the remaining capacity to flow with id 2, since flow with id 1 cannot utilize its fair share on this link (1/3)[11]. So the rate assignment for flow 2 is 4/5. The same occurs on resource C with contending flows 3 and 4. Although these two flows have equal weights the rate assignment is 1/5 and 4/5 for flow 4 and 3 respectively.

Note here that although the weighted max-min fairness discipline is efficient, it does not always retains maximum utilization for the links. If for example the flow with id 3 had less demand than the 4/5 of the link, but lets say could only occupy the 2/5 of the total capacity, then resource C, would remain unoccupied during the 2/5 of time. This scenario is realized in case (c) of *Fig. 2* . Here we have added one more resource and one more flow. Flow with id 3, now uses two resources(C and D). This flow has a more strive bottleneck on resource D and can only use the 2/5 of this resource. So this flow is confined on this rate leaving 2/5 of resource C to remain unused. For this example a max occupation discipline would assign 3/5 rate, for flow 1, 2/5 to flows 2,3,4 and 3/5, to flow 5.

Note that flows, assigned rates and utilities at subsequent steps of the procedure are favored and get more utility. This is because these flows perceive less demand in the weighted max-min fair view. These arguments describe the intuitiveness behind the demand for max-min fair networks. Algorithms for computing the max-min fair allocation can be found in [Keshav97][UtMaxMin].

2.2.3 Analytical algorithms

In [Keshav97] the algorithm presented stands an constructive definition of the notion of weighted max-min fairness. It is presented as follows. Let $d_1 \ldots d_n$ be the demand and $w_1 \ldots w_n$ be the priorities assigned to sources $f_1 \ldots f_n$ respectively. Then to construct a weighted max-min fair allocation one has to do the following:

- **Resources are allocated in order of increasing demand, normalized by the weight**
- **No source gets a resource share larger than its demand**
- **Sources with unsatisfied demand get resource shares in proportion to their weights**

The above description-algorithm for weighted max-min fairness is actually a mixture of a situational description – look rule 3 – and a constructive one. It can be shown to reduce to weighted max-min fairness, if it is interpreted correctly[12].

[11] This flow confronts a more strive bottleneck on resource A (1/5)

[12] The more common pitfall – we fall in at start – is in the first rule, where demand should be interpreted as being dynamically formed, in parallel with the construction of the fair shares.

Additionally it captures the cases, where demand is finite and smaller than the max-min fair share in the never-satisfied-demand case. The more "programmable" algorithm that we present bellow ([UtMaxMin]), is the algorithm that we have used for computing the ideal allocations, in our simulation models.

At each iteration of the algorithm the following three steps are executed. We start by marking all flows as being *unbottlencked* and end when all flows are marked *bottlenecked.*[13].

1. At each link l, determine the available bandwidth for the unbottlenecked flows $i \in U \cap F_l$ crossing l:
   . $A_l = C_l - \sum_{i \in B \cap F_l} r_i$
2. At each link l, allocate temporary bandwidth to each unbottlenecked flow crossing l, by finding the bandwidth allocation that either saturates the link and results in equal utility, or absorbs the demand of a flow f:
   $\forall i \in U \cap F_l$, allocate $t_{i,l}$ such that:
   . $\sum_{i \in U \cap F_l} t_{i,l} = A_l$ and
   . $\forall j \in U \cap F_l, f_i(t_{i,l}) = f_j(t_{j,l})$[14].
3. Find all flows with minimum allocated temporary utility and label them *bottlenecked.*
   if there exists a link p and a flow g such that
   $f_g(t_{g,p}) = min_{l,i \in U}(f_i(t_{i,l}))$ then
   . $B = B \cap \{g\}$
   . $U = \overline{F \cap B}$
   . $r_g = t_{g,p}$
   endif

To verify that this algorithm correctly computes the weighted max-min fair shares, note that all flows will be finally bottlenecked and that the minimum utility received by all flows, is as max as possible. If we could assign to some flow f, more utility at some step, then we would either have to steal it from a flow g with less utility (assigned at a previous step of the algorithm), or to steal it from a flow j with equal utility(on the same node), or to over-subscribe f's source.

If all flows are continuously backlogged, the algorithm will terminate after at most m iterations, where m is the number of the links l in the network. To see

[13] f is the set of flows, U is the set of unbottlenecked flows, B is the set of bottlecked flows, F_l is the set of flows crossing link l, C_l is the capacity of link l and $f_i(x)$ is the utility of flow i for the value of bandwidth x

[14] This is easy, simply normalize the weights of the not already backlogged flows(i) using resource l ($\frac{w_i}{\sum_{k \in U \cap F_l} w_k}$) and assign to each rate equal to the product of the available capacity with the respective normalized weight ($w_i / \sum_{k \in U \cap F_l} w_k \cdot A_l$). If a flow cannot absorb this rate because this is bottlencked at the source, you can simply compute the flows that gets the minimum utility – due to source constrains – and continue since this flow will get the minimum utility on this node and this is what we are interested in. Otherwise this operation results in equal utilities for flows.

why, note that when at some iteration we find the minimum utility allocation (e.g on link l) over all flows and links, we have certainly found the allocation for all the flows crossing this link l. So at most after m iterations, all flows will have their final rate assignment. As noted in [UtMaxMin], a tighter time-complexity bound can be obtained. The algorithm actually terminates after at most b steps, where b is the number of bottlenecked links in the network.

2.3 On the nature of weights

Ending the objectives section, we will discuss methods for assigning weights to flows – or better, applications of weights on current networks. In earlier paragraphs of this section(section 2 , section 8), we advocated for a dynamic scheme for assigning weights, based on the nature of flows and the state of the network. In this scheme we expect weights to represents the relative urgency of flows/applications. This is intrinsically the same utilization of weights, also used by many adaptive scheduling disciplines that try to obliterate (objective-based)discrepancies in the actual service, by developing a dynamic weight assignment. Instances are the LRU and many other, that we will briefly discuss in the next section. Here we will discuss more practical applications of weights in current networks.

For one weights can be assigned, in the aim of providing end to end delay bounds and rate assignments, that is QoS guarantees to flows. Hard bounds normally require admission control schemes to bargain the deal. Softer bounds that are more relax and only provide an average of the desired behavior, may need no admission control. In both cases weights can sustain the objective, if these are used in conjunction with service disciplines that realize them. In an other view weights can be used to allocate constant bandwidth to institutions or other organizations – whenever these have traffic to send. This differs with the previous utility of weights, in that it does not relate to the requirements of a single application – but rather to groups of people, or set of applications. Other methods may use weights to discriminate among routes(paths)of a network, in coordination with a flow/congestion control scheme.

Finally weights can be used in sub-networks where micro-flows are aggregated and handled as one entity. In these occasions, weights can be assigned to groups of flows for operational reasons. In switches for instance, all the flows that traverse the same route – stem from the same input and are destinated to the same output – can be aggregated and handled as one, for the sake of simplicity. In this case the weight could be the number of the ingredient micro-flows that are active in each path of aggregation. Then, if flows-aggregates/routes are served in proportion to their weights, per-micro-flow, fair allocation on rates is guaranteed. But of-course if simple $FIFO$ is used to keep all the traffic of the same route, then the micro-flows in one route are not protected from each other.

It should be clear that we do not actually stick to one type of weights utilization, in the analysis that we will present. The only hypothesis is that weights, will more or less under any circumstances, represent the relative-weighted priority of flows-sessions(streams of data packets/cells that share something and can be regarded homogeneous). We also use the term packet/cell to speak for the unit of traffic/transmission. In the absence of further comments, you can fairly regard, that variable size packets are implied. Note though, that in the simulation experiments, fixed size packets(cells) where used.

3 Background

In this section we discuss background work related to the problem of scheduling. This work can be partitioned into two (disjoint?)groups. On the first path we find scheduling disciplines that allocate *one single* resource common to the demanding competitors. On the second path we reach the problem of constructing a set of distinct input/output pairs that are non-conflicting under the given topology characteristics of a switch. Here the switch topology that we will examine is a crossbar.

3.1 One resource servers

.

3.1.1 The problem

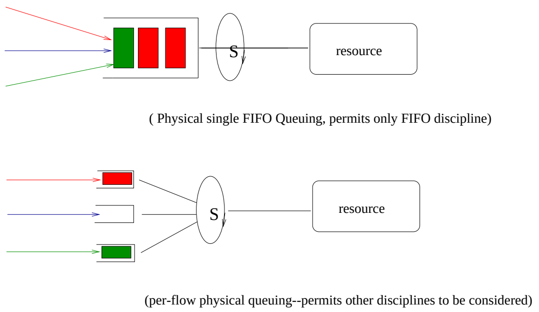


Figure 3: The single logical $FIFO$ in front of a scheduler, dictates the scheduling discipline to $FCFS$. This discipline cannot protect flows from misbehaving ones. Per-flow queuing permits other better alternatives to be used, since it isolates flows histories.

Service disciplines distribute a common and *finite* resource capacity to contending clients, or requests in general. In packet-networks the finite resource is primarily bandwidth(bits/sec) – wires capacity or number of optical wavelengths – so service disciplines differentiate on the allocating, different rates to different flows. Through an other view, servers/queuing-disciplines, differentiate on the allocation of different waiting times(delays) to different packets, which is the minimum unit of transmission in packet networks. Packets are conceptually organized into groups(flows), defined as a set of packets that *share* common routing and quality of service requirements[15] So disciplines can be though of as working for the good of the flows. Traditionally flows are seeking small delay and high service rate, probably including small jitter(maximum difference in experienced delay).

[15] Usually the requirement of transmitting(delivering) in order is added.

3.1.2 Work conserving disciplines and the conservation law

In a basic level, servers can be categorized as being either work-conserving or non-work-conserving. A work conserving discipline always “consumes”/distributes available capacity when there is demand, whereas a non-work-conserving discipline may stay idle – serving probably some other more profound motive. It is not difficult to see that if a work-conserving server is used, the average delay over all requests R serviced during a time interval T, which is equal to :$\frac{1}{|R|} \cdot \sum_{r \in R} d_r$, will be constant, since if this server reduces the delay of one request it will increase the delay of an other. Expanding this formula by aggregating individual packets(requests) into flows(groups), we obtain that if N is the number of these groups and λ_i is the rate that group i has been served, the sum of the normalized average delay of flows is constant – $\sum_{i=1}^{N} (\lambda_i \cdot d_i) = c^*$ – and independent of other features of the scheduling discipline. This result which is known under the name of Conservation Law [Keshav97], makes clear that what a sophisticated discipline can only achieve, is to distribute the delay to flows in different ways, but cannot result in less average delay or more aggregate rate, than a simple $FIFO$ or any other work-conserving technique.

3.1.3 Some known-adaptively fair-alternatives

Known queuing disciplines include Least Recently Used(LRU), Round Robin(RR), First In First Out($FIFO$), Last In First Out($LIFO$), Longest Queue First(LQF), Oldest Queue First(OQF) etc. Simple $FIFO$ does not protect flows from misbehaving competitors *Fig. 3* and other disciplines (LRU, RR, OQF) try to solve this problem, still without actually differentiating in the nature of flows, but through differentiating them based on the actual service experienced by them so far. So RR for examples, after serving a flow’s packet, conceptually assigns temporal-higher priority to all other flows.

3.1.4 Differentiations – coming closer to the QoS paradigm – and the GPS model

Differentiation is realized either with strict priorities – example: always server flow 1 when this is active – or through weights. Weights provide the more elegant solution for sophisticated service disciplines, whereas priorities should only be used for critical-special resources/flows, since this discipline can drive low priority flows to starvation. Weights usually represent the relative priority of flows, and either translates into a minimum rate and/or maximum delay guarantee, or simply stand as an index of the urgency degree.

Most weight based disciplines are built upon the Generalized Process Sharing(GPS) fluid model [ParGalPGPS]. In this conceptual/ideal but physical model, the unit of transmission is infinitesimally small, so that all active flows – those that have queued/waiting packets – can be served simultaneously[16]. The GPS discipline, serves all active flows in proportion to their weight, so it assigns to them equal marginal-utilities (rate_of_service/weight). Each flow in the set of flows F, using the same server, is guaranteed a minimum service rate $r_f^{min} = \frac{w_f}{\sum_{w_g \in F} w_g}$ under all times when this will be backlogged, and will actually receive during a period with the set of active flows(B) constant, rate equal to $r_f = \frac{w_f}{\sum_{g \in B} w_g} \geq r_f^{min}$. This model has very nice properties, desired in packet networks, like assigning to each flow its weighted max-min fair share, ensuring protection and isolation – flow are guaranteed service independent of other behavor –, is memoryless – no flow is punished latter for receiving bandwidth at low congestion times –, perfect rate assignments etc. Packet-system schemes based on GPS like Packet-GPS and Weighted Fair Queuing($P-GPS$,WFQ) proposed in([DemKesShenWFQ],[ParGalPGPS]) respectively, try to keep up with GPS, by serving packets in the order that these would finish service in the fluid world. Although this is not always possible, since it requires either to stop being work-conserving or to forecast future arrivals[17], it is proven in [ParGalPGPS] that the service received by a flow in a packet system emulating GPS with this way, cannot be smaller compared to the service that this flow would receive in the ideal GPS, for more than one maximum length packet. Using this relationship it is possible to assign delay bounds to leaky-bucket constrained flows[18]([ParGalPGPS]).

3.1.5 Virtual Time implementations

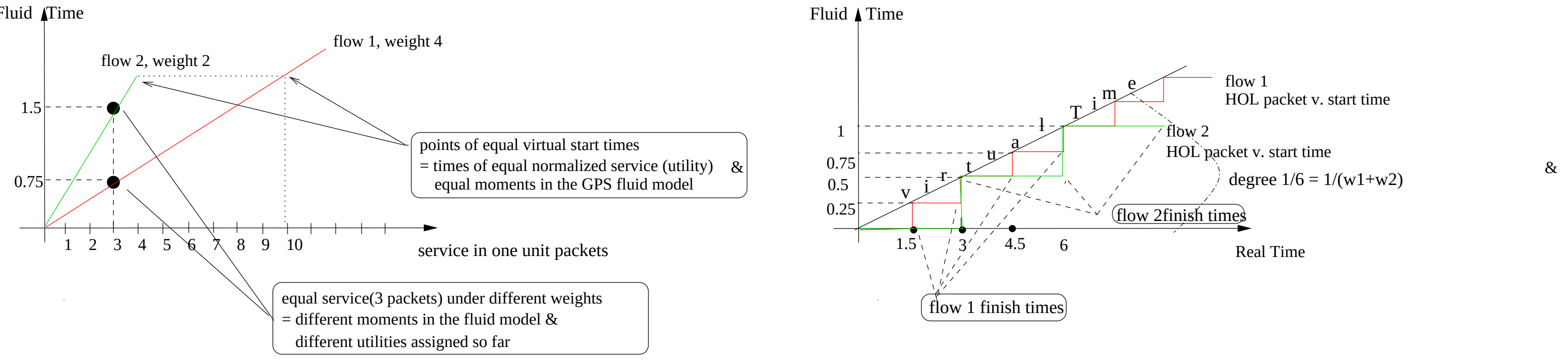
Packets systems that emulate GPS use the technique of virtual time. For each flow f, a virtual start time is maintained(S_f^i), a metric that captures the time in the fluid model(virtual) when this flow's $i-th$ HeadOfLine packet (HOL) would start service. In a constantly backlog scenario for the flows, equal virtual start times for a set of flows at some point, means equal normalized service(utility) for these flows. This is shown in plot (a), of *Fig. 4* . If we know the virtual start time of a flow's packet p_f^i we can compute the virtual finish time of this packet F_f^i, by simply adding $\frac{L_{p_f}^i}{w_f}$.

In addition a global virtual time (VT) is computed (can be found and as system potential) that at every time t, increases with rate inversely proportional to the sum of weights of all active flows in the virtual model, so it increases at the rate that all backlogged flows receive fair(=equal), normalized(=divided by the weight=utility)

[16]...not all their packets, but only the packets in the front of their respective queue(HOL). The $FIFO$ order among a flow's packets is maintained. Of course all packets will become HOL before these are served.

[17]During time t, when the packet-server is about to make a new scheduling decision, this flow's packet that would finish service first in the fluid model, may have not arrived yet(think that this now inactive flow has, relatively big weight or relatively small packet size).

[18]A flow is (b, λ) constrained, if this can sent with average(long term) rate λ and maximum burst(back2back packets) b.



(a) (b)

Figure 4: We assume uniform and unit size packet for both flows, continuously waiting for service. Flow1 has weight 4, while flow2 2. It appears that these will be serviced with rates, 2/3 and 1/3 respectively. Virtual start times, is defined uniquely on different packets of flows(this can be seen in (b)), but can be generalized into the notion of fair service received by a flow so far(as plotted in (a)). Virtual start times of flow, progresses with rate inversely proportional to that flow's normalized-weight, so progresses with the rate that this flow receives marginal utility, at least for times when this flow is backlogged. Equal start times for backlogged flows correspond to equal times in the fluid world. In the fluid world, flows fair service(=utilities) progresses with the same rate for all backlogged flows, with respect to time.

service(see *Fig. 4* (b)). This is used to re-insert a previously inactive flow f, into fair competition with others, by calibrating the new virtual start time of this flow in case this was left behind in virtual time. In this case the server asks "how much fair service would have other flows received in GPS, during this flow's inactive period?". The answer is that when this flow becomes active(same time for both the packet and fluid model), other flows, would have received fair service that incremented with rate inversely proportional to the the total weight of the flows, that were active during this time. The system potential, which at any moment increases with rate inversely proportional to the sum of weights of active flows in the fluid system and becomes zero if all these flows are inactive, represents this normalized service, so if we want to be as close possible to the fairness of GPS, we should compute f's new virtual start time at this point.

So summing up, flows served in increasing order of virtual finish times and the aim of the scheduler is to keep the virtual finish times of flows close, since this is what happens with the conceptual model - all flows receive equal normalized-service - and the virtual finish times(and virtual start time) progresses with the rate of this service. When flow's f $i-th$ packet arrives to the head of queue, (either arrived in an empty queue or the packet in front of it has just been served, or..), the following equations are used to compute the new virtual finish time of this packet:

$$S_f^i = max(F_f^{i-1}, VT)$$

$$F_f^i = S_f^i + \frac{L_f^i}{w_f}$$

This discipline, as described so far, requires to emulate the fluid model in the background, since it requires to maintain the set of backlogged flows in the fluid world in order to compute the virtual time. This translates into operations not only when a packet is to be served at the packet system, which usually stands as a minimum complexity requirement for packet networks servers running at high speed. Heuristics have been considered, that can approximate $PGPS$ well enough, without emulating the fluid world, but through extracting properties from the packet system.

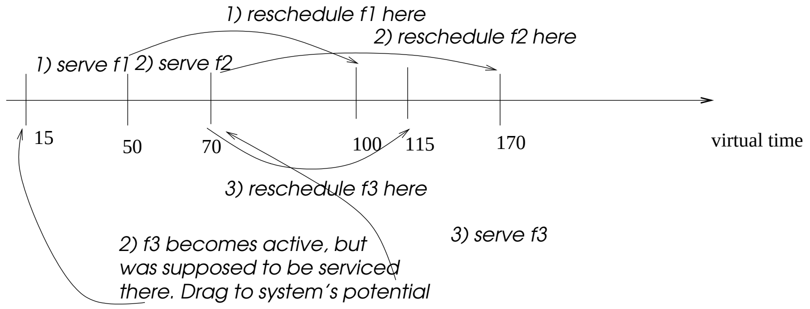


Figure 5: One execution example of the WFQ-like discipline, that belongs to the family of the WRR ([KaSC91]) scheduling disciplines, implemented through the virtual clock method. Flows 1,2 and 3 have service_intervals 50,45 and 100 respectively. Events 1), 2), 3) correspond to continuous scheduling cycles. At time 1 the active flow with the smallest next_service_time is flow 1(virtual time 50), which is served and re-scheduled at virtual time 100. At time 2 the flow with the smallest virtual time is flow 2(VT 70), which is served and re-programmed for virtual time 170. At the end of this cycle flow 3 – which was inactive – becomes active again and is "dragged" to the current virtual time(70), since its previous next_service_time was left behind(15). It is served at step 3 and reprogrammed according to its service interval

A WFQ-like discipline, with Virtual Implementation We primarily use one technique, (WRR/WFQ-like). Although other implementations of the WRR discipline are possible (i.e.: based on frames, where weights represent number of services per frame), we describe here a virtual time implementation. This only keeps the virtual next_service_time of flows, and instead of weights uses service intervals, a metric which is inversely proportional to weight. Flows are served in increasing next_service_time order and when a flow is served its next_service_time, is updated, by adding the service_interval. Now if a flow becomes active and its next_service_time is smaller than all next_service_times of the other active flows, then it is assigned a value equal to the minimum of these. So the virtual_time is defined to be equal to the minimum next_service_time of all active flows(again this variable propagates in time, with rate inversely proportional to the sum of active flows, but now in the packet system and not in the fluid world *Fig. 5*).

The discipline that we have just described, has many common points with

Smallest Start Time, or other WFQ-alternatives/implementation-cost-simplifying-WFQ-disciplines. Note that we use the general term WRR, to refer to this discipline.

Virtual Clock Two other techniques that will concern us, both for different reasons, are the Virtual Clock (VC) proposed by L. Zhang in a revolutional for the time paper [ZhangVC] and WF^2Q+ proposed in [BenhZhangWFWF]. Virtual clock differs with many other weight-based disciplines, in that it punishes flows for receiving fair bandwidth in the past and by the same means gives priority to previously inactive flows. In the WRR terminology you can think, that when a flow is left behind in virtual time – because this was inactive during a time interval and missed scheduling opportunities – when becomes active again, its next_service_time is left were it was and is not calibrated with that of other flows, potentially resulting in a burst of services for this flow, until its virtual finish time reaches that of the others. Although this is not a very good characteristic, since it may result to periods of starvation for well behaving flows that fairly utilized excess bandwidth – and through this rationale it does not offer protection – it may be considered in some special cases, for example in non-work conserving schemes, or in schemes were we want to give priority to inactive flows.

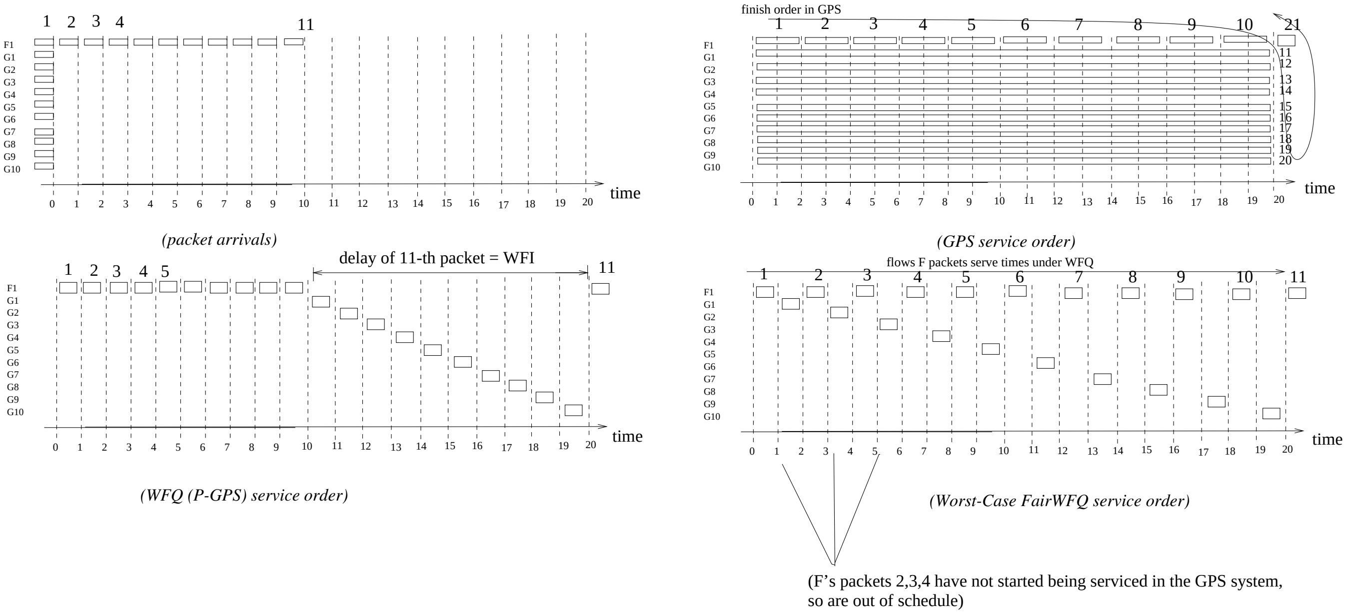


Figure 6: The scenario where $PGPS$ provides non smooth-schedules and increases the jitter of flow F_1. F_1 has a weight of 10 and G_i all have a weight of 1. 11 equal size packets(1 unit) of flow F_1 arrive back2back, starting from time 0 and one unit-sized packet of each other flow arrives at time zero. Since all G_i have virtual finish time 20 – since then these finish service in the GPS, the $P-GPS$ discipline will first serve 10 packets from F_1 and then service G_i's packets.

WF^2Q: **The optimal GPS-related packet disciplineRR pure generalization** WF^2Q+ ([HierFairQ]) on the other hand is an accurate and inexpensive approximation of WF^2Q, which is proved to be the optimal GPS-like, packet discipline. The distinguished feature of WF^2Q, is that it provides much more smoother allocations to flows than $P-GPS$. *Fig. 6* that can be found in [BenhZhangWFWF], presents an example that explains the superiority of the latter scheme. Consider a flow $F1$ with weight 10 and ten other flows G_i with weight 1, that start competing at time zero. Flow $F1$ should receive half the available bandwidth and each G_i, one tenth of the remaining half. The $P-GPS$ discipline would then service ten packets of the "heavy" flow, followed by ten services one for each "light" flow. This allocation is not smooth at all – it would be smooth if the $i-th$ service of the "heavy" flow, was followed by a service for the $G_{i \mod 10}$ flow. This is accomplished as we can see in *Fig. 6* , by WF^2Q.

The problem with $PGPS$ is that while for every time t, it will not lag behind GPS in the aggregate service of a flow for more than one maximum packet, it can be much in front. The trick of WF^2Q, is that it considers as candidates, only these flows that their HOL packets – defined in the packet system – have *started* receiving service under GPS, that is those flow's packets with virtual start time less or equal to the system's potential VT(*Fig. 6*). Since in a packet system flows packets, are served with maximum-full rate r – remember a packet is the unit of transmission –, it can happen that a flow's $i-th$ HOL packet, may finish service in $PGPS$, much earlier than in the fluid world would; so the next($i+1-th$) packet of this flow has not yet started receiving "fluid" service, when the next scheduling decision time in the packet system comes[19]. The $i+1-th$ packet, will have virtual start time greater then VT, since it would start service in the fluid system, at point when the fair service received by all other active flows is greater than what is now. Returning to *Fig. 6* , the second packet of the "heavy" flow $F1$, has not started being serviced under GPS at time_slot 1(the time that the second scheduling decision must be taken), since its first packet is still being served – each $F1$'s packet needs two real time time_slots to finish service in the GPS model. So an other flow is selected.

The good news are, that the server remains work-conserving, though it restricts the number of candidates flows. Here we will give a simple intuitive proof. That the server remains work-conserving means that if there exists at least one flow in the set of active flows of the packet system, then there will be a flow with virtual start time less-equal to virtual time. If this was not the case, then *all* active flows in the packet system, would have start times *greater* than the virtual time. Since virtual time represents the rate with which active flows in the virtual system, receive normalized service, that all active flows in the packet system have greater starting virtual time than VT implies that, *all* the previous packets of these flows where serviced earlier in a packet system than in the GPS – which is still occupied serving them all. This means that the packet system(WF^2Q), passed less average

[19]this holds for all cases where more than one flows share the resource

delay than a work-conserving discipline(GPS), which is inept in the view of the conservation law.

3.1.6 WFI, an allocations smoothness metric

The name WF^2Q, is an acronym for Worst-case Fair Weight Fair Queuing. Worst-case fair is discipline with low WFI(worst-case fair index). This metric intuitively tells us, the maximum amount of time that a flow may have to wait, to receive its (next)fair service, in the worst case.

Definition 3.1 *A service discipline s is called worst-case fair for flow i, if for any time t, the delay of a packet arriving at t, is bounded above by $\frac{1}{r_i} \cdot Q_{i,s}(t) + C_{i,s}$, i.e*

$$d_{i,s}^k \leq a_i^k + \frac{Q_{i,s}(a^k)}{r_{i,s}} + C_{i,s}$$

where r_i is the throughput guarantee to flow i, $Q_{i,s}(a_i^k)$ is the queue size of flow i at time a_i^k and $C_{i,s}$ is a constant, independent of the queues of other flows sharing the multiplexer. A service discipline is called worst-case fair, if it is worst case fair for all flows.

Note first, that GPS has a WFI equal to zero. It is proven in [BenhZhang-WFWF] that $PGPS$ has a WFI equal to $N \cdot \frac{L_{max}}{r}$ – we show this case in the example above where the $11-th$ packet waited N other packets – and that WF^2Q has a WFI, $C_{i,WF^2Q} = \frac{L_i^{max}}{r_i} - \frac{L_i^{max}}{r} + \frac{L^{max}}{r}$. For uniform size packets this reduces to $\frac{1}{r_i}$. Round Robin, has a WFI equal for all flows, equal to $\frac{1}{N}$(optimal), while virtual clock has infinite WFI[20]. It appears that WF^2Q, is the best generalization of RR.

In addition to the fair properties of a smooth schedule, WFI is an important metric if we build hierarchical GPS schedulers[HiGPS][21] and helpful in bandwidth estimation techniques used in congestion avoidance schemes – if we do not want the estimations that we get, to oscillate between peak and zero bandwidth([BenhZhangWFWF]). In general smooth traffic is desired, for the regular operation of networks. On the contrary, bursty traffic poses problems to most of the switching systems that we are aware of.

In general the following equation hold, relating the discrepancies of the packet servers that we have described, compared to the fluid, GPS model[22]:

[20] A flow utilizing excess bandwidth, may be delayed later, in proportion to the excess service received

[21] In this scheme, flows queues are logically in the leafs of a tree and the resource is in the root. At each level, servers decide the service analogy of child nodes-flows/aggregations. So service is distributed according to the hierarchy. If a server(s), in a high level of the hierarchy, serves an aggregation of flows(a) in a burst probably exceeding temporarily a's fair service, if later an urgent flow that belongs to the same aggregation a becomes active, then this might encounter high delays while the server s obliterates the excess service by this aggregation previously. It is proved in [HiGPS], that in order to assign delay bounds in hierarchical GPS schemes, all the servers above level one must have low WFI. If additionally level one servers, have low WFI then, the whole discipline will have low WFI.

[22] $S_f^S(0,t)$ stands for the service provided by server s to flow f, in the time period $(0,t)$ where f, is continously backlogged. For simplicity, we assume here uniform-unit size of packets(1). For more complete description of these bounds, see [BenhZhangWFWF], [ParGalPGPS].

$$
\begin{gathered}
S_f^{GPS}(0,t) - 1 \leq S_f^{WF^2Q}(0,t) \leq S_f^{GPS}(0,t) + 1 \\
S_f^{GPS}(0,t) - 1 \leq S_f^{PGPS}(0,t) \leq S_f^{GPS}(0,t) + O(N) \\
S_f^{GPS}(0,t) - O(N) \leq S_f^{WRR}(0,t) \leq S_f^{GPS}(0,t) + O(N) \\
S_f^{GPS}(0,t) - O(N) \leq S_f^{Vclock}(0,t) \leq S_f^{GPS}(0,t) + 1/0 \text{ !!} \\
S_f^{GPS}(0,t) - 1 \leq S_f^{RR}(0,t) \leq S_f^{GPS}(0,t) + 1 \\
S_f^{GPS}(0,t) - 1 \leq W_f^{WF^2Q+}(0,t) \leq S_f^{GPS}(0,t) + 1
\end{gathered}
.
$$

3.1.7 Complexity

Most of the algorithms that we have considered require per-flow logical queuing(distinct $FIFO$ per flow) and a priority queue data-structure(or engine), that finds the minimum(or maximum) value out of a set with cardinality N – equal to the maximum number of eligible flows – including LRU, LQF, OQF and $FCFS$ if per-flow queuing is used. WRR – remember that we refer to an WFQ variant – has only this bottleneck where other, like $PGPS$, experience delays in the computation of virtual time with complexity $O(N)$ and operation on virtual events(active/inactive in the fluid model). WF^2Q+ solves this problem with complexity $O(log(N))$, without sacrificing the nice properties of WF^2Q[HierFairQ]. You can find implementation schemes for WF^2Q in [Stephens99] and for WRR in [IoanKateHeap].

The find minimum set operation can be computed in $O(1)$ time, if a priority heap is used but with complexity of insertion and deletions in the order of $O(logN)$, or in $O(log(N))$ time, if a tree of comparators is used for computing the minimum on the fly[HarteKatevCompTree]. This implementation is primarily useful in cases where the set of eligible flows changes dramatically from time to time – like in WF^2Q – so $N \cdot \log(N)$ worst-case operations count is prohibit. Other implementation use calendar queues, which are better suited for frame based scheduling.

In section 9 , you can find execution plots of the progress of virtual time functions, for WF^2Q+ and WRR and some additional comments on their relative complexity and accuracy.

3.1.8 Applications

Application of one resource servers distributing service to competing flows, can be found anywhere in nature, society or engineering. Here in packet networks and more specifically switches, the most prominent application, is on the outputs of an output buffered switch, where their role is to statistically multiplex incoming traffic. With a pure output queuing architecture, all the amount of traffic that arrives at some point in the inputs, can be forwarded in the respective output. This architecture is shown schematically in *Fig. 7* . In the worst-case, all inputs will send to the same output. If we use per-flow queuing in the outputs, so that not only the HOL of the $FIFO$ is eligible, a server is needed to pick choose which input, or flow in general, will be served next. Output buffered switches need to

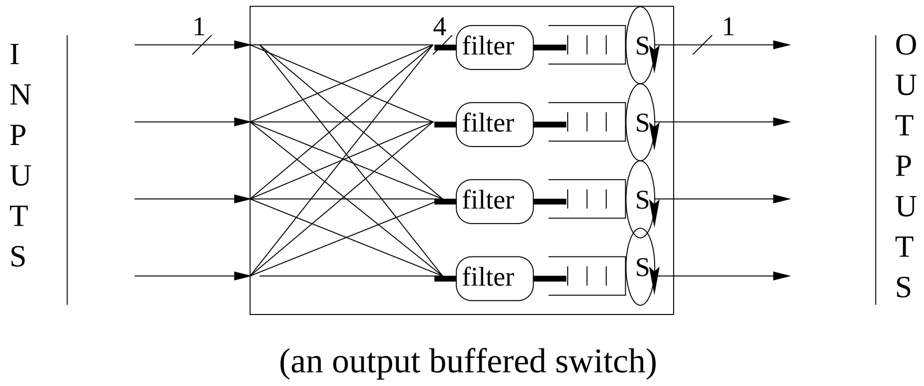
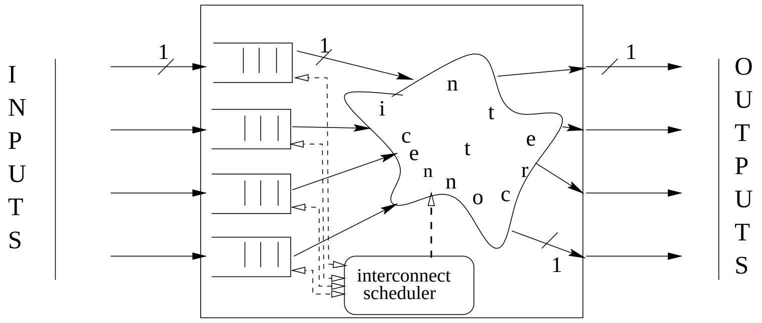


Figure 7: the architecture of an output buffered switch, with one resource-servers at each outputs. Although a single buffer is shown, per input or per flow distinct logical queues can be used, at each output. These buffers will need operating throughput $N+1$ times the throughput of the external links. The filter – that needs to provide again throughput N – forwards to the output buffer only these packets that are destinated for this output.

be occupied with memory at each output. This memory must be able, during any time interval, to receive("push") all traffic stemming from all inputs and additionally to "pop" traffic to the output link. So the memory throughput required is at least $(N+1)\cdot T$, where T is the throughput of the external links. High throughput buffers can be implemented, either through very wide memories or with very fast memory modules. It is not the aim of this master, to prove that these fast buffers cannot be made arbitrary large[23]. Furthermore, control logic must run at speed proportional to N, or we must use multiple(N) hardware components to sustain parallelism. Finally one must account and for the power consumption that is increasing with speed-pins-parallelism. All these arguments advocate, that pure output queuing, although ideal for scheduling, is not acceptable for packet networks in the general case.

[23]We need large buffers to store packets and not drop them at the "first bad occasion". Large means implementation off-chip(separate from the logic-control) while regularly large implies slow access time. So wide memories should be considered, but what about the pins?

3.2 Crossbar scheduling



(a generic input buffered switch with central scheduler)

Figure 8: A generic input buffered switch, with central scheduler. Inputs send their state(with dashed lines) to the scheduler and he decides which will send and where. So he replies to the inputs and configures the crossbar(thick dashed lines) through control signals. The buffers in the input need to have twice the speed of the external links(one write and one read of a unit of traffic, in the time interval that a unit of traffic arrives to or departs from the switch. Input buffers need no be a single $FIFO$ in general.

As we show in the last paragraph of the previous section, output buffered switches do not scale well and additionally need expensive in hardware-economic terms components. Input buffered switches, that is switches that store traffic at the inputs, need have memory throughput at each input, equal to two times the links throughput(one write,one read). An interconnect with limited capacity, is then scheduled to forward cells from the input buffers to the output links – during one time instance, one at most input can send a unit of traffic to a unique output and an input can only send one unit of traffic[24](see *Fig. 8*). Different interconnecting-topologies, have different "capacity" properties, different forwarding peculiarities-capabilities, and by these means, need different schedulers. Here we discuss input buffered switches with the crossbar interconnect, and the related scheduling-forwarding problem.

3.2.1 The problem

The crossbar topology is probably the most widespread switching fabric considered in the research community of networking and additionally this architecture that has been adopted by most commercial vendors, building switches. This does not imply a natural superiority against other topologies. On the contrary, crossbar is considered to be highly expensive and inefficient, since theoretically there are switching fabrics with similar performance[25] and much lower cost than $O(N^2)$. The acceptance of the crossbar topology in research, is to be attributed to its

[24]We consider the basic view with no internal speed-up, or small output buffering

[25]The crossbar is strictly-non-blocking(means that can schedule any inputs to outputs permutation, without having to be reconfigured to switch from a given permutation to an other(strictly)). A non-

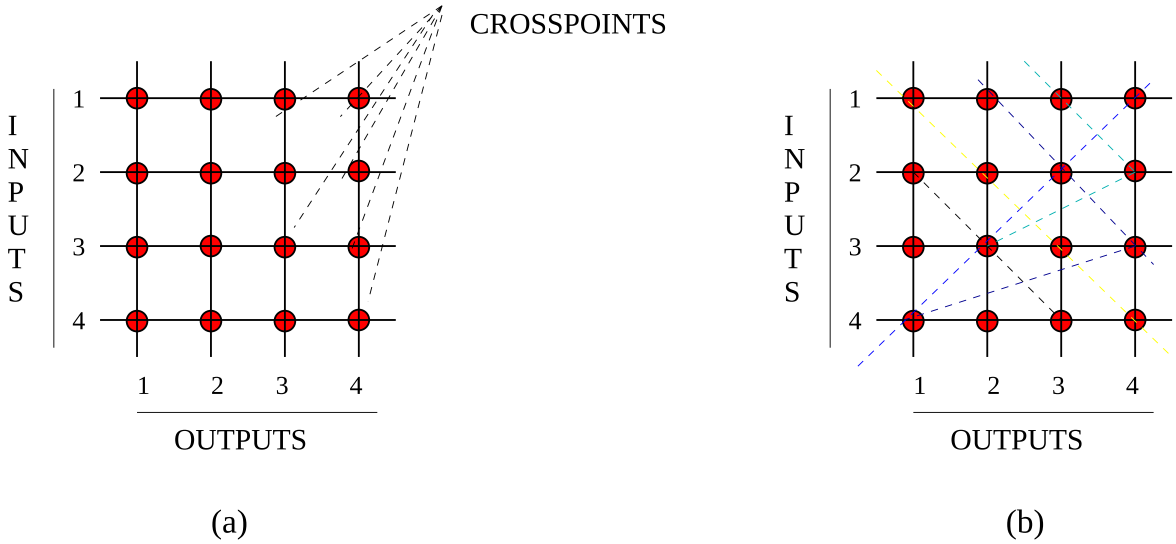


Figure 9: The crossbar topology is shown in(a). There is one crosspoint for each input-output pair. Each crosspoint may have one out of two states, either active(on) connecting the input 2 the output, or inactive(off) disabling contact. On each row(input-horizontal line), one at most crosspoint can be on and the same for each column(output-vertical line). The crossbar scheduling problem, is to configure the crosspoints based on the requests from inputs for outputs, so that the desired packets are forwarded. Deciding to transmit one packet, delimits the available choices for the other packets that will complete this time_slot schedule, since we have to exclude flows using the same input or output. In figure (b) we see different with different colors, different configuration that belong to the feasible scheduling area. The yellow and black dashed lines, are diagonals of the crossbar that belong to the feasible area, since the set of diagonals is a subset of the permutations set.

conceptual simplicity, that facilitates the development of clever forwarding and scheduling techniques. In addition crossbar is a quite modular and simple design, appropriate for $VLSI$ implementation.

Crossbar scheduling is the problem of finding a set of active input-output pairs, that are non conflicting under the crossbar topology – probably with the addition of objective-based desired characteristics. If we put aside any additional objective constrains and consider only the feasible scheduling area, conflicts in a crossbar occur either in the inputs or in the outputs, but not somewhere else internally. So the crossbar scheduling feasible area SE, includes all the sets S_i of distinct connection c_i, from inputs I_{c_i} to output O_{c_i}, that comply with the requirement that $\forall c_i, c_j \in S_i : I_{c_i} \neq I_{c_j} \cap O_{c_i} \neq O_{c_j}$. It follows that the feasible scheduling area for a crossbar is simply the set of all permutations, from all subsets of inputs

blocking switching fabric need to be able to configure/transmit $N!$ different permutations, so it should have at least that number of states. Since a crossbar has N^2 crosspoints, it seems that the crossbar has adequate states(2^{N^2}) to fulfill the non-blocking demand – this does not directly entail that it actually does, but it is simple to prove that this is the case. The minimum number of switching point that can theoretically fulfill the non-blocking idiom, is M where $2^M = N! \Rightarrow M = \log_2(N!) \Rightarrow M \simeq N\log_2(N)$.

to outputs *Fig. 9* . When only a subset of connections are active, the feasible area is additionally constrained. In terms of graph theory the problem of crossbar scheduling is reduced in finding a match in a bipartite graph.

3.2.2 Iterative algorithms with no differentiation on the nature of flows

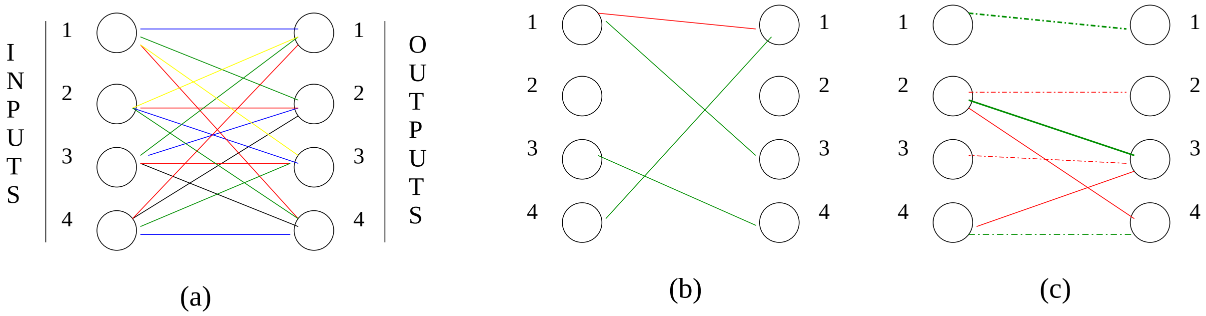


Figure 10: The crossbar scheduling problem defined in graph theory terms. Bipartite is a graph, with edges only from one set of nodes to an other and with no edges among nodes of the same set. Bipartite match is a set of edges in a bipartite graph, where no node participates more than one time. In (a), we can see different matches, assuming all edges are active with different colors. In (b) not all edges are active. The green match has maximum size (3), while any other match that would include the red color would have less size. If a maximum match discipline is followed, then the red colored edge(flow) will not be selected and will be starved. In (c) we see with green think lines, a maximal match under construction. The other green edge, can be included in the maximal match, since no other edge in the match so far needs to change. The red lines cannot be included. The dash-dot lines represent the maximum match, that has size 4, while the maximal match that is being constructed in the figure with the green lines, can only have size 3 – since it is dictated by actions-choices taken previously in the construction of the maximal match.

Different schedule instances of a crossbar differ in size. Former objectives in literature, were concerned only with the creation of schedules of big size, since increasing the size of each schedule, increases saturation throughput and reduces average delay. In this view Karol was the first to publish a paper, exclaiming the need of input-buffering – multiple per output queues at the inputs-known and as virtual output queuing –, instead of input queuing – one logical $FIFO$ at each input – to increase the scheduling area of a crossbar and potentially increase the switch throughput, since he proved that a single logical $FIFO$ in the input, that keeps all packets despite their destination, can confine throughput to $2-\sqrt{2}$ when $N \rightarrow \infty$. This is because the feasible scheduling area is additionally constrained to contain one edge per input, the one formed by the head packet in the respective $FIFO$ and the problem called HeadOfLine blocking appears[26]. We need to consider

[26]The cell in the front of the $FIFO$ queue, may be blocked by the scheduler(someone else is sending to its desired output) and so other cells behind this HOL packet cannot be scheduled, although nobody

many choices for each input to increase throughput. For example if a maximum match algorithm is considered, then the switch will sustain throughput equal to that of an output queuing switch. This approach has two disadvantages. For one computing maximum size matches in a bipartite graph has a complexity near to $O(N \cdot (N+M))$, where N is the number of nodes and M the number of edges[27], but even if we put aside this technology obstacle, this discipline may lead to starvation these flows that do not belong in the maximum match – maximum match discipline is very rigid *Fig. 10* .

The next critical date in crossbar scheduling was when Anderson proposed Probabilistic Iterative Matching(PIM) in [AnderPIM], a decentralized iterative scheduler for crossbars with input-buffering. In the same paper, Anderson proposed the architectural position to decouple packets forwarding(configuring the switching fabric) from packets scheduling(selecting which flow to send), a stand that requires the manipulation of fixed size packets internally to the switch and is usually taken for granted nowadays. PIM computes maximal(opposed to maximum) matches through an iterative, distributed procedure. Maximal matches are defined as these matches that cannot be expanded without re-configuring connections included so far in the match. There is a result, that claims that a maximal match cannot contain less than half of the connections of an maximum match, while simulation results of PIM showed that in most cases throughput near one can be achieved. The distributed iterative algorithm PIM computes matches as follows:

ITERATE

1. *All inputs send request to all outputs for which they have queued cells (input-buffering systems have this knowledge)*
2. *All outputs gather the requests sent for them, select one using a uniform random choice and pass a grant to the selected input*
3. *All inputs that receive at least one grant, accept one using a uniform random choice. Each accept results into a new connection being added into the crossbar configuration.*

In PIM is proved that after the $i-th$ iteration of the algorithm, on average 3/4 of the connections that were available at the beginning of $i-th$ iteration, are either included in the match or excluded out of the set of candidates flows(because one of their competitors is included), so the algorithm will have an average time complexity of $\log(N)$. In addition, no request(input/output session) will be starved, since grants and accepts are selected randomly. With one iteration, the average

else has to send to their desired outputs. Besides efficiency loss that happens in this case, even if the HOL can be sent – and is sent –, there is is the problem of reducing the sophisticated scheduling area.

[27] Randomized algorithms can take on average $O(N+M)$ to find a maximum match

occupancy achieved is near $(1 - \frac{1}{N})^N$ [28], but hopefully this number increases when more iterations are used – reaching near 95% with four iterations under Bernoulli generated, uniform destinated arrivals(look simulation results in [AnderPIM],[SLIPMcKeownAndersComp]). In the paper, they propose the use of four iterations, considering trade-off between complexity and performance. Even with four iterations, PIM is not always fair[SLIPMcKeownAndersComp], since it tends to prefer connection with fewer competitors in the inputs.

$i-SLIP$ which is a very similar scheme, has actually boost PIM performance by replacing random selections with the round robin discipline([SLIPMcKeownAndersComp]). This makes possible(it appeared to be so) the introduction of the *desynchronizing* characteristic of $i-SLIP$, on which it owes its efficiency, with a single operation. The desynchronizing features, makes sure that in high congestion cases(i.e. when all inputs have many packets to send to all outputs), matches will increase in size(*"slip"*) and after reaching a maximum match, a new maximum, but completely different, match will be computed in the next scheduling cycle, even with one iteration of $i-SLIP$.

ITERATE

1. *All inputs send request to all outputs for which, they queued cells(input buffering systems have this knowledge)*
2. *All outputs gather the requests sent to them during this iteration, and select one the using round robin, implemented with a next to serve pointer and any, constant enumeration of inputs. The next to serve pointer to, is updated to the next(module arithmetic) input, only if the grant is accepted by the respective input in the next step of the current iteration.*
3. *All inputs that receive at least one grant, accept one using in round robin fashion . Each accept results into a new connection being added into the crossbar configuration.*

To understand this, consider that a complete match is computed during the $i-th$ scheduling cycle. Then all outputs will update their next_to_grant pointers to a *different* input, with respect to the previous scheduling cycle, and to *different* input, with respect to the updated next_to_grant pointer of all other outputs[29]. This argument explains why $i-SLIP$ behaves so well, even with one iteration; the algorithm will tend to "slip" to maximum and different matches. The term different matches, implies fairness and actually $i-SLIP$ results in much more fair schedules than PIM, but in the case of a single iteration there have been

[28] This is the probability that no output will grant a particular input in the second step of the single iteration algorithm

[29] since all granted different inputs during scheduling cycle i and their grants were accepted – otherwise the previous match would not be complete – it follows by the rule in the second step, that all will be ready to grant a different input at time-slot $i+1$

found configurations where $i - SLIP$ is unfair [SLIPMcKeownAndersComp]. So more iteration can be used to sustain the fairness objective. In the same spirit, Serpanos proposes $FIRM$ ([SerpFIRM]), a variant of $i - SLIP$ that can achieve lower delay bounds[30]. Finally in [Serp2DRR], a method called Two Dimensional Round Robin for computing crossbar schedules is proposed. This is based on performing round robin upon primitive diagonals – are elements of the feasible scheduling area – and proves to produce fair matches, with properties similar to round robin scheduling in the single resource, contention case.

3.2.3 Coupling QoS, with the crossbar scheduling problem

So far the crossbar scheduling schemes that we have considered are primarily interested with the provision of high-throughput and a relative degree of fairness, mainly interpreted as non-starvation. In the same paper that PIM was proposed, a generalization called Statistical Matching(SM) was developed that promises to service sessions based on their relative weights[31]. With this approach the matching is initiated by the outputs and can happen that these pick some input which is inactive, resulting in poor performance. An other alternative by the name Weighted-PIM($WPIM$) described in [StilVarmWPIM], assigns to each flow(input-output pair), a number of quotas/reservations for a given frame and treats all flows equally, while all have quota. When a flow has no more quotas for the current frame, then this is excluded from the schedule creation procedure. $WPIM$ results in very bursty schedules and can provide discrimination in a coarse scale. An other proposed scheme by the name of Fair Iterative Matching(FIM) [FIM], is in our view the most prominent solution – in the class of discriminative-iterative schedulers. It utilizes GPS-related servers at the outputs, to grant the inputs, and the $FCFS$ scheduling discipline in the inputs, to accept the grants. Although this solution seems to carry the characteristics for differentiated fair allocation of bandwidth, this cannot sustain fair allocation in weighted max-min fairness sense, since this does not consider weighted-fairness in the inputs contention area. In addition the desynchronizing characteristic is no longer there – we will see why latter on in an other section(section 5.6) – and at least four iterations are needed for a 16x16 switch(remember $O(log(N))$), to reach a respectable throughput.

Stable Marriage Matchings Finally, a somehow differentiated approach is proposed in [Linear]. There, flows(edges) are assigned weights, based on various criteria to be discussed bellow, and the maximal matches are generalized to stable marriage matches. Stable marriage matches are again computed with an iterative procedure. In a stable marriage match, previously assigned edges ($i \to j$) can be broken, if an other presumptive "lover" of input i(or output j) l shows up, that

[30] actually proves delay bounds for $i - SLIP$ as well

[31] The uniform distributions of the random selections, in the outputs is replaced with non-uniform one, which gives advantage to flows relative to their weights

wants-values-weights i(or j respectively) more than his current "love", and i(or j) desires-weights l more than j(or i respectively). When this happens at some iteration of the algorithm, the previous marriages(matches) are broken($\rightarrow$ connections are excluded from the match) – since these were not stable – and a new edge is included. In this paper, weights are assigned according to various criteria, either stemming purely from QoS contracts(enables QoS with a credit based mechanism – i.e:you will have one credit for every three scheduling cycles) or with combinative methods that ensure fairness. A simple weight mechanism to use for differentiating edges, could simply be the number of accumulated credits. Since this may lead to starvation – flows may accumulate credits during their inactive periods, so when these suddenly become alive again, others will be starved – other hybrid schemes are considered. These either place an upper bound, on the number of accumulated cells, or add adaptive mechanisms on top of the credits for the computation of weights(i.e: weight of flow f, is equal to the number of time slots a validated(= has credits) cell of f, is weighting for service). In the same paper, an additional algorithm is proposed that could run after the QoS specific algorithm has completed, for distributing excess bandwidth to unsatisfied flows. These algorithms, that again compute stable marriage matches, closes max-min fairness in distributing the excess bandwidth – simulation results in [Linear] suggest, that the 85% of flows get 95% of their max-min fair share.

3.2.4 Speed-up and/or output/internal buffers–hybrid systems

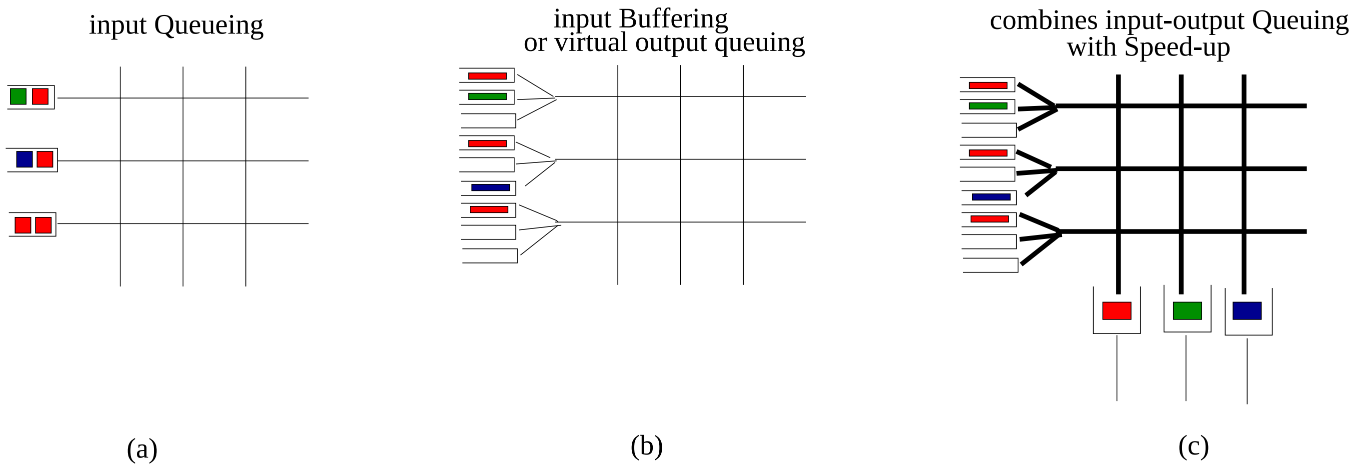


Figure 11: The alternatives solution for buffers organization on the crossbar topology. We have omitted one last alternative, to be presented under our proposed scheme.

Hybrid schemes, that combine both input and output queuing, have been recently considered under various ways in the context of crossbar scheduling. These schemes normally utilize internal speed-up[32], otherwise what could be the utility of an output buffers?.

[32]Fraction of the internal line rate to the rate of the external links

A first result is, that there exits a way to emulate any pure output queuing discipline, using an input buffering system with crossbar interconnect, but you have to pay the cost of speed-up two ([In2OutSpeedup]). Besides, the discipline that accomplishes this is quite complicated and expensive, so you must actually pay much more than the speed-up cost. In the same paper it is proven that with a simpler method and speed-up, necessary and sufficient, equal to $2 - \frac{1}{N}$ the $FIFO$ discipline of an output queuing architecture can be emulated. Other alternatives, like $LOOFA$ [LOOFA], promise to provide work-conserving crossbar scheduling[33] with a speed-up of two. This algorithm uses an iterative maximal match algorithm, that runs twice during each cell-segment time_slot, where the outputs with the smallest occupancy in their output buffers, are given higher priority.

Small internal buffering in combination with input buffering, has been considered since [Kate87], as a cheap solution for distributed scheduling for most switching fabrics. Only recently these ideas are taking some place in the research for switching fabrics. Internal/crosspoint buffers are proposed in [MagBufFullCap], while output buffering, crosspoint buffering and speed-up have been considered together in [DGPS]. Both proposals build upon the crossbar topology with virtual output queues. In the first ([MagBufFullCap]), it is prooven, through a fluid model analysis and big numbers laws properties for the arrival process, that the proposed system will accomplish 100% throughput, whereas in [DGPS],[DGPS1] QoS contracts and the distribution of excess bandwidth are examined upon two similar architectures. These architectures([MagBufFullCap],[DGPS]) reassemble ours, although theirs and our work were developed independently. We will describe our architecture in the following section, so we postpone arguments on their analysis for subsequent sections, after we have presented our results. Figure *Fig. 11* pictures the buffering alternatives that we have considered so far.

3.2.5 Complexity issues – examined fast

On the complexity issue, PIM related algorithms can be physically distributed, but require the communication of information – request/grant/select messages –,in order to compute a schedule. This information must be exchanged through wires connecting all inputs to all outputs. In addition iterations increase complexity and hardware-costs.[34]. $i - SLIP$ amends the high cost of peaking random variables in hardware, which is necessary for PIM and instead the round-robin disciplines for the grants/accepts selections, can be implemented through fast and cheap priority encoders as proposed in [35].

$FIRM$ has similar complexity with $i - SLIP$, while in [Serp2DRR] they propose hardware friendly methods for selecting the diagonals and excluding the flows

[33] An output will be occupied, whenever there is a cell destinated for this output in the system

[34] PIM with four iterations, has been incorporated into the architecture of the $AN2$, switch, and has been actually manufactured

[35] $i - SLIP$ has also been used in the implementation of the Tiny-Tera switch.

that cannot be selected after a diagonal has been incorporated into the match. Alternatives that generalize $i - SLIP$ and PIM, for producing weighted service allocations, require incremental cost analogous to the disciplines used for granting and accepting requests – but these will certainly require more than the one iteration. The stable marriage matching iterative algorithm [Linear], is quite expensive in our view – at least twice the cost of a simple maximal match, although in [Linear] they propose various scheme for amending these high costs – and in addition needs to run twice for distributing excess bandwidth.

Finally speed-up is the middle name of the cost(require memory/logic to run faster) and in our view should be used, only when it is necessary(i.e: in order to handle variable packets lengths as described in [DGPS]). Finally output(or cross-point) buffers are affordable in hardware-economic terms, if these are adequate small(technology depended). A flow control scheme is needed to prevent them from overflow. This will be part of the discussion in the next section, which describes the architecture that we propose and analyze.

Related to this discussion, is section 5.6 , that studies two expensive crossbar scheduling techniques.

4 Switch Architecture

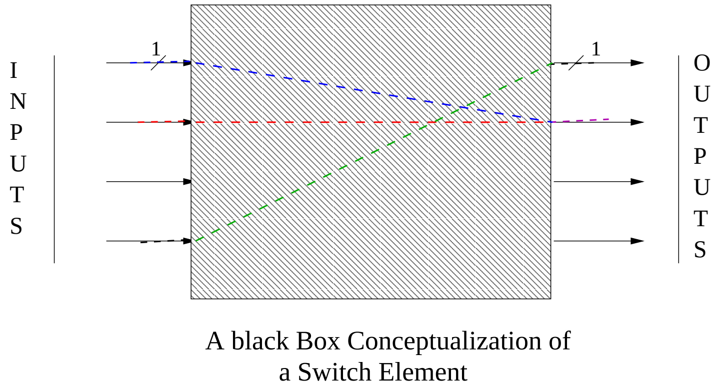


Figure 12: With a black box conceptualization of a switch, only flows destinated for the same output content. This can only be realized through the expensive and ideal output-queuing architecture

4.1 Queuing Architecture

.

4.1.1 Buffers positioning

Input buffered switches, provide the framework for building scalable switches. Large memory modules, positioned at each input-card of the switch, need only provide twice the throughput of the external links[36]. So the goal is to use input buffers as the main packet repository, where medium term congestion will be handled without dropping packets. Pure input queuing(keeping a single logical $FIFO$ at each input for all cells) suffers from the well known HOL, that limits the maximum switch throughput roughly to 2-$\sqrt{2}$ when N – the number of inputs – grows. So the alternative of using input-buffering[37] – that is per flow logical buffers at each input – is the only viable solution. The most primitive form of advanced input queuing, is when the number of logical buffers at each input is equal to the number of the switch outputs. At the other end, per micro-flow queuing may be deployed in inputs, but here we consider the basic-working scheme.

Input buffering introduces one more scheduling problem in the hart of a switch architecture. Typically one would think that traffic contention occurs where traffic from different sources is being multiplexed. So a typical black box conceptualization of switch would reveal, that contention occurs only at the switch outputs, which will be eventually shared by traffic from the switch inputs *Fig. 12* . This

[36]In the case where speed-up is necessary or introduced to launch performance, memory throughput at inputs must be a constant multiple of the links throughput

[37]or advanced input queuing, or virtual output queuing

is really, the nice feature of the output(or shared) queuing family, that comes in the expense of high throughput memory modules(need to have throughput which grows with N). Input queuing stands on that it cannot transfer all potential traffic to the switch output links(or output buffers), during one time slot. So it holds excess traffic in the inputs large repositories. A scheduling decision is needed, to determine which packet – among all packets kept in one input – will be forwarded through the interconnect to its destination. As we have seen this decision cannot be local, but has to be taken inter-dependently among all inputs, with concern of the interconnect peculiarities, usually in the aim of sustaining the switch throughput high.

Even if the interconnect is non-blocking[38] finding a set of not-conflict requests is not that easy since: 1) aggregate traffic from the inputs does not arrive in permutations, 2) the match problem can be difficult and the algorithm for computing the solution should be either central, or if it is distributed communication must be exchanged multiple times in a scheduling period. But even more problematic is the fact even in the case of such a scheduler, scheduling is inevitably drown upon the demand of high throughput, greatly reducing the sophisticated-scheduling-area. Coupling sophisticated-scheduling with the high switch throughput demand, is a difficult problem as we have seen ([WPIM],[FIM],[Linear]).

Decoupling means that the sophisticated-scheduling decision will be taken independently of the demand for high aggregate switch throughput: *"If I want to serve this flow(sophisticated) - even though I have cells for other output - I will do so"* as much independently of *others* choices or obligations as possible. It turns up, that sophisticated scheduling must be done independently at all contention points. Contention points are at least the switch inputs and certainly the switch outputs. If inputs take independent decisions, then packets cannot go far and occupy resources that are shared among many inputs, since then collision can occur. Just before the point of using such a resource, an other scheduling decision has to be made, in order to choose which among all candidate inputs, will use this resource. Packets that are not selected in this second level decision must be held somewhere safely and wait for their turn. So internal buffers/memory(access buffers) is the solution to distributed and independent scheduling. This organization can easily be generalized for an arbitrary number of switch stages.

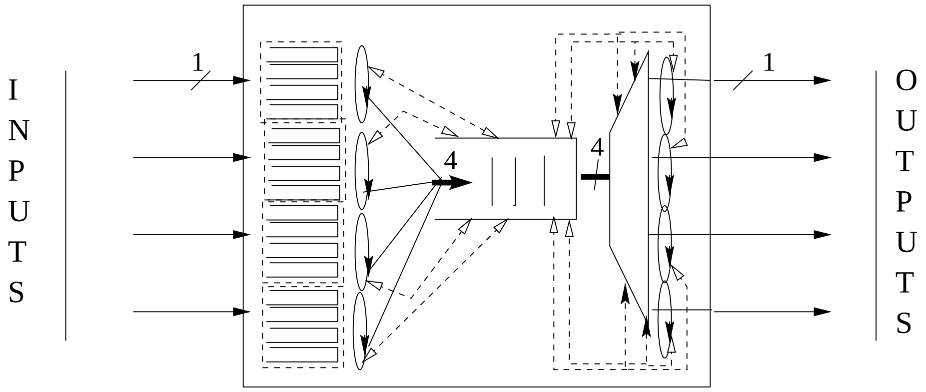
A point here is that we do not expect from these buffers to be huge, since their role is to resolve the short-term internal contention and not the congestion in large scales[Kate87]. Input buffers are maintained for the latter. So these small buffers[39] that are logically present at each crosspoint, can safely be implemented on chip. Of course a flow control scheme is needed to prevent these buffers from overflow. A loss-less scheme like the backpressure, credit-based or on-off flow control is appropriate. As we shall see the buffer management scheme, will also play an

[38]In general means that can forward any set of request as long as these do not over-load the outputs

[39]Can be found and by the felicitous nick-name *access resolution buffers*

important role into the scheduling characteristics of the system.

4.1.2 Flow control and selective backpressure



(an input buffered switch, with small internal buffering and distributed scheduling)

Figure 13: An input-buffering, internal buffering switch with distributed scheduling. With dashed lines we see the information that must be exchanges between buffers and the state of the internal buffer space, in the view of flow control and scheduling. Each output server, must know which cells in the internal buffer are destinated for the respective output, in order to take a scheduling decision. Each input must now if there exists buffer space in the internal buffer, adequate to receive its candidate packet.

So far the queuing architecture that we propose – with scheduling considerations – is something like that in *Fig. 13* . A final but critical point with the queuing architecture is the organization of the backpressure scheme. As we implied so far, each input can safely send packets on the crosspoints buffers, as long as these buffers do not overflow. Under the view of protection among inputs, it is critical that a mis-behaving(or simply greedy while other are inactive) input will not occupy these buffers, keeping the back-pressure held for all other inputs that may decide latter to transmit. These inputs will have less chances to get their fair share on the output links. Instead the backpressure signal must be send to this misbehaved input that is congesting the output links. So internal buffers must keep separate space(statically partitioned) for each input. In addition considering one input, if the requests from this input to an output fills-up its input-dedicated buffer – for instance because this output is highly occupied –,the backpressure will prevent future packets of the same input to other outputs from being served. The latter case is inherently similar to the HOL problem as shown in *Fig. 14* . So in order to guarantee fairness/protection/isolation and efficienct, distinct per input *and* per output buffer space and backpressure signals should be used, a scheme which is known under the name of selective backpressure.

Selective backpressure makes sure that every competitor will have a different view of the congestion at the same link and will be given equal possibility of getting

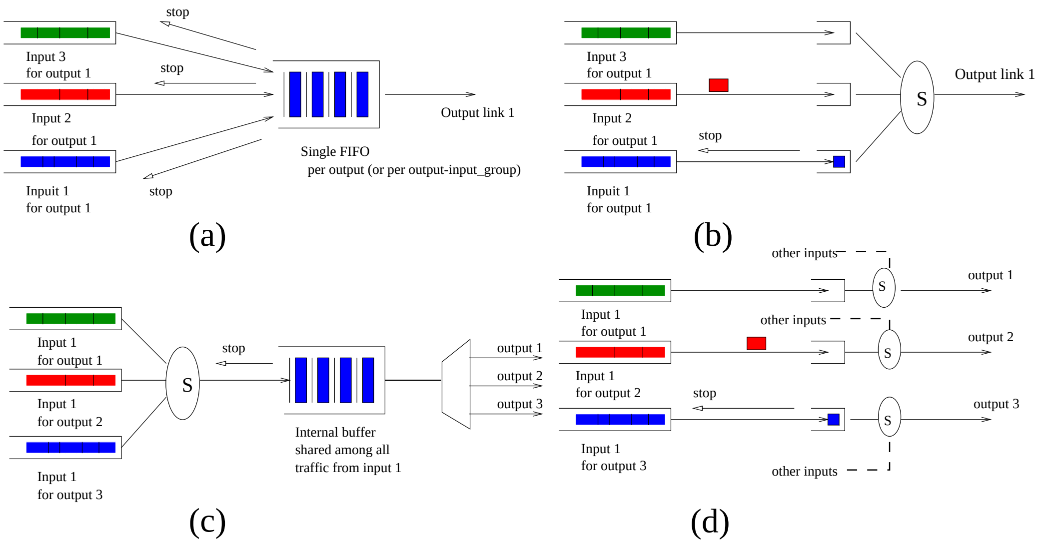
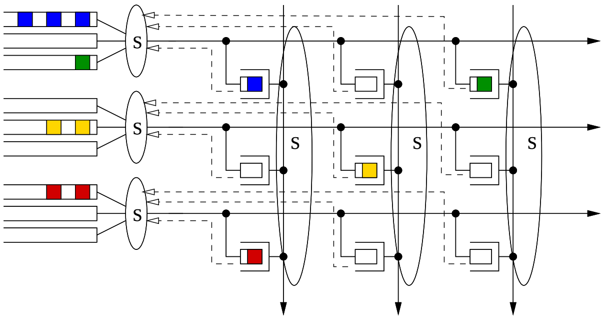


Figure 14: In case (a) three inputs send to the same output, but these inputs share the internal buffer space. In case (b) a distinct per input buffer is used among inputs, that solves the unfairness seen in (a). In (c) all flows from one input, share the same buffer (as in (b)). In (d) a distinct per input-output pair buffer solves the HeadOfLine present at (c)

service no matter of others requestors previous histories. That is, selective backpressure is asserted for one flow, only when this flows rate of service is less than the flow's arrival – in other words provides isolation. On the other hand indiscriminate flow control does not distinguish between well behaved and misbehaved sources and is a bad architectural practice. Even if we do not accept the characterization of mis-behaved – after all there is no contract –, indiscriminate backpressure is not fair. In *Fig. 14* in case (a), where all inputs share the same internal buffer, if input 3 fills up the buffer when other inputs have no traffic, then all inputs will be prevented from sending traffic, although not all are responsible for the congestion. This certainly results in unfairness. In (c) all flows from the same input share the same buffer. If during some interval the flow directed to output 3 has traffic, but the others do not, the buffer will fill-up. Until flows 3 backlog is resolved – an event that relates to the rate that flow 3 is receiving from output 3 – all other flows in input one, will be prevented to sent traffic, even in the case where the respective outputs are free. This is a classical HOL problem, that is prevented with selective backpressure and per-flow queuing.

A final argument for selective backpressure, is that if indiscriminate and common backpressure is used for all inputs, when the available/common buffer space can only accommodate a couple of packets, inputs servers will not be able to take their scheduling decisions independently of other inputs – if all inputs sent, packets will be lost. Partitioned per input buffer space, makes this possible – each input server only needs to *independently* consider its own backpressure signal(s) – and highly reduces complexity. An argument against partitioned buffer space, is probably space under-utilization when not all flows are active. But again since internal buffer space is not expected to be that huge, in our view this is not even a point – why should a flow occupy internal buffer space, when others are absent? It makes some sense though, only if congestion occurs in a single input, in which case you could say that you want this input to occupy all the internal buffer space. If this is your view also, then you will have to sacrifice the independent scheduling and in addition pay the (temporal?)unfairness in service when other inputs will become active.

4.1.3 An input buffered crosspoint buffered crossbar – putting it (almost)all together – the system operation.



The input buffered crosspoint buffered crossbar architecture
with selective backpressure

Figure 15: The input buffered crossbar with internal buffering & crosspoint buffering. Note that actually we did not have to make all the arguments in support of selective backpressure, since the crosspoint buffer naturally imply it.

So far our scheduling consideration when designing the buffers architecture and the flow control scheme, perfectly map on the crossbar architecture; see *Fig. 15* . If we put the contention resolution buffers on each crosspoint, we have realized the per-flow queuing and the selective backpressure that promises fair and discriminative scheduling. Each buffers at the crosspoints needs to run at twice the speed of the input links, so we do not have to worry about broad or fast and expensive memory modules[40]. Distinct servers are needed on all inputs/outputs which will operate independently, without any global synchronization or dependency. An input server will be able to send traffic to any output as long as the buffer at the crosspoint connecting this input to the respective output, has not been filled. This information is conveyed to the input through the selective backpressure scheme. An output server, will be able to select any input, as long as this input has some traffic "pushed" in the crosspoint buffers.

[40]The same architecture can map in a shared buffered system. These two alternatives are logically identical, but with the crossbar the crosspoint buffers can be implemented on the same chip with the crosspoints, while the shared buffered system has the disadvantage that someone must schedule the accesses to the shared medium, a problem similar to crossbar scheduling.

The crossbar is an excellent architecture in terms of performance, with or without internal buffers. Each packet passes only through one crosspoint(switching point) in its journey from input to output. What internal(or crosspoint) buffers merely add, is the simplification of scheduling and the decoupling to two different problems that have to be considered inter-dependently in the non-buffered case; crossbar scheduling and quality of service. So far we have examined the queuing architecture of our system and only its implications on scheduling. We will now include in this analysis the fundamental scheduling decisions.

4.2 Scheduling on contention point

.

4.2.1 Weights

At this point we must decide the class of schedulers that we will place on the contention points. Contention in the buffered crossbar occurs at the inputs, where multiple connection directed to different outputs content for the horizontal link that directs to the crosspoint buffers and at the outputs, where multiple input links place their requests- are expressed by the presence of packets in the crosspoint buffers-to use the output link. Our assumption is that every distinct input-output pair has an associated weight that represents its relative urgency of service. This is the only assumption that we make for now, on the nature of weights. Our intention is to examine the scheduling characteristics of this architecture with no additional hypothesis on the use of weights. So we will ignore here any implication of these weights on the characterization of the candidate flows and examine the general scheme, in the presence of an random weight, for each input-output pair. Of course we keep whenever necessary, the hypothesis of relative service, in proportion to weights.

4.2.2 Ideal service

An ideal scheduling architecture, would be if each input i will receive service from output j, in proportion to the weight of this flow $f_{i,j}$ with regard to all other active flows using the same output link. For all input-patterns, this can only be accomplished with pure output queuing and GPS related servers at the outputs, since this the only architecture with contention points only at the outputs.[41]. Our architecture has many contention points – ++the crossbar inputs – and we can demonstrate that in the absence of speed up, there exists an arrival pattern, where

[41]It has been shown that with input buffering and internal speed-up of 2, we can emulate any output queuing discipline – with extremely sophisticated crossbar scheduling methods for hardware implementation, in our days

output queuing cannot be emulated with no service discipline. So considering our switch topology we reform the scheduling objective, in something that can be realized by our architecture.

4.2.3 Ideal with topology constrains

Recalling our ultimate objective, which is to to create a GPS like switch that will allocate fair service to flows – which for our view of the moment are distinct input-output pairs – the solution is to provide whenever possible, fair share on the contention points by placing a distinct-packet-GPS-work-conserving-server at each. So far we have equipped our topology with $2 \cdot N$ GPS related servers.

It follows that in the case of arbitrary weights, a flow might receive different service rate from the input and different service rate from the contention point at the output, since the sum of weights in the input and the output servers need no be the same. Of course this cannot continue in the long run – in the absence of infinite buffers at the crosspoints. The target is that the selective backpressure flow control scheme, will equalize these two rates in the medium/long run, so that every flow will receive at least the minimum of them. This means that some bandwidth might be left over at the less congested resource – congested with respect to the demand of the other flows and their weight. This excess bandwidth will be re-distributed to the other flows that use this less congested resource, since our schedulers are work conserving. You might have already started finding similarities with the weighted max-min fair allocation, and otherwise you should.

For example in *Fig. 16* flow with id 1 will receive at least the minimum of its guaranteed shares, which is 1/3 in the input server and 1/5 in the server output. So the input server will find at several times, when it is flow's 1 turn to be served, that this flow is not-active(due to backpressure). Since the server is work conserving, he/she will try to find an other flow to service in the place of flow 1. So if all work ideally, flow with id 3, will get more than its guaranteed rate from the output server and will reach the long term rate 4/5 – since this is not congested in the input. Similarly in the scenario of case (b), flow 1 will end-up with rate 1/5 and flow 2 with rate 4/5. Of course all these hold in an ideal fluid model, where flows instantaneously adapt to backpressure and servers serve in infinitesimal units. In this case we can prove that the rate allocation to the fluid sources, will be equal to the weighted max-min fair. Then, the abbreviation $WRFC$(= WeightedRoutes-FairCrossbar) that we use in some points later in this document, to refer to the generic system that we have just described, cames natural; this simply follows the multiresource-scheduling-discipline that this system is able to enforce.

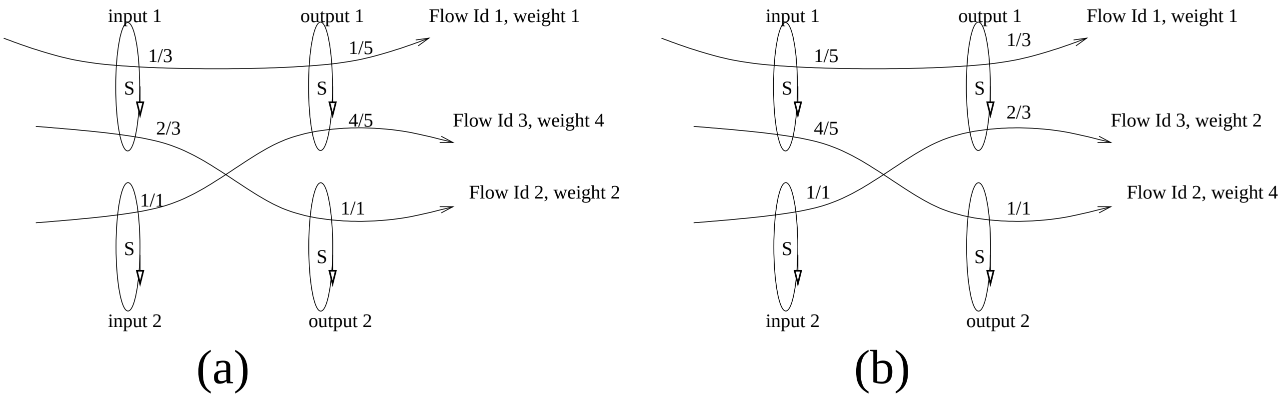


Figure 16: Aspirated allocation examples with the fair schedulers in the buffered crossbar architecture. Flows are considered continuously backlogged.

4.2.4 Fluid model analysis

It should be clear that if the traffic is fluid, then the proposed architecture will reach something close to weighted max-min fairness. What follows in this section is a prove, that this is actually the case for our crossbar topology system, when flows are continuously backlogged at the input buffers(active or inactive). Anyway this was the argument of Helen in [HaydenMaxMin], when she tried to prove that the combination of round robin schedulers with backpressure flow control, will eventually allocate max-min fair rates to the flows. Of course her analysis regarded arbitrary networks and several hypothesis of demand – pushing her to end up with very loose performance bounds and only under very demanding hypothesis(backpressure thresholds requirements).

Theorem 4.1 *In a fluid model of continuously backlogged sources using the crossbar architecture, with GPS servers and instantaneous rate adaptation, the final rates of the fluids sources will be exactly equal to those assigned by the weighted max-min fair discipline.*

Proof
The proof will be primarily rhetorical. Consider a resource r(input or output) and denote as F_r the set of flows using this resource. Also for every flow $f \in F_r$ mark as $\overline{r_f}$ the complementary resource that is being used by f. That is if r is an input and a flow f uses r, then $\overline{r_f}$ will be the output used by f.

Then if a flow $f \in F_r$ with weight w_f is continuously backlogged during any time interval T, then from the GPS definition it follows that this flow is guaranteed to receive service from this server at least equal to this flows fair share

$$S_f^r(T) \geq \frac{w_f}{\sum_{g \in F_r} w_g} \cdot T$$

But which resources will be constantly backlogged in our system? Even though the sources place continuous demand, this does not mean that in the view of the input servers, these flows will be always backlogged since the backpressure may deactivate them(even though this deactivation is instantaneously, this can happen for arbitrary many times(infinite) in any time unit).

If we order the resources in terms of decreasing congestion (that is the sum of the weights of the flows passing through this resource), then certainly the first resource CR_1 in this ordered vector CR will be continuously backlogged, since for each flow $f \in CR_1$ the rate received by this server to this flow, will be at most equal to the rate received by the complementary server $\overline{CR_{1f}}$. So the rate for all flows $f \in CR_1$ cannot be greater or smaller than their fair share in this server. That is:

$$\forall f \in CR_1 \qquad S_f^{CR_1}(T) = \frac{w_f}{\sum_{g \in F_{CR_1}} w_g} \cdot T = S_f^{\overline{CR_{1f}}}(T)$$

iteratively we can find the final rates of all flows, removing the flows considered so far and their assigned capacity. All we need at each step, is to find which flows will certainly be backlogged at some server so that by the GPS definition we will be certain for their rates. This does not means that we examine the resources in the order that these appear in CR. Since some flows have left over excess bandwidth in adaptation to their backlogged fair shares, it comes that the new fair shares at resources may be greater than those in the ever-backlogged case and consecutively the congestion of flows in general does not follows the sum of weights on the resources. The flow(s) f that will next, be continually backlogged at some server s, will certainly be those that receive from s the minimum utility over all remaining flows and all servers. If this flow is not backlogged at this server, means that the rate of arrivals(or activations) on this server s, is less than the rate received by s which is inept.

So what the fluid model will guarantee, is that the next most congested flow will certainly receive its fair share, since their is no way of getting less-or more than this. If a flow f could get more rate than this, it means that f stole rate, either from a previously backlogged flow(the GPS server would never permit this), or from a flow that is backlogged together with f(again GPS discipline cannot permit this – since GPS assigns equal utilities to backlogged flows). That this flow f cannot get less than its fair share is for one more time a consequence of the GPS discipline – since f is backlogged at s.

Again this flow may leave excess bandwidth to other flows which may be able to use it, if these do not face a more strive bottleneck somewhere else. The bottlenecks appear as this procedure is being born, but since we have found a way to take the "next" step we can determine the final rates.

Since this procedure perfectly maps to the algorithms for computing the weighted max-min fair allocation, all fluid-flows will be finally assigned rates equal to their weighted max-min fair share. ♠

Of course the most "privileged" sources in this procedure, are those that are assigned rates during the final steps of the algorithm, since these have higher probability to increase their rates upon their minimum guarantee(in the input and output servers), by utilizing left-over bandwidth.

4.3 The packet system behavior

.

4.3.1 Sources of inaccuracies

Of course the nice theorem above represents the reality on a fluid world, a world that although is frequently used as an rough and elegant description of the "packets" worlds, in the same time is million miles away. What lies in-between these two "realities" is the subject of this paragraph, in the perspective of-course of our system and the weighted max-min fairness objective.

To start with the GPS servers must be replaced with some packet approximation. Although good approximation really exists even the slightest inaccuracies in service can accumulate into big discrepancies for the real system, since in our model the services for f by a server r, is not only the "service" given by one server, but in addition steers the arrivals process of the same flow f, for the complementary server ($\overline{r_f}$). Such inaccuracies are inevitable.

And secondly the instantaneous adaptation between the rates assigned to flows by different servers, in our packet system can only be approximated through the backpressure flow-control scheme – which we initially used to prevent buffers from overflow. The size of the backpressure threshold T, plays a central role in the provision of fair service. A known fact, is that if we want a flow under credit based flow control to sustain a maximum peek rate R_{peak}, the size of the backpressure threshold T must be at least as big as this flow's peak rate times the round-trip time[42]

$$T \geq RTT \cdot R_{peak}$$

This buffer size ensures that the buffer will not overflow and additionally that the peak rate can be realized. This is the point where usually the analysis for inter-switch flow control stops. This is logical since in great distances the round-trip time is a considerable and most of the times the dominant factor[43]. In our case the distance of the circuits that change and realize the change of the backpressure cannot be greater than some tens of centimeters and can be neglected. De/Activation and realization can happen even in consecutive clock cycles, so for example a maximum internal packet worth of buffer can guarantee that an no contented flow will occupy full capacity, that is will transmit at peak rate. So we can fairly assume

[42]When the backpressure is deactivated for a previously congested flow, it will take RTT time for a new cell to arrive. In this time, the flow should sent $RTT \cdot R_{peak}$ cells to sustain each peak rate.

[43]This buffer size, is usually denoted as *window*

that the latency of the backpressure signals is close to zero and can be ignored[44]

For our system and in the view of providing max-min fairness, there exists other delay sources that can impose a higher backpressure threshold. Since our model is not fluid, it follows that the servers cannot instantaneous assign to a just re-activated flow its fair share(This would imply that more than one flows can be serviced simultaneously).

4.3.2 Providing the minimum fair share

To start with, a prerequisite in providing weighted max-min fairness, is that every backlogged flow f, will receive at least the minimum of the rates, that this flow rightfully deserves from the input and the output servers individually. Following the analysis presented in [ChQoS], it follows that a buffer size of one maximum packet is adequate in the case WF^2Q are deployed. Although the analysis is quite intuitive and simple, it is not that clear[45]. Consider the two following examples.

Suppose that a flow deserves a rate R_f^{in} at the input and rate R_f^{out} at the output. Now suppose that $R_f^{in} \geq R_f^{out}$. So the input tries to fill up the crosspoint buffer of this flow with a rate greater than the output drains it, which will result in the backpressure activation. Suppose now that backpressure is activated and comes the time t_{out} that the output services this flow and the backpressure is deserted. The input will receive this backpressure status change, during the consecutive scheduling cycle-a result of the zero backpressure latency- and will be able to serve this flow again. But in the worst-case, how long may it take for the input to reschedule this flow? During this time interval how many services should this flow receive? This number of services $S^{out}(D^{in})$ which translates into a minimum backpressure threshold in the view of providing the minimum rate is:

$$S^{out}(D^{in}) = D_f^{in} \cdot R_f^{out}$$

Now remember that for a WF^2Q and in the case of constant packet sizes, the following relationship holds between the arrival time a_f^k of the $k-th$ packet of flow f and its departure time d_f^k:

$$d_f^k - a_f^k \leq \frac{Q_f(a_f^k)}{R_f} + \frac{L_{max}}{R_f}$$

In our case we are interested in the delay of the packet in the front of the queue of flow f, since the deactivation of the backpressure translates into the activation("arrival") of this HOL packet. So using $Q_f(a_f^k) = 0$ we have that:

$$d_f^k - a_f^k \leq \frac{L_{max}}{R_f}$$

and the worst case delay D_f^{in} that we are interested in, reduces to $\frac{L_{max}}{R_f^{in}}$. So it seems that it suffices to allocate a buffer size of one packet for each flow since:

$$S^{out}(D^{in}) = \frac{R_f^{out}}{R_f^{in}} \leq 1$$

[44]Of course additional delays of technical nature, may impose a larger window.

[45]In [DGPS] for example a quite different result is presented

Symmetrically to the case presented if $R_f^{in} \leq R_f^{out}$, that is the buffer drains with more speed than it gets filled then starting with the prerequisite that the backpressure should never be asserted and following the same analysis as above we come up that the buffer should be at least equal to $D_f^{out} \cdot R_f^{in}$, which in the case of WF^2Q servers reduces to one cell.

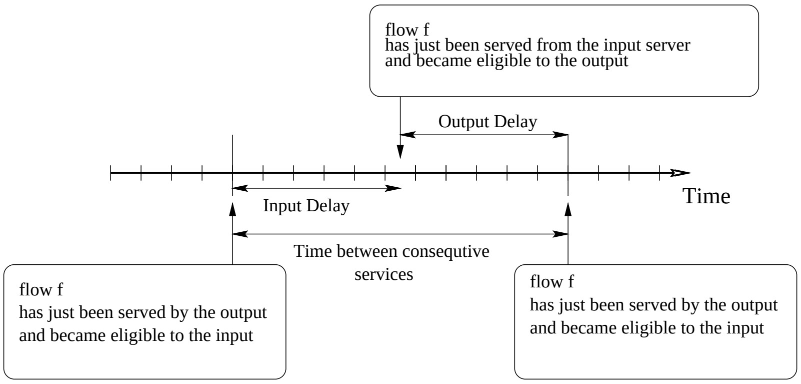


Figure 17: One service scenario

Although this analysis is generally realistic, it can be proven that it does not apply to the most degenerate cases. What if both server take us much long as possible to service the flow, like in the scenario shown in *Fig. 17* . The theory that we have presented so far did not reveal this possible discrepancy, where both servers put off flow's f service, as much as possible and as a result f is granted only one service every $D_f^{in} + D_f^{out}$. If this phenomenon is to happen repetitive then a flow might receive only half the minimum throughput, even in the case that WF^2Q servers are used.

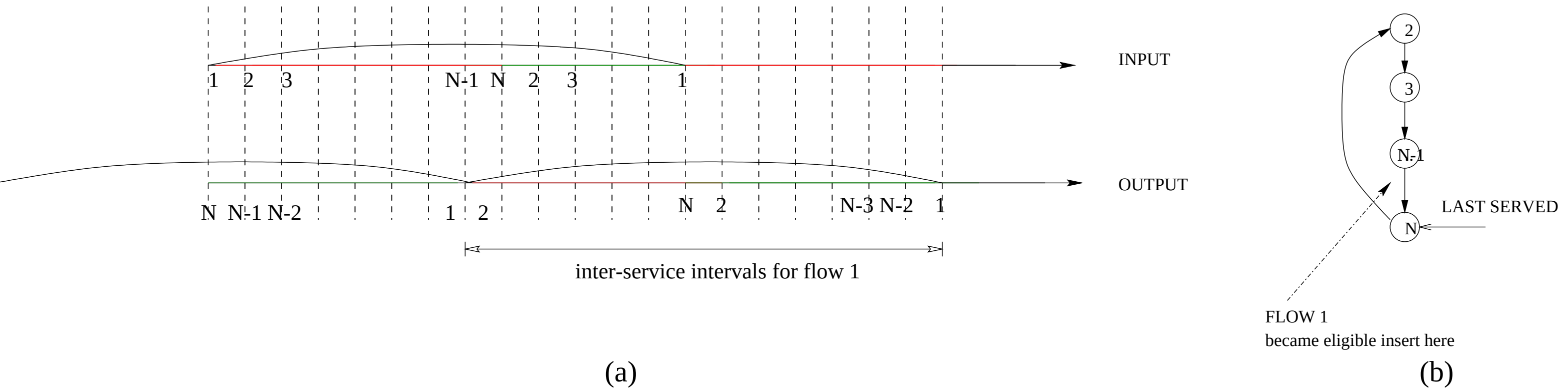


Figure 18: A linked list implementation that applies worst case delay to fresh eligible flows, will harm synchronized well behaved sources. In the case of our system, this will lead to the flow receiving only half the fair throughput.

An other simpler way to see this is two consider that the servers at the input and the output are simple Round Robin schedulers. In most of round-robin implementations the maximum delay that an active flow may experience is N schedule cycles. So it appears that one cell buffer space will be adequate, so that each flow will receive $\frac{1}{N}$ of the total bandwidth. But in the case that both servers re-schedule the departure of a fresh eligible flow in the end of the current round-robin cycle – this is shown in *Fig. 18* (b) for a possible linked list implementation of the Round Robin discipline – then it may be the case that some flows will receive only $\frac{1}{2 \cdot N}$ rate – *Fig. 18* (a) –, if only one cell buffer space is provided.

To remedy this, Stephens ([DGPS]) suggest that a buffer space equal to $(D_f^{in} + D_f^{out}) \cdot R_f^{min}$ would be adequate, since every $D_f^{in} + D_f^{out}$ of time only the buffer space of service is provided[Stephens]. In the case of Weighted Fair servers this buffer space is close to 2 cells buffer. The question of weather this bad synchronization will happen or not is reduced to weather or not a server will apply the worst-case delay to well-behaving flow – that is a flow that is eligible in less or equal rate than its guaranteed share. Most work conserving rate-servers do not punish flows for receiving excess service in the past and in the same sense do not necessary prefer flows that were inactive in the past. This is true for the $P-GPS$ servers and WRR that we have presented in section 3.1 . It is also true for the implementation for the Round-robin that we presented above[46]. Instead the virtual-clock method, will give much higher priority to a new eligible flow that has missed some of its service rounds in the past. It is interesting to see how all these service disciplines behave under the buffered crossbar topology.

4.3.3 Buffers role, in providing the weighted max-min fair share and higher utilization

Additional buffer space in the general case can provide better utilization in the case of transient sources. A simple scenario is shown in *Fig. 19* . Buffer space pushes traffic closer to the congestion points, in the places where this-the traffic-will be able to grab any service opportunity. An other view is that it places the flows service histories onto consideration for the present service decisions. In absent of these online-histories all "presents" should be coordinated, with the extreme of no buffer at all when a bipartite graph matching is needed. So the higher this history-window the more elaborate the scheduling decisions.

So far we have elaborated on the buffer requirements that guarantee that each flow will receive their minimum fair shares. Does the same buffer space guarantee that excess bandwidth will be distributed fairly – or will be distributed at all? Returning on the GPS-model, we showed that the constructive theorem – which is equivalent to the algorithm for producing the weighted max-min fair shares – assigns at each step the fair share to this backlogged flow that receives the minimum utility over all other flows. It follows then that the solution to our problem is to guarantee that this flow will be there ready to receive this fair share. Note that this fair share can be the combination of guaranteed rate and excess bandwidth

[46] Although as presented in [Kat-534], a better strategy can be followed

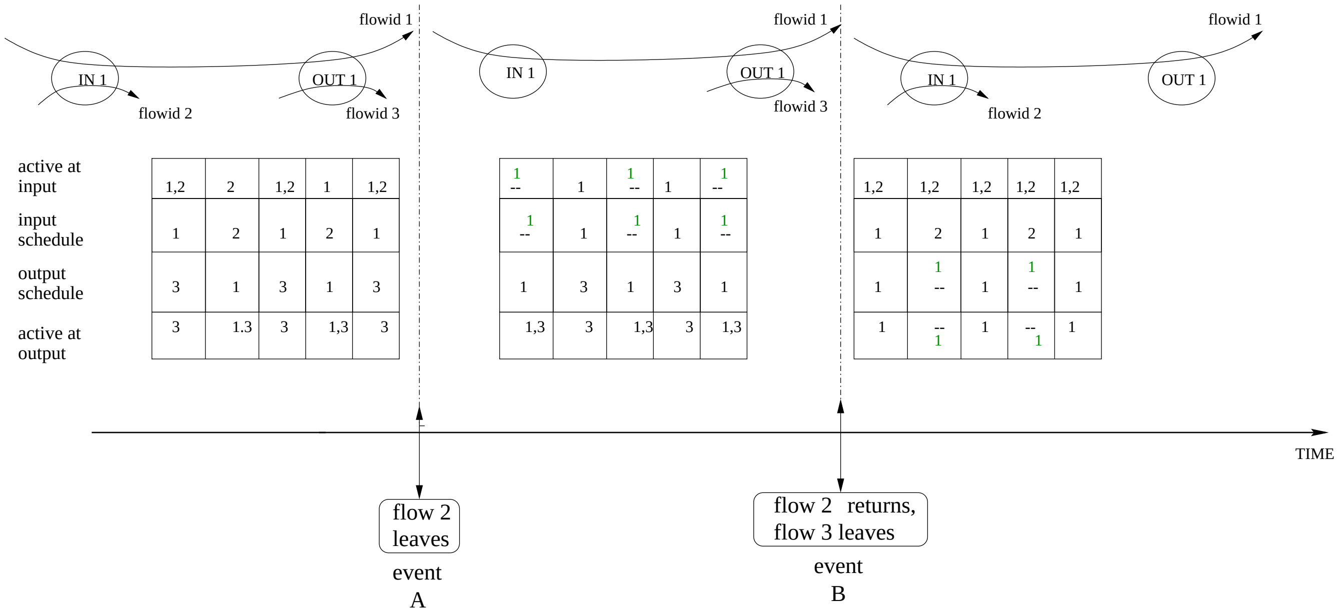


Figure 19: Tables illustrates the argument that bigger crosspoint buffer can provide better utilization. These figures shows one input and one output server. There are three flows in the system, flow 1 which uses both the input and the output, flow 2 that uses only the input-and some other output-while flow 3 uses only the output-and some other input. In the case of a single buffer, this output will remain under-utilized after event B, because flow with id 1 cannot use the excess bandwidth since this flow confronts a bottleneck at the input. If this flow could use a buffer to put the excess bandwidth between events A and B, when flow 2 was inactive, then this flow would be able to occupy the output link after event B(as is shown with the green color), even for a while.

left over from more congested flows. This excess bandwidth will not be distributed uniformly in time and the flows that want to get access to this, will need to sample the resources frequent enough. This sampling can be emulated by using higher buffer space.

In *Fig. 20* we see two different “sampling” scenarios for excess bandwidth distribution. Two equal flows content for the same output but flow with id 1, is more congested than flow with id 2. It turns out that flow with id 2, misses some of each schedule turns (those marked red). In case (b) flow with id 1, happened to be able to take advantage of these excess even with buffer space 1, which did no happen in (a) due to unlucky sampling. With two cells buffer space the problem is solved due to higher sampling frequency.

In sections section 5.2 we study through simulations, the hypothesis that the system converges to the weighted max-min objective.

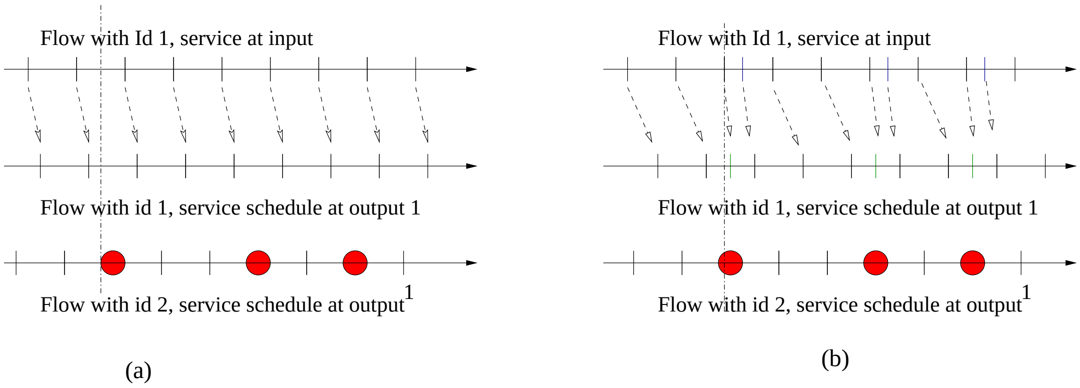


Figure 20: Flow with id 2 and flow with id 1 have equal priority for the output link. But flow with id 2 suffers from a more strive bottleneck in the input, which results in empty slots(red circles). With one buffer space, the flow with id 1 – which can accommodate more bandwidth from the input –, can happen to be able to get advantage of these empty slots(as happens in case b), or may not(as happens in case a). If we use additional buffer space, the flow will always be able to get advantage of the excess bandwidth

4.4 Transient and probabilistic traffic Behavior

Before we turn on the simulation results, we stop and consider two issues which are highly critical for the final performance of the switch. First we define the notion of flows trajectories under equilibrium, a valuable concept when examining the solidity/elasticity of the system under equilibrium – the time to reach the equilibrium and the time to recalibritate after a change in the conditions – and additionally a “catalyst” idea in understanding the process of reaching the equilibrium in the first place. Next, we analyze the system behavior under probabilistic sources and discuss on the saturation and the worst-case throughput of the system, compared to pure output queuing or maximum bipartite match algorithms. Both issues are of major importance, since these will qualify our scheme under more realistic conditions.

4.4.1 System performance at condition changes – how fast does the system (re)calibrates

So far we examined flows with constant state – either active or inactive – and constant weight/urgency degree. We presented results that support our hypothesis that the system will stabilize at rates very close to weighted max-min fairness. It seems that if *adequate* buffering is used at the crosspoints and we *wait* for some time, then weighted max-min fairness can be enforced – intuitivelly in these cases we expect our system to behave close to the fluid model. Although necessary in the common purpose of observing the objective, these two prerequisites are in some sense conflicting. Big buffers generally increase system inertia and delay the enforcement of any scheduling objective, while small buffers may fail to produce it at all. In this section we study exactly this, the time needed to produce the weighted max-min fairness objective and the system’s elasticity at condition marginal changes.

4.4.2 Motivation and methods

Any theoretical or experimental result, must weight-value its self under *normal* condition – that is conditions which are usually much different from the theoretical model or the experimental environment, used to produce it in the first place. Of course by definition this is not possible through additional theories or experiments, but this is what scientific-methodology is all about. In this section we ask our selves. Ok, if our hypothesis is true and under the scheduling discipline-topology of the buffered crossbar that we propose, the set of backlogged flows converge to rates close to the desired objective, how much beneficial can this be actually, in more *realistic* environments of demand?

Under normal network conditions, where the demand changes in an unpredictable fashion, we cannot sustain that all flows will keep their state(active/inactive), waiting to receive their fair shares – no matter the aggregation level that these flows represent. Additionally with most hypothesis on the nature of weights, we expect weights to change – either if these represent the degree of urgency of a parallel application, or the number of active micro-flows, or even the congestion at downstream nodes. So the motivation of this section is to examine at some level, the applicability/importance/special-weight of the enforced discipline under more realistic(= changing) environments. The methods can be easily captured from the motives.

4.4.3 Flows trajectories/how far can a change propagate

The demand of a flow f – considered in isolation – can change, when its weight changes or when f switches states, between active and inactive. So we regard as intrinsic change in the condition of a flow, either the change from active to inactive and vise versa, or a change in the priority index(positive-or negative). But since we talk about rate assignments under equilibrium – implies fragile interconnected bones that construct a "stable" state in time –, a flows demand-state can also be affected, when the rate of some other flow(s) changes . This again can be interpreted – and propagated in the system – as a new change in the demand of this flow.

Clearly if an intrinsic-change does not affect nothing in the system, the time to reach a new equilibrium will be minimal. But if this disturbance(s) affects all other flows (and links), then we expect this time to increase. The area of the change is not the only parameter for the time to reach the new equilibrium. The degree of parallelism among flows, in the process of creating the new equilibrium, also affects the time to converge.

We define as the trajectory of a flow f, this set of resources that must (have)change(d) states – rates and utilities considered at equilibrium states – if f change(d) rate. This notion is important, if we want to argue about the time to reach a new equilibrium after a perturbation in the condition of one, or more flows. This notion is also important for the time the system needs, to reach the equilibrium in the first place and for the actual convergence of the system. Clearly if one flow g belongs to the trajectory of an other flow f, then g will have to wait for f to stabilize before this can start "safely searching", its fair share. Since the rate adaptation of flows is not instantaneous – is provided through the backpressure – flows may delay their adaptation to their fair shares. This is because the state of the crosspoint buffers may delay the propagation of congestion from input to output and vice-versa. This (delayed)change in the demand of f, will appear as an delayed event for the other flows that belong into f's trajectory. So the two problems, namely system's elasticity after condition intrinsic changes and the system's convergence realization/time from the scratch, are highly dependent if not equivalent – anyway system stabilization can be seen as the big bang, where all flows change state from inactive to active.

The chain(s) of changes An intrinsic change can cause disturbance(s) to other flows, and these again may act as new change(s) in the chain(s) of interactions that transform an equilibrium A to a new equilibrium B. If we want to locate these chains, all we have to do is to run the algorithm for computing weighted max-min fair allocations for these two different states and mark were these differ. Here we want to describe the "degree" of the change (probably in the worst case), how far can this spread and what is the time it will take for our system to stabilize. We are interested *in* the process of the *system's change* as a whole and the best way to understand it is to mechanically construct it.

Who have not been affected, a situational description Lets return to the notion of the trajectory of a flow. We will try to formulate a situational description for this set that will help us to extrapolate some first properties. In this target, two situations are of obvious importance. The one prior to the change and the one after the change. Lets call the respective equilibriums A and B. We know that these two equilibriums can be described by the respective allocations utilities vectors – that is an increasing ordered vector, of the utilities that have been assigned to flows. Now let f be the flow that intrinsically changed. We argue that after the disturbance has converged, a flow g that has been affected should have a bigger or equal index than f in this rank, either in A or in B[47]. That is g should have a smaller congestion index, either after or before the change.

Theorem 4.1 *After an intrinsic change in one flow's (f) demand, that pushes a network from equilibrium A to B, a flow g will change rate only if $U_g^A \geq U_f^A$ or $U_g^B \geq U_f^B$.*

Proof by contradiction
Suppose that g had strictly smaller utility in both equilibriums and g was affected by f's change. Let V^A and V^B be the utilities vectors of equilibrium A and B respectively. 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:

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

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

So far **Theorem 1** informs us, that a change of f cannot have affected one flow g, that is "more bottlenecked/congested" both before and after the change. It does

[47]To prevent some possible misunderstanding here, when we talk about congestion and say "higher congestion index", we mean that flows get less utility and appear in the left side of the utility vector. Here we say bigger/higher or equal *index* and we mean the opposite, that is this flow gets equal or greater utility and is more to the right – opposite to the lexicographic order.

not say that all flows which are less bottlenecked during the previous or the current equilibrium are actually affected. This theorem, partially establishes a causial order for the events/changes, in the system. It can tells us, which event/change is the cause and which the affect/change, in case we are not sure. But not completely. For one, flows have to be *connected* if we want them the interact, so that we can establish a causial order. If these are not connected, then the events are independent and there can be no causial relationship, but only a meta-physical one.

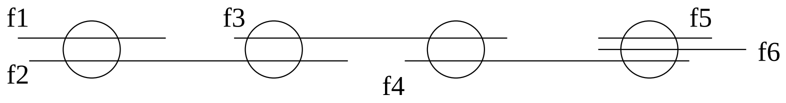
Flows interaction graphs, adding flows to the *not* affected set *Two flows (f,g) are said to interact, if these share a common resource, in which case are called neighbors - a flow is neighbor of itself -, or a neighbor of f interacts with a neighbor of g.*

Flows interaction graphs, are formed independently of weights and express rather topology and not demand properties. Of course the interaction graph depends on the existence of the flows that construct it. So the interaction graph depends on demand(existence), but does not reveal the trajectories notion - which is the ultimate demand dependent correlation concept, but discipline dependent[48]. A flow's (f) change, can potentially affect all the flows g in the interaction subgraph of the network that contains this flow f and only these flows g. Demand and enforced disciplines, restrict the set of flows(or links) that will be affected, into this subset of the interaction graph that we call trajectory. **Theorem 1** does so, taking the change for granted. That is, it can find some flows that are not affected, given the change. But if for example, the weight of flow f increases to this point that in equilibrium B, f gets the smallest utility over all flows, then all flows in the interaction graph of f might change according to this theorem, no matter the congestion index of f, previous to the change. Next, we will try to make this set of flows even tighter.

For each flow g in the trajectory of a flow f under equilibrium A, we can consider probably many distances, one for each different path/chain in the interaction/interconnection graph that includes both flows. For each path, a distance metric can be defined. Neighbor flows have distance 1, a neighbor of the neighbor of a flow has distance 2 etc. This is shown, in *Fig. 21* . There we see, that neighbor flows, can have either the same distance from an other flow, or distances that differs at most ± 1.

Proposition 4.1 *For each flow g in the trajectory of flow f under equilibrium A, if g changed rate after a change in f, their should be at least one path connecting f with g - call n their distance under this path -, for which the flow with distance $n-1$ from*

[48] If one flow will affect an other flow, depends on the discipline, since if for example strict priorities are used, no change can affect the highest priority flow.



not_neighbor(f1,f3)
d(f1,f2) =1, d(f2,f3) =1 ---------------------> d(f1,f3) =2

not_neighbor(f1,f4)
d(f3,f4)=1, d(f1,f3) = 2 --------------------> d(f1,f4) =3

not_neighbor(f1,f5)
d(f4,f5)=1,d(f1,f4) = 3 ---------------------> d(f1,f5) =4

not_neighbor(f1,f6)
d(f4,f6)=1,d(f1,f6) = 3 ---------------------> d(f1,f6)= 4

Figure 21: 1

g, changed rate in the opposite direction than f.

Proof by contradiction

If this is not the case, then for all paths connecting f with g, f "communicates" with g, through a flow j of distance $n-1$, that changed rate in the same direction with g. If g dropped rate, then all of g's neighbors with distance $n-1$ from f, cannot have increased rates. Then g is either affected by an increase of a flow(j), with the same distance under all paths – which becomes a cyclic argument and cannot happen[49] –, or from a flow(j) with distance $n+1$ for all paths; in this case, if g was affected by j and not the opposite, then j must be interacting with f through another path that does not contains g, which is contradiction, since then in this path, the distance of f with j, cannot be greater than the distance of f with g.

So we came up with that, if all neighbors of g, with smaller distance than g, have not increased their rates, then *all* of g's neighbors, must have not increased rates, which means that the second allocation is not max-min fair, since we can increase the rate of this flow without risking neighbor flows utilities.

If f increased rate, then through the same arguments, all of its neighbors increased rates also, then why f did not do so in the first equilibrium? – the first equilibrium was not max-min fair ♠

Proposition 4.2 *If under all paths connecting f with g_1 and g_2, the distance of g_1 with f, is equal to the distance of g_2 with f, then after a change in f, both g_1 and g_2 can only be affected in the same direction.*

Proof

Since for all changes in the interaction graph there must be path that contains them, alternating between increases and decreases when the distance increases,

[49]These candidate flows j, must have been affected only by flows with the same distance, cause otherwise there is a path in which j has less distance than g. This causes a null change to be realized.

g_1 and g_2 that have the same distance can only be affected in the same direction, starting with a common event ♠

Lemma 4.1 *A change in the demand of some f (intrinsic or induced), will positively affect the rate of a flow g, only if the change induces(or is), a negative change in the rate of a flow j - in terms of stabilized allocations, where j is g's neighbor in the bottleneck link of g at equilibrium A(b_g^A).*

Proof
The proof is obvious, since $U_g^A < U_g^B$ means that the maximum demand-congestion in both links used by f during B, is less tough than the harder facing during A, it means that some flow dropped its demand for b_g^A ♠

Lemma 4.2 *A change in the demand of a flow f (intrinsic or induced), will negatively affect a flow's g rate, only in the case that the change induces(or is), a positive change in a flow j, where j is a neighbor of g and the utility of j is less or equal than the utility of g, in the first equilibrium($U_j^A \leq U_g^A$).*

Proof by contradiction
Suppose that this is not the case. Then g is not a neighbor of f and all the flow's that share a link with g and have increased their rates were taking at least equal utility with g. Since g rate dropped under equilibrium B, one of its neighbors must have increased his rate(demand) in the new bottleneck of g - otherwise the new allocation is not max-min fair. But since all flows that have increased their rates(and by so utilities) where granted greater utility than g under the first equilibrium, theirs utility grew where g's dropped, so g cannot have a bottleneck and the contradiction is proven ♠

Theorem 4.2 *The trajectory of a flow f under equilibrium A, is bounded into the neighbors of f and all other flows in the interaction graph of f, that belong into a chain/path of interacting flows which starts from f or some neighbor of f, and this chain/path contains flows, of not-strictly increasing utilities under A.*

Proof by induction

By proposition 1, all flows in a trajectory must belong in a chain, alternating between positively and negatively affected flows. The proof will be on the distance of the chain, in the interaction graph.

Denote as Y the smallest utility, in the neighborhood of f under A. If the distance is one, means that g is a neighbor of f so theorem holds. The induction hypothesis tells us, that if a flow g with distance n from f changed its rate, then $U_g^A \geq Y$.

Now, if a flow g increased its rate, then this must be in reaction to the reduction of the rate of some flows i, neighbor of g. By lemma 1, we have that this reduction occurred at g's bottleneck at equilibrium A, so in any case g had greater-equal utility than i, under A. According to proposition 2, there is a path, so that if g has distance $n+1$ from f, then i has distance n. Now the inductive hypothesis tells us that $U_i^A \geq Y$. So $U_g^A \geq U_i^A \geq Y$.

If a flow g, reduced its rate, then Lemma 2 tells us that this reduction was in response to the rate-increase of some other flow i, with smaller utility. Again $U_i^A \geq Y$ and through the same use of the proposition 2 and the inductive hypothesis: $U_g^A \geq U_i^A \geq Y$ and this completes the proof ♠

So if a flow belongs into the trajectory, then this must obey to the description of the theorem above. Note though that we cannot say for sure, that all flows in this set will change[50], but certainly we have bounded this set enough.

In the above formulation we think of an inactive flow, as getting infinite utility. A flow can affect its neighbors, no matter its previous congestion index, by simply increasing its weight(or becoming active). This explains why Theorem 1, used the description *"smaller congestion index than f, either before or after the change"* - it did so, to capture exactly this case. But this change will next propagate to others(with greater distance in the interaction graph), only if f shares a link with a flow having a big congestion index.

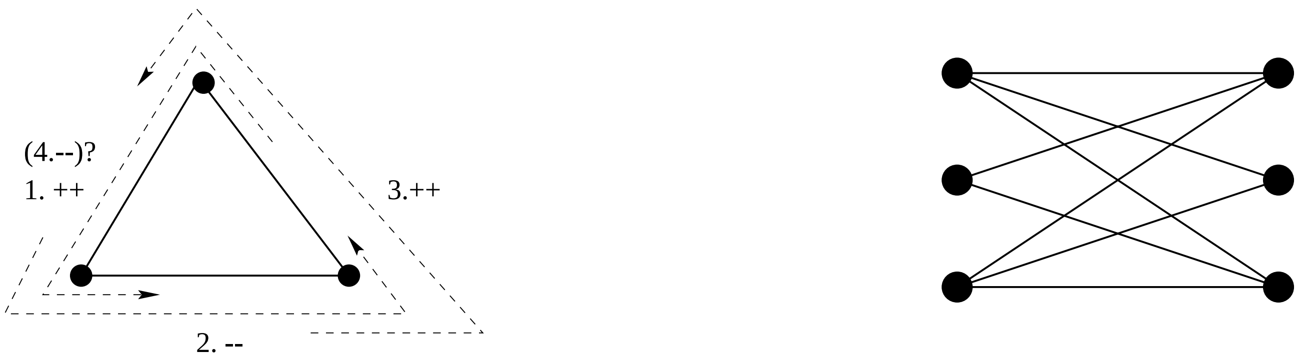
Corollary 4.1 *A flow f, that changes states from active to inactive or vice-versa, can only affect other flow's in the interaction graph of f, with smaller-equal congestion index(greater utility) than f, considered at the equilibrium with f active.*

Proof

If f becomes inactive, a neighbor g will be affected(by Lemma 1), only if this faces as bottleneck the link shared with f. If the link shared is the bottleneck of f, then

[50]There is the suspicious that if the congestion index is strictly decreasing then we could say it

g and f had the same congestion index. If the link shared is not the bottleneck of f then g has smaller congestion index than f. It follows from the theorem that all the flows affected will have equal or smaller congestion index than f.
If the flow becomes active from inactive, then the flows that are affected are those that were affected by the opposite event, so we are finished ♠



(a) arbirtary network interaction graph

(b) crossbar interaction graph = bipartite graph

Figure 22: In a random network a change in the rate, can potentially cause fluctuations, because of the co-existance of odd and even distances, in interaction paths(a). This cannot happen in the crossbar topology, since any change can only reach a flow, after only even or odd transitions(negative2positive, positive2negative)

Crossbar considerations The results above hold for arbitrary network topologies. The crossbar topology has some interesting properties. First a change in one flow's rate, that reaches a second flow through any of the possible dependency chains on the interaction graph, can only have the same effect-either increase or decrease in rate. Since a change propagates alternating between increases and decreases in flows rates, a change that reaches a flow through two different paths on the interaction graph, might be conflicting about this flows new state(rate). Although this might hold in arbitrary networks it cannot hold in the crossbar topology, since any cycle in the interaction graph(which is a bipartite graph) of the crossbar has even number of edges(transitions). So starting with a change on a flow(induced by an other flow or by a change in weights), we cannot end to affect the same flow – through an interaction path –, the one we started with, in the opposite manner(*Fig. 22*). This could be problematic and for the stabilization process of the actual system, to the weighted max min fairness, as we shall see in a later section.

4.4.4 *Big* changes - *Small* changes - *Slow* changes in the buffered crossbar

To estimate the worst case duration, that it may take the system to calibrate, starting from the scratch or after the a condition change, we have to find the flows with the greater trajectories. If the condition of this critical flow changes, then it may be the case that all flows in the system needs to react. We would also like this change to be stepwise, that is to occur one step at a time – if we want to capture the worst case.

We know that the depth of a flows trajectory cannot be more than the bottlenecked links of the network (since this is the maximum length(depth-distance) of an increasing utility chain of interacting flows). The changes propagates as congestion waves. A flow increases rate in response to a flow decreasing rate and the decrease of a flows rate increases some other flow's rate etc. Each of this dual-events do not necessary occur in parallel in the crossbar. A flow that drops its rate because this becomes congested at the input will need to transfer congestion to the output – cause the crosspoint buffer was probably full(changed bottlenecks). If this flow's drop is because this flow views a new bottlenecked at the output, the input will realize the change after the crosspoint buffers fills(changed bottlenecks).

In the opposite direction, a flows increase of rate at the input, the output will realize it "instantly"(i have to send, the crosspoint buffers are no longer empty). Again if a flow increases its rate at the output, this will be instantly transferred to the the input(i can send – the crosspoint buffers are no longer that full). Since each chain contains half increases and half drops, it follows that in the worst case the transfer will have to stop and wait buffers to switch states, at approximately half times the length of the chain. Since the maximum length of a path/chain is $2 \cdot N$, it follows that the final flow may have to wait N times the fill or drain of the crosspoint buffers, before this receives its new fair rate – here we can see more clearly, why big buffers increase system inertia. The rate that the buffers switch states affects this maximum time. The lower thess rates, the greater this time interval. So a first worst-case time estimate is, $UnfairTime \leq O(\frac{N \cdot B}{min(old\_rates)})$.

But these rates, will be proportional, to the old rates of the flows dropping their rates and in this view is inversely proportional to the unfairness realized during the stabilization process. So if count the unfairness that occurred in bits and not in time, then probably the old rates will not play critical role. So $UnfairBits \leq O(N \cdot B)$.

In the simulation results(section 5.3) we present these scenario and examine how, the time to reach the new equilibrium in these worst-cases, is affected by the crosspoint buffers size(B), the old rates of the flows($min(old\_rate)$), and the length of the interaction chain(N).

4.4.5 $\exists$ crosspoint buffers size: The buffered crossbar with $P-GPS$-related servers, will converge to weighted max-min fairness

In this section we will try a first proof, that there exists a size for the crosspoint buffers, that enables the buffered crossbar architecture, equipped with packet GPS-related servers to converge to the weighted max-min fairness objective, with arbitrary precision. The assumptions that we will make on the packet servers are not very demanding and can be actually provided by most virtual time implementations.

Definition 4.1 *A server s with competing flows F, is called (l,k)-fair if for each time interval T and subset of the competing flows G that fairly receive/consume total aggregate rate C_G during T, $\forall f \in F \cap \overline{G}$, if f is continuously backlogged during T, the following equations hold:*

$$S_f^s(T) \geq \frac{w_f}{\sum_{g\in F\cap\overline{G}}(w_g)} \cdot T \cdot C - k$$
$$S_f^s(T) \leq \frac{w_f}{\sum_{g\in F\cap\overline{G}}(w_g)} \cdot T \cdot C + l$$

where C, is the available capacity of the server, after subtracting the capacity assigned and used by flows in G($C = C^s - C_G$)

The above definition bounds the the maximum discrepancy that a server may apply to a flow, when this receives a combination of guaranteed and excess bandwidth. So flows in G may utilize only part of their guaranteed bandwidth, leaving excess for the others. But the requirement is, that flows in G *use/consume* constant bandwidth – not more or less – that does not changes in time. Note that a GPS server, is (l,k)-fair with $l = k = 0$.

Lemma 4.3 *In the buffered crossbar architecture equipped with (l,k)-fair servers, if all flows with bigger congestion index than f have stabilized into their max-min fair rates, then f will eventually be at some point, backlogged in its proper backlog - defined under weighted the max-min fairness.*

Proof

Since all flows with bigger congestion index, have stabilized into their fair rates, it follows from Theorem 2 that these will not change, what ever other flows will less congestion index do[51]. Lets call b_f the bottleneck link of f under equilibrium and $\overline{b_f}$ the complementary resource used by f. First we will prove that this $\overline{b_f}$ will observe an increase in the demand by f. Mark as F_r the set of flows in a server r that have not greater congestion index than

[51] here is probably the only problem of the proof, but i think that this can be fixed probably using some properties of the crossbar interaction graph

f and C_r as the remaining capacity at server r, after subtracting the capacity assigned to the flows that use r and have greater congestion index than f. Since the bottleneck of f – in the fluid model or equivalently, under weighted max-min shares – is at b_f, it follows that the following relationship holds:

$$\frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f} \geq \frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} \qquad (1)$$

If the bottleneck b_f is an output, then the buffer should eventually fill and if the bottleneck is the input, the buffer should eventually drain. We are going to show the case where the bottleneck is the output, since the other case is simply symmetric. We know that the following general relationship holds for the state of the buffer:

$$\Delta B_f(t) = S_f^{in}(t) - S_f^{out}(t) \qquad (2)$$

Now if the crosspoint buffer is not filled, then this flow will be active at both servers. Using the properties of the (l,k)-fair servers we can show that:

$$\Delta B_f(t) \geq t \cdot \left(\frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - \frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f}\right) - k - l \qquad (3)$$

Using (1) it follows that for $t \geq \frac{k+l}{(\frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - \frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f})}$ the $\Delta B_f(t)$ will be certainly increasing until it reaches maximum. So the crosspoint buffer will be eventually full and the flow will be backlogged, in its proper resource/server-node.
Symmetrically we can prove that in the case where the bottleneck of f is the input, then the buffer will get eventually emptied and this completes the proof of the lemma ♠

Next we will proof that there exists a size for the crosspoints that can guarantee, that after the buffer of f – f has the same description with the previous lemma – is eventually filled(or drained) according to lemma 3, then this will never drain(or fill) again, so that f will never miss any more scheduling chances in its proper bottleneck, defined under the respective weighted max-min fair equilibrium.

Lemma 4.4 *In the buffered crossbar architecture equipped with (l,k)-fair servers, if all flows with bigger congestion index than f have stabilized into their weighted max-min fair rates at some moment t_o and f has filled(or drained) the crosspoint buffers correctly(lemma 3), then $\exists$ crosspoint buffer size, that can guarantee that this will not drain or fill again respectively.*

Proof
We will again prove this in the case that the bottleneck of $f(b_f)$ is the output, which is identical to the symmetric case. We are interested then, in the maximum burst of services provided by the output for this flow and in the maximum delay from the input in servicing this flow, since if these burst/starvation periods overlap, the buffer occupancy will be reduced as max as possible. So starting at some moment t_o the buffer is filled(previous lemma). We claim that we can find a size of this buffer, such that for any interval Δt after t_o, this buffer cannot be emptied at time $t_o + \Delta t$. For any moment till we reach time $t_o + \Delta t$, the flow f, will be continuously

backlogged at the output. The input cannot be always "back-pressured" not to service this flow, since this means that and the output does not service this flow and the buffer occupancy remains full – and under this scenario/hypothesis we are done. So lets think of an occasion, where both the input and the output can serve this flow. Using again (2) for the buffer occupancy and substituting the starting buffer occupancy, using $B(t_o) = B$, where B is the maximum size of the buffer, we get that:

$$B(t_o + \Delta t) - B = S_f^{in}(\Delta t) - S_f^{out}(\Delta t) \qquad (4)$$

or equivalently:

$$B(t_o + \Delta t) = S_f^{in}(\Delta t) - S_f^{out}(\Delta t) + B \qquad (5)$$

Now using the properties of the l,k server we get that:

$$S_f^{in}(\Delta t) - S_f^{out}(\Delta t) \geq \frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - k - S_f^{out}(\Delta t) \geq \Delta t \cdot \frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - k - \Delta t \cdot \frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f} - l \qquad (6)$$

or

$$S_f^{in}(\Delta t) - S_f^{out}(\Delta t) \geq \Delta t \cdot \left(\frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - \frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f}\right) - k - l \qquad (7)$$

So combining $(5), (7)$ we get that:

$$B(t_o + \Delta t) \geq B + \Delta t \cdot \left(\frac{w_f}{w_{F_{\overline{b_f}}}} \cdot C_{\overline{b_f}} - \frac{w_f}{w_{F_{b_f}}} \cdot C_{b_f}\right) - k - l$$

now from (1) we get that for every Δt:

$$B(t_o + \Delta t) \geq B - l - k$$

and if we use an initial buffer $B > l + k$ we get that $\forall \Delta t : \quad B(t_o + \Delta t) > 0$ and this completes the proof of the lemma ♠

Now using these results, we are able to prove that if we use buffer size greater than $l + k$, the architecture that we proposed will eventually reach the weighted max-min fairness, in the case of constantly backlogged flows.

Theorem 4.3 *If the flows using the architecture of the buffered crossbar equipped with (l,k) servers, are continuously backlogged at the inputs, then starting from any conditions in the crosspoint buffers, the rates of the flows will converge to weighted max-min fairness, if the size of the crosspoints is greater than $l + k$.*

Proof by induction

The proof will be with induction on the congestion index, but considering individual trajectories with different induction.

First we will prove that all flows having the highest congestion index, will converge to their weighted max-min rates. By Lemma 3, we know that these flows will eventually backlog the servers, that these are supposed to be congested under weighted max-min fairness. And by lemma 2 we know that the crosspoint buffer size that we use guarantees, that these flows will never miss any scheduling chances. We know then from the definition of the l,k servers, that these flows will receive their weighted max-min fair rates with accuracy increasing by time.

Now suppose that the same holds for all flows having index in the utility vector (-produces inverse taxonomy than the congestion index) less-equal to n. Then we have to prove that this will holds, for all flows having index $n + 1$. If a flow with index $n + 1$, is not in the trajectory of any other flow with higher congestion index, then we are done. If a flow belong in the trajectory of a flow with greater index, then by Lemma 3 and 4 we know that this flow will be eventually backlogged and that will never miss any scheduling chances, on the proper backlog server. Now in the long run, this flow will receive its weighted max-min fair rate, and the theorem is proven ♠.

Discussion-conclusion-future work This arguments that proved the theorem have probably two(or more?) pitfalls. The first has to do with the definition of (l, k) fair servers. A possible problem is that the flows consuming part of their allocated bandwidth, will actually in a packet system oscillate between periods of consuming less than C_G and periods consuming more than C_G, but on average will consume their C_G. It remains to be seen, if this actually poses some problem in the proof of the lemmas. The second is the point, that the less congested flows cannot affect the rate of more congested flows – after these have stabilized. Although this is true in the fluid world, it may happen in the packets world – if we do not prove the opposite. The final/(or whole the) problem is, to find the (l, k) for various servers. The definition of these servers, axiomatically resolves the "hart" and the difficulties of the analysis. This turned out to be clever, so that the analysis/rational could continue to the end of the proof outline. But it remains an open issue, and probably a dificult problem requiring discrete-algebric mathematics, on the details of the disciplines and the system's architecture/model.

In general we expect that l, k will be proportional to N – the number of competing flows. This is true for the $P-GPS$ and WRR/WFQ – since these servers have bounds of this order even in distributing guaranteed bandwidth – but the question is, if this is true for the WF^2Q also, that can bound the maximum discrepancy from the GPS model, under a small constant – invariant of other flows. But the definition of the (l, k) relates also the capacity received by other, more congested flows. Since these flows will receive oscillating rate in the packet system, it remains unclear if actually the (l, k) values for WF^2Q, are invariant to N. You can find experimental results, relative to the arguments made here, in section 5.2 , section 5.3 and section 10 .

4.4.6 Probabilistic sources

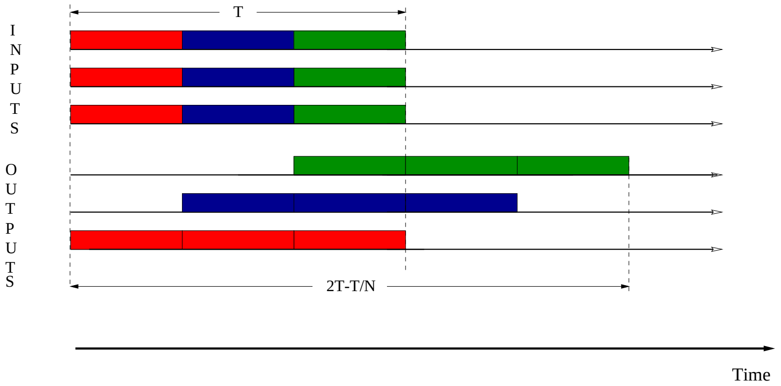


Figure 23: This figure show the worst case arrival pattern of long term non-conflicting traffic to a 3x3 switch. Although for any buffer size greater equal to 1 cell, the buffered crossbar will provide the same delay as output queuing , there are other cases when the buffered crossbar may provide less occupancy than the output queuing or maximum match crossbar-scheduling algorithm.

The simplest and easier arrival scenario, best suited for the crossbar topology, is when at each time_slot, the aggregate arriving traffic from the inputs, has the form form of full or partial permutations of the outputs. It does not require a genius to understand that the crossbar topology will be able to transmit these permutation with no delay, even in the case that we place the buffers. So the strictly non-blocking characteristic of the crossbar is invariant to the deformation introduced with the crosspoint buffers.

Lets now enlarge the arrival space into something more broad. We permit traffic to be short term conflicting(requesting the same output), but we still require that in the long run is non-blocking. That is we examine a time interval T when:

$$\forall i \in \text{Inputs} \sum_{f \in i}(r_f^T) \leq 1$$

and symmetrically

$$\forall i \in \text{Outputs} \sum_{f \in i}(r_f^T) \leq 1$$

Our question in this case is the maximum backlog that can appear in the system or equivalently, how long it will take for the system to transmit the traffic in the worst case. The best behavior can be accomplished with pure output queuing that will consume the minimum time possible to resolve the traffic and by the same time will provide excellent scheduling. Even with this ideal topology the system may take us much as $2 \cdot T - \frac{T}{N}$, to transmit the traffic, as is shown in *Fig. 23*).

Since the number of the packets destinated for one output (j) cannot be more than T and the burstiness of the arrivals to j is maximized when are all inputs diabolically synchronize and send simultaneously to j – N packets per time-slot can arrive in this case –, the worst case behavior is experienced when *all* sources have “deferred” the transmission of the $\frac{T}{N}$ packets destinated to j for the end of the period T. With crosspoint buffers equal to $\frac{T}{N}$, our system, will exhibit exactly the same behavior since in some es-sense belongs to the output queuing family of switches. But what will happen will less buffer size?

Since the output servers are work conserving, the occupancy of the output links, will always increase as the number of servers that are active – have backlogged traffic – increases. On the contrary occupancy decreases when the input servers select the same outputs and some outputs remain idle. We can verify that with the arrival pattern in *Fig. 23* the number of active output servers will be the same regardless of the crosspoint buffers size.[52]. In section 5.4 we have verified this argument.

So the time to transmit this traffic does not depend on the crosspoint buffers, and is equal to minimum possible. This result can also be verified with infinite queue push back theorem ([kat-534]).

Although in this worst traffic arrival, our system exhibits the same behavior with pure output queuing, it can be the case(in less dramatic arrival patterns), that the service of our system is much less than that provided by the output queuing. This simply has to do with that in our architecture, the backlogs are formed in the inputs buffers, while with pure output queuing these are placed at the outputs. If the input servers where then coordinated and provided individual scheduling decisions that when assembled at each time-slot, formed a maximum bipartite match, best occupancy could be guaranteed. In reality we cannot do nothing to deter inputs from selecting the same output continuously, for an time interval as large B(the size of the crosspoints)[53], when the backpressure will be certainly activated for some input.

So the worst case will occur, if all inputs make the same decisions, as long as the backpressure does not oblige them to do otherwise. If this synchronization continues, the number of cells transmitted can be approximately $1 \cdot B + 2 \cdot B +$

[52] *Proof overview*

When the second burst comes, one at most of the input servers will be able to send at the red output, so the rest will be forced to send blue. By this time the $1/N$ of red traffic has been sent. When there is both red and blue traffic in the system, since at most one input will be able to send at the red output, the blue output will also remain backlog. So when green(the $3-th$=$n-th$ burst) arrives, 1(=$n-2$) servers will have no other possibility other than sending to green. This means that there will always be some input sending to green. Generalizing, when the $n-th$ traffic burst arrives, there must be at least one input with no other decision, than serving this burst. So all output will remain backlogged from the time traffic arrived, till they absorb whole the traffic. So till the end of the T period, server 1 was continuously backlogged, while servers at outputs 2 and 3 where backlogged for the same time as with the output queuing architecture. Using the infinite push back theorem(or the conservation law), we can prove that the time to transmit all traffic cannot be greater than $2 \cdot T - \frac{T}{N}$.

[53] Since the server during the B time interval will transmit B packets, if $B \leq N$ then after B time slots, some input must be back pressured. If $B \geq N$, then at the time that all inputs have sent their B's, the output may, on average, have serviced all inputs, so at time $B+1$ all inputs can select this output again. But this cannot continue for time more than $t = \frac{N \cdot B}{(N-1)} \simeq B$

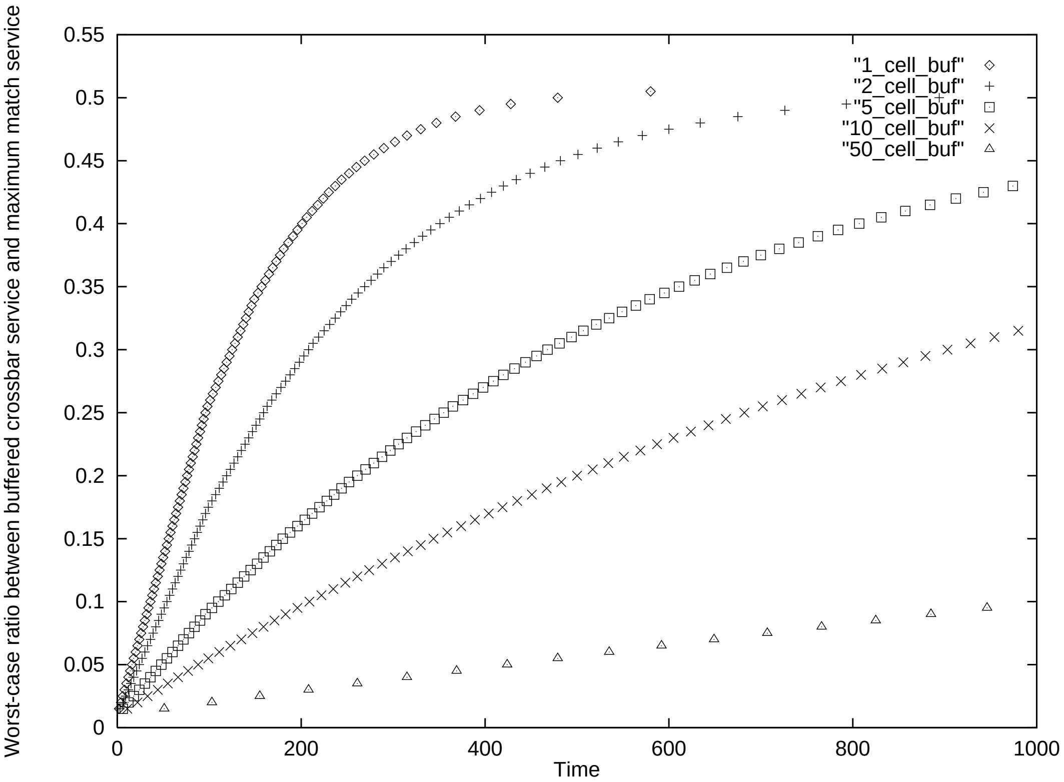


Figure 24: This figures shows the worst-case ratio between the cumulative service provided by a maximum match crossbar scheduling algorithm- and buffered crossbar, while it evolves in time for a 100x100 switch. Assuming that input buffers remain continual backlogged, the system can provide only half the maximum bandwidth, if it oscillates between one service per time-slot and N services per time-slot. We can see that smaller buffers reach maximum throughput in much less time.

$\ldots + N \cdot B$, when with a maximal match (or with pure output queuing) $N \cdot N \cdot B$ cells, would have been forwarded – if flows for all outputs were present. The ratio in the service provided by our system and that of a maximal match can be as large as $\frac{N+1}{2 \cdot N}$. So it seems that bad synchronization could force the system into bad configurations, but in the long run these cannot continue. For instance the ratio of buffered crossbar service and maximal match service can be as bad as $1/N$, but this can only last for B time-slots, where B is the backpressure threshold. After x such intervals of duration B, this ratio is approximately $\frac{x \cdot (x+1) \cdot T}{2 \cdot N \cdot T}$. This function is plotted in *Fig. 24* for a switch 100x100 and various size of crosspoint buffers. This means that under any traffic arrival of period T, that does not overload any output, our system can take as much as $2 \cdot T$ time to transmit the traffic, since T is the time that a maximal match(or output queuing) architecture would take, if all packets had arrived at time zero. It should be clear, that this scenario, is very unlikely to happen under normal arrival patterns – something, that does not look like *Fig. 23* , but again, not exactly (not)this, since then the behavior is identical to the output queuing – and/or "normal" scheduling disciplines. If we accept, that an output queuing system, can take us much as $\simeq 2 \cdot N$, to transmit a "trace" of feasible cells with duration T – we presented this scenario previously –, then a more certain bound for the buffered crossbar system would be twice the bound of the output queuing system($\simeq 4 \cdot N$). But this analysis is very approximate, and from our point, underestimates the system, since as we proved almost-analytically – we will later in this report confirm it and through experiments –, the buffered crossbar and the output queuing system behave identically for this worst-case, with respect to the output queuing system delay, feasible arrivals pattern. These claims, certainly worth to be further investigated.

A point here is that this worst-case analysis was driven only by topology and congestion constrains. For example, if round-robin was the discipline in the inputs(or something similar), it should be obvious that the inputs cannot fall/ "conspire", in the unlucky synchronization that we described. It can happen though if weight-based disciplines are used, probably through some degenerate combination of weights.

This analysis also reveals that we want the service histories, produced by the input servers, to be as much smooth as possible, so that we maximize the active periods of the outputs.[54]. So it seems, that if we use WF^2Q in the input contention points, we will have the best possible results under the "constrains" placed by the differentiation requirements and the independence that we require in scheduling; this discipline generalizes round-robin in the optimal way, and flows cannot be "starved" for more time, than something inversely proportional to their guaranteed rate, so burstiness is reduced. Results on saturation throughput under probabilistic sources, can be found in section 5.4 .

[54]Actually this ensures local maximum, since we do not account all servers decisions together

4.5 Conclusions

We provided arguments that our system can distribute service arbitrary close to weighted max-min fairness, by increasing the buffer space. This result is somehow rude and for exhibition purposes only, cause as remains to be seen, in most cases, a small constant size of buffer is adequate, since the servers of the system are unlikely to exhibit coordinated bursty behavior. Besides the incremental cost, we show one other reason why we may advocate for small crosspoint buffers. This is that we want, the system to be able to quickly adapt to the new fair rates, in the case of transient sources/weights. We will now turn to the simulations results, that hopefully will clear out the fog from the scene.

5 Simulation Results

.

5.1 Model - goals and parameters

.

5.1.1 Time model and event ordering

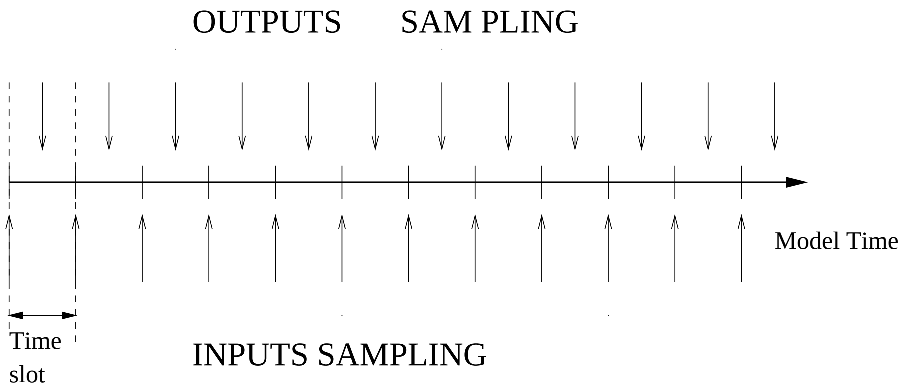


Figure 25: The simulation event ordering.

In this paragraph we discuss the causal order that governs the conceptual model of our hypothesis. This hypothesis was then translated into an programmable event ordering and analyzed through simulations.

We assume that time is partitioned into discrete time intervals, all equal to the time it takes to receive/transmit one constant size cell(or segment)[55] of traffic from/to an input/output link. We assume that traffic comes into the system partitioned into fixed size units called cell – even though we might use the term packet referring to the same entity if not noted otherwise. In this time model we have structured the crossbar architecture shown in *Fig. 15* , so that servers take their scheduling decisions synchronized – but still independently. Backpressure state propagates instantaneously on reaction to servers decisions. Input servers take their unique for the current time-slot pick, at the beginning of the time-slot while output servers at the "middle" of the time-slot time – after all inputs have decided. So input servers might take under considerations the state of the system at the end of the previous scheduling time-slot plus the arrivals at the beginning of the current time-slot, while the output servers can take into consideration the state of the system at the end of the first half of the current time-slot, after all inputs

[55]If not noted otherwise, we will use both the terms packet and cell to name units of traffic, with constant size. Variable packets can be segmented into cells in the switch interface. But the architecture can handle variable size packets internally to the switch, as we shall comment in the finals sections.

have established their decisions. This is schematically shown in *Fig. 25* . Next we present the pseudo-code of the simulation:

Repeat
$\forall$ flows $i \in [1 \dots N^2]$ traffic_arrives(i)
$\forall$ inputs $i \in [1 \dots N]$ input_server_take_decision(i)
$\forall$ crosspoints $i \in [1 \dots N^2]$ backpressure_propagate_state(i)
$\forall$ outputs $i \in [1 \dots N]$ output_server_take_decision(i)
$\forall$ crosspoints $i \in [1 \dots N^2]$ backpressure_propagate_state(i)
modelTime++

Note here that in the simulation model, for the sake of simplicity, we have synchronized in some sense the servers decisions. Servers decision can be completely asynchronous in the real system – a property that makes possible the use of variable packets internally to the switch. A critical issue in a completely asynchronous system is the way that we handle race conditions on the backpressure state, in the view of any performance deficits that a viable solution might hide.

5.1.2 Goals-methods and environment and technical parameters

We have tested by simulations various parameters involved into the process of reaching the weighted max-min fairness. Our main goal is to examine the *degree* that weighted max-min fairness can be reached in the long run, the *time* it takes for the system to stabilize into the fair rates and the *reaction* of the system to marginal changes over configuration parameters – that is the time to reach a new equilibrium when flows become in/active or their degree of urgency changes. Ofcourse all these properties should be tested against every possible scenarios – against all potential *environment* parameters. Although we can not search the weights-space exhaustively ,to check all the possible configurations, we have made several experiments in the hope that these are indicative for the *absolute* system behavior.

We have also tested other features of the proposed architecture, such as latency and saturation throughput. In this end we compare our system with pure output queuing. To summarize our objectives these are:

- Long run convergence to weighted max-min fairness
- Time to reach equilibrium
- Reaction to marginal environment changes
- Saturation throughput and delay distribution

The *technical* parameters that are primarily involved with the scheduler, are the *type* of the servers used at the multiplexing points and the *size* of buffers used at the crosspoints – implicitly determines the backpressure threshold. We have examined three types of servers, WRR/WFQ, WF^2Q+ and VC. As we have seen in section 4.2.4 , if our model was fluid with GPS servers and instantaneously feedback, we will realize the weighted max-min fairness objective. It comes natural to test different approximations of the GPS model, in order to check the impact of the servers used on the final performance. This gives us a view of the complexity that we must pay to get most of the potential utility. In any case we primarily use WRR/WFQ servers, so in case that nothing else is mentioned, you can fairly assume, that this type of servers are simulated. The other central design-parameter that we examine is the size of the crosspoint buffer. The trade-off to decide the value for these buffers crosses most of the performance and complexity issues. When examining saturation throughput, we will test the alternative of using adaptive (as opposed to strictly-discriminative/weighted) servers at the inputs, like LRU, $FCFS$,LQF.

A final, rather subtle technical issue, has to do with initialization of the WRR/WFQ servers. In the simulation, all input and output servers clear any scheduling ties in the same way – that is they give preference to the same competitors when the conceptual, simulated discipline has no preference. This can result in many cases, to the observation of a phenomenon similar to this explained in section transient sources(section 4.4.6), where all servers take the same scheduling decisions.

5.2 Providing weighted max-min fairness

.

5.2.1 Constant flows state and The performance metric

Since weighted max-min fairness is better formalized and understood, during intervals when demand is constant, we have created configurations where all sessions (input-output pairs) are either continuously active or continuously inactive. Although you might argue that this scenario has no realistic counterpart for a long term, it can happen at times when congestion is very high (e.g. when the downstream nodes cannot accept any more traffic) particularly at backbone routers where flows are aggregated and partially occurs in any saturated input-buffered system. So this scenario exhibits how our architecture resolves big backlogs but more importantly, provides the opportunity to evaluate our architecture with the fairness objective. As we have made clear earlier we expect that our system will assign rates to sessions that will close to that of the weighted max-min fairness. Here we want to examine the impact of technical parameters – like the size of internal buffer space – in this procedure. The metric that we use to evaluate this hypothesis is called Percentage Absolute Weighted Distance:

$$PAWD_f^{system}(t_1,t_2) = 100 \cdot \frac{|fair\_service_f(t_1,t_2) - simulate\_service_f^{system}(t_1,t_2)|}{fair\_service_f(t_1,t_2)}$$

This metric represents the value of service(worth in cells), that our architecture will lag[56]behind weighted max-min fairness for each 100 cells of service, that the fair discipline provides. So if a flow that deserves a weighted max-min fair share of 0.5 has an $PAWD$ of 1, it means that we simulated $0.5 \cdot 500 \pm 1 \cdot \frac{250}{100}$ services, when this deserved 250.

We use this metric upon individual flows or for the flows aggregate in which case we present the average, median or max $PAWD$ value.

5.2.2 Equal weights

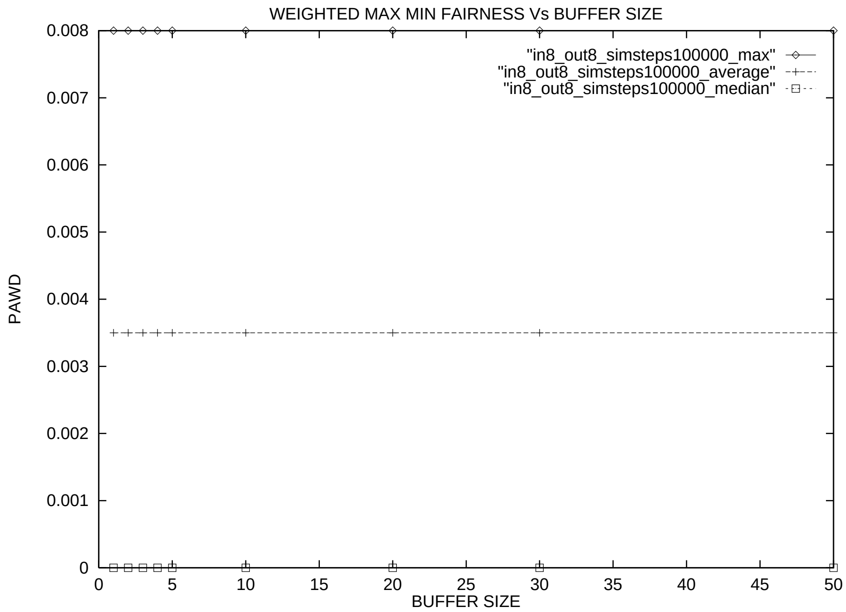


Figure 26: Average Max and median discrepancy from weighted max-min fairness over all flows.Configuration 1

All backlogged To start with, we present the easiest case where all flows have equal weights and all the flows are backlogged – in this case weighted max-min fairness reduces to max-min fair allocation on rates. Since the configuration implies that each server faces equal demand and all flows are treated as "equal", it is easy to verify that all flows deserve equal rates and the enforcement of max-min fairness is reduced to applying the minimum guaranteed rate to all flows.

[56]...or exceeds actually, but since under the weighted max-min fairness view, no overtake can be with no undertake we count both cases as discrepancies. You might say that (+3, -1,-1,-1) = 3 or = 0 or =6. Here we take the last position.

In *Fig. 26* we present the results that correspond from a configuration for an 8x8 switch, where all flows have equal weights and all flows are continuously demanding. We plot the max, median and average $PAWD$ values over all flows, against the crosspoint buffer size. We can see that all flows have received service very close to fair. All output links remain continuously backlogged, except for the first 7 time-slots when the coordination of the input servers, resulted in the loss of $64 - 1 + 2 \ldots + 8 = 28$ cycles(as explained in section transient behavior).

No flow is distanced from its fair service, no more than 0.008 service for every 100 fair services. It follows, that the service received by each flow, is never more than one cell behind its fair service. This is a quite pleasing result, since this is what the two dimension round robin proposals all want to accomplish. We can also see that this result does not depend on the crosspoint buffers size.

This is a good time, to clear something out, about the $PAWD$ metric that we use. To see how fast this equilibrium in rates is being reached we have created an other configuration, where we check the $PAWD$ metric, while the simulation proceeds. In this configuration we have used one cell crosspoint buffers, since these are adequate for reaching to the weighted max-min fair equilibrium. The result in *Fig. 27* are in log-log scale. We can see that after ~ 300 simulation steps, the weighted max-min fairness has been enforced in computational terms. The fact that the plot appears as a straight-line in the log-log scale, with angle almost -45 degree, reveals that:

$$PAWD(t) = \frac{1}{t}$$

This is the natural behavior of a system that takes some time to find an equilibrium but afterwords stabilizes in it. Since our metric counts the starting inaccuracies, it will need time to obliterate them and this is why the $PAWD$ has this form. This is exactly the case with the current example, where the only inaccuracies on rates occurred during the first $N - 1$(=7) time-slots. The system behaved in the optimal way that we can define for packet max-min fair system, after this starting interval. So we can say that the system converges into the equilibrium, with accuracy inversely proportional to time – if accuracy is defined with the $PAWD$ metric – but the equilibrium can be reached much more earlier in time – but is difficult to define exactly when.

Some backlogged One way to do thinks more difficult for the system, is to let some flows be inactive – that is to configure some flows to have zero demand. This complicates thinks, since in general the sum of weights in *some* input will be different from the sum in *some* output. This means that excess service will need to be redistributed and the procedure of reaching max-min fairness will not be so straightforward, as with the previous example.

Once again the system came very close to the max-min fairness objective, but

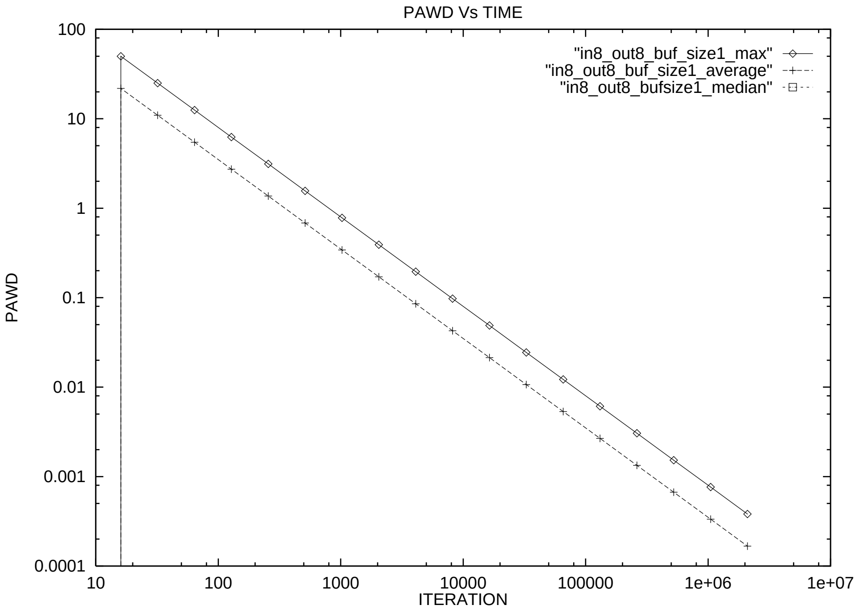


Figure 27: Average and Max discrepancy from weighted max-min fairness over all flows. Configuration 1

as we can see in *Fig. 28* , the crosspoints buffers size enter now into the game. The system will not apply the weighted max-min fair shares in the case that small buffers are used. The optimal value seems to be 3-4 cells, since there can be very small marginal gain to use additional buffer space. But in any case with 1-2 cells buffer, most flows receive rates very close to weighted max-min fairness.

In *Fig. 29* we plot the time evolution of aggregate $PAWD$ metrics. There are some worth noted properties in this figure. First as we can see the average $PAWD$ has again the form of a straight-line in the log-log scale. But the max discrepancy, although is generally decreasing, faces also some increasing periods. The median metric expresses the opposite behavior. This means that the discrepancy histogram of the flows, moves in time, from a long-tail densely populated organization to a delta function and vice-versa. A wild-guess could be, that while a flow or a group of flows looses calibration with weighted max-min fairness, other flows in the trajectory of this flow are obliged to do the same.

5.2.3 Different Weights

Introducing weights to the system, does not necessary imply the need for bandwidth re-distribution, since it can still hold that the sum of weights in all inputs and outputs is the same. This degenerate case is almost identical to the first case presented, where all flows had equal weight and were all backlogged[57].

[57]A technical issue is, that more time might be needed to perceive the system equilibrium through our metric – and most other metrics – when weights are introduced, specially for these flows that deserve very small rates

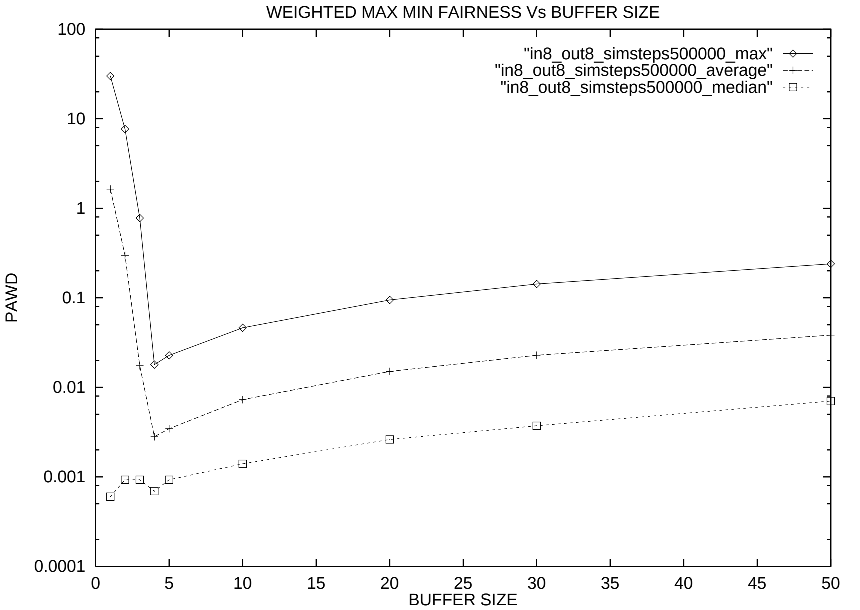


Figure 28: Average and Max discrepancy from weighted max-min fairness over all flows. Configuration2

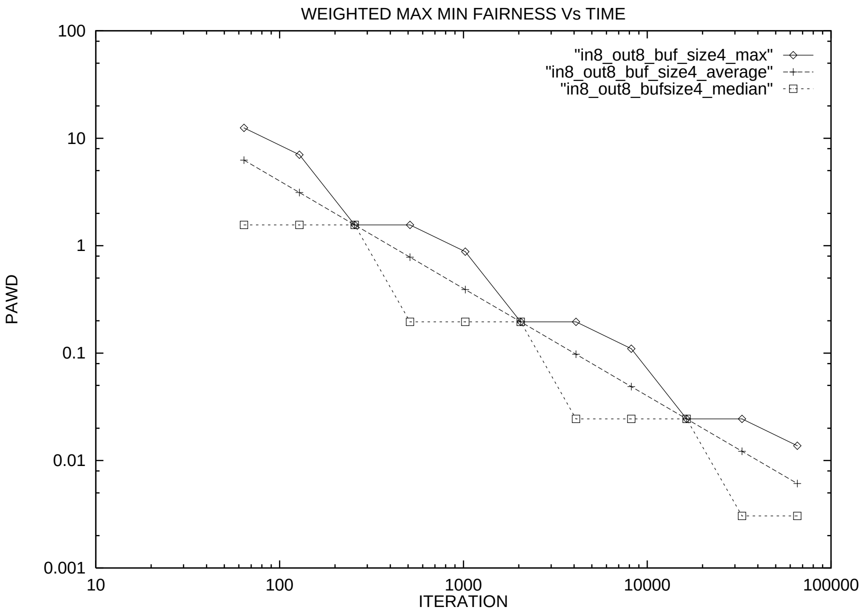


Figure 29: Average and Max discrepancy from weighted max-min fairness over all flows over time.Configuration2. buffer size 4

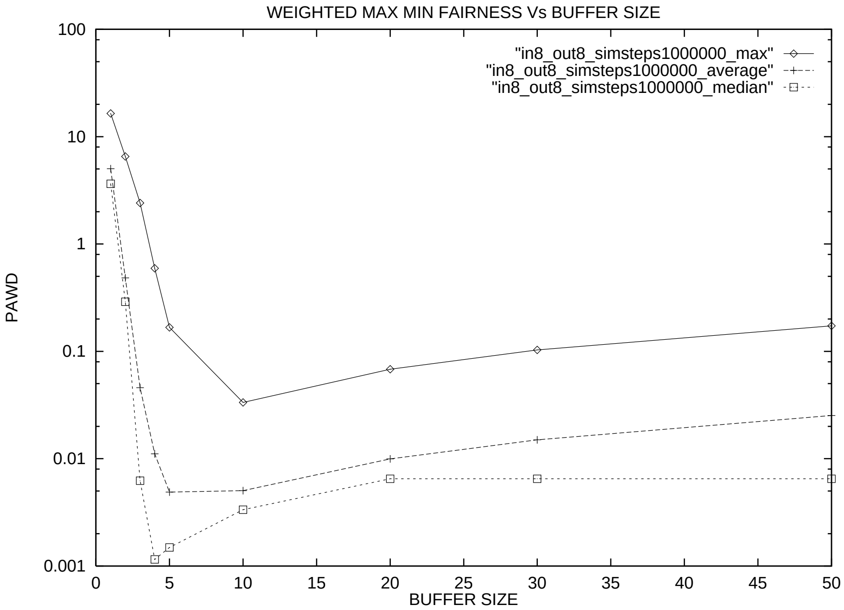


Figure 30: Average and Max and Median discrepancy from weighted max-min fairness over all flows. Uniform distribution of weights. Configuration3

In the general case of-course, this will not be the case. Weights can be arbitrary distributed on flows. This distribution can be expected to be uniform or to exhibit preference on certain links(hot-spots). The latter case has special interest for us, since the abnormal distribution of weights conceals the need for redistribution of great amounts of bandwidth.

We begin with uniform-random distribution of weights on flows. All flows are active and each has a weight randomly selected from a uniform distribution, in the interval 1 to 1000. In *Fig. 30* we examine the degree that the system reaches weighted max-min fairness, for various sizes of crosspoint buffers. Once again the weighted max min allocation has been enforced, almost with the same accuracy as with the previous runs. Buffers again play a critical role. For buffer size one there are flows, that receive only receive $90 - 110$ percent of their fair service. This interval is reduced to 0.1 services, when the buffer size is equal to 5 cells with the tension to reduce to zero for greater buffer sizes and more time-slots.[58]

	CLASS A	CLASS B
Average PAWD	5.696	0.294
Max PAWD	27.86	4.64

Table 1: Distribution of unfairness to classes for one cells, crosspoints buffers.

Figure *Fig. 31* , is an other experiment, where one quarter of the flows are

[58]When the buffer size is close to ten cells, a full buffer at the end of the a one million time-slots simulation for a flow deserving 0.2 of the links capacity, can appear as a discrepancy of 0.05 percent.

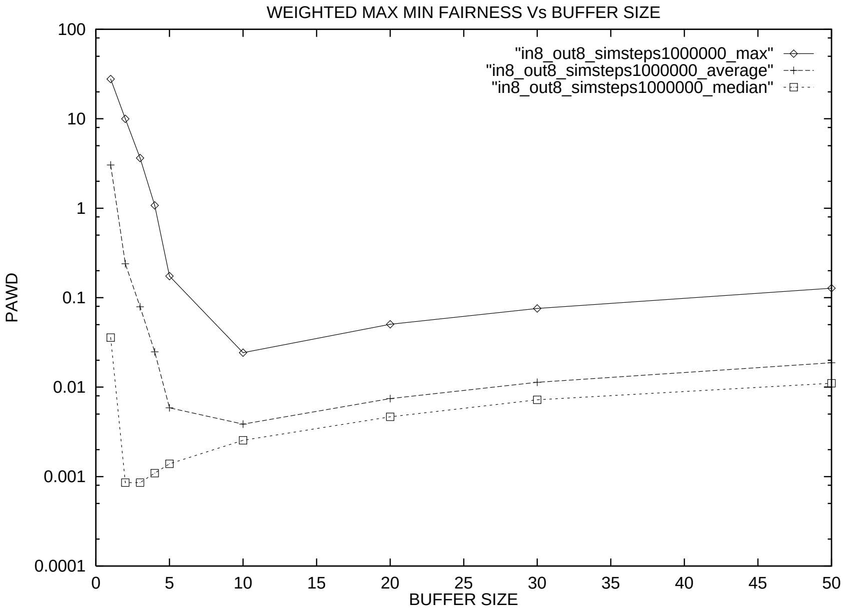


Figure 31: Average and Max and Median discrepancy from weighted max-min fairness over all flows. Two classes of flows. Configuration4

inactive. Active flows belong either to class A with weight near 1 or to B with weight near 20. Flows belong to one of the A or B classes with equal probability.

In table *Table 1* , we have computed the average and max $PAWD$ metrics for each class independently, under one buffer size configuration. Class A flows have higher $PAWD$. This primarily because these flows have to utilize excess bandwidth to reach their fair-share – which in many cases is much higher than the share deserved if the input and/or the output servers where working in isolation.

Hot-Spots/irregular-topological-distribution-for-demand In the next experiments we examine hot-spots cases, that is links that carry flows with high degree of urgency. This view represents a twofold interest for us. For one, it is widely known that traffic on Internet is anything but regular[59]. Aggregate requests for service follow, highly dense topological(spatial) and chronological(temporal) locality, that can translate in periods of great demand, for some output links. On the other hand since we want the sum of weights in input and outputs to exhibit irregular distribution, the congested links view is a good opportunity to enforce it .Since the flows sharing these congested links, will have to be fair in their inter-relationship(s), it means that excess bandwidth on the other links used by these flows will need to be re-distributed.

Here we consider a highly irregular case where the weight of each flow is determined by the output that this flow wants to go. If we enumerate the outputs from

[59]Request of web users to sites, is known to follow the so called Zipf distribution, that claims that the most popular site is *much more*. The i in the popularity rank, gets only the $\frac{1}{i} - th$ of the population.

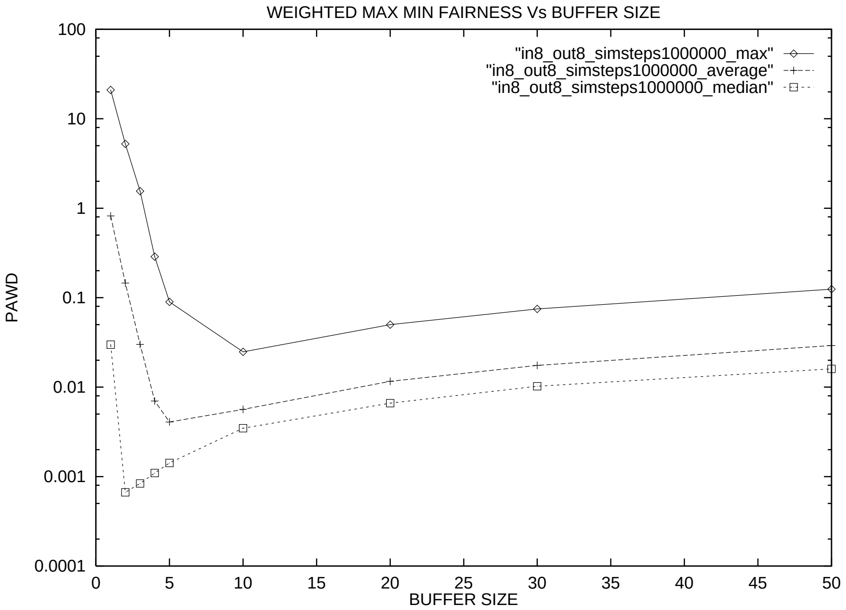


Figure 32: Average and Max and Median discrepancy from weighted max-min fairness over all flows. Hot spots case. Configuration5

1 to N, then each flow has a weight analogous to the square of the output that it belongs. In *Fig. 32* we can see that although we tried, we did not actually made thinks much more difficult for the system. Again the same behavior is experienced.

Conclusions We have restricted the number of the results presented in this section, but the same behavior has been experienced for all the configurations that we have tested. Before we turn our attention on the saturation throughput, we will present a couple of figures for greater scales, that is switches with more inputs. We expect that configurations with more competitors, will in general, increase the buffer requirements.

Although in *Fig. 33* we can see a slightly worst behavior compared to smaller switches, this is because the system needs more time to obliterate the starting inaccuracies and additionally because the smaller rates that flows generally deserve, result in slower convergence of the $PAWD$ metric – and most other metrics. In section 10 you can find additional plots, for switches of greater size.

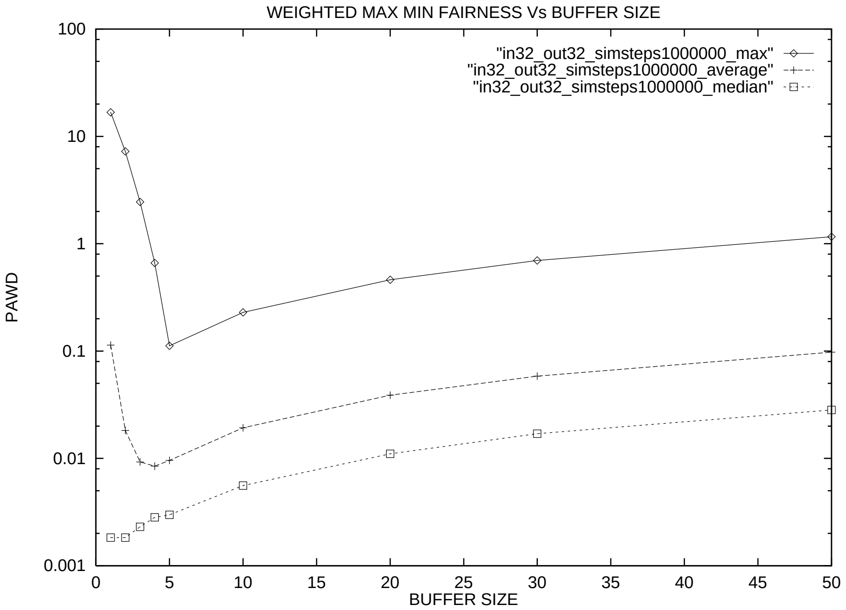


Figure 33: Average and Max and Median discrepancy from weighted max-min fairness over all flows for a 32x32 switch. Two classes of weights. Configuration6

5.3 System (re)calibration time-experimental methods

So far so good. The system seems to converge to the weighted max-min fairness. The $PAWD$ metric closes , if adequate *buffering* and *time* is available, for all the configurations that we have checked. But with this metric we have not been able to tell the convergence point. In this section we discuss how to recognize the convergence and having established the means, we apply them to odd configurations that in our view represent the worst case. So we will try to experimentally-computationally find the time needed to enter the weight max-min fair state, starting from the scratch or after condition changes[60]. But here-in contrast with the previous section-we will try to find better methods to recognize the weighted max-min fairness other than the $PAWD$ metric that simply computed the degree of convergence in the long run. Here we need tools to recognize thinks like- *now* we have fairness or *now* we do not-suppressing now into the moment as much as possible. It appears that we would like to use the notion of momentous rate, a rather vague and unrealistic tool when dealing with packets instead of fluids.

Momentous Rate Momentous rate is defined as the $lim_{t->t_o}(\frac{S(t)-S(t_o)}{t-t_o})$. This definition requires that time is a continuous variable, and the service function $S(t)$ is continuous from both sides at t_o. In our model time is a discrete variable, so we cannot use this notion. Even if we accept that $lim_{t->t_o} = t_{o+1}$ (or t_{o-1}), then even for a flow that receives a constant rate – that is constant number of services every T time slots – the momentous rate will range from zero to one. We have to define a a rate variable that will capture in the shorter possible interval, the actual rate received by a flow. But this interval certainly depends on the actual rate received by the flow and additionally on the burstiness of this service history. There are at least four methods to compute a flows rate:

1. Static window of size w, ($R_w^{st}\text{\_}^W$)
2. Sliding window of size w ($R_w^{sl}\text{\_}^W$)
3. Interval for s services ($R_s^{st}\text{\_}^S$)
4. Interval for s sliding services ($R_s^{sl}\text{\_}^S$)

The first method, partitions time into fixed intervals of length w. Then in each partition w_i, the rate of a flow is defined to be $R_w^{st}\text{\_}^W(t_i) = \frac{S(t_{(i-1)\cdot w}, t_{i\cdot w})}{w}$ [61]. The second method looks much like the first but differs in that for each time slot, the window rate is re-computed for the w previous time-slots. That is $R_w^{sl}\text{\_}^W(t_i) = \frac{S(t_i, t_i - w)}{w}$. Applying a sliding window, will result in smoother plots,

[60] the system calibration from the start is an environmental change where all flows becomes active

[61] Even with this simplistic formulation there is the problem on which partition to assign this rate computation, on w_i or on w_{i+1}.It is more intuitive to assign this rate on w_i

when the actual rate changes. The third and fourth methods stand upon the following observation:

$$Average\_rate = \frac{1}{Average\_Interservice\_Delay}$$

With the third method we partition the series of f's service instances – instead of time units – into disjoint groups of size s and we compute the rate as: $R_s^{st\_S}(t_{s\cdot i}) = \frac{s}{t_{s\cdot i}-t_{s\cdot(i-1)}}$, that is as the phliko of the number of services (s), to the time it took to receive these services. The last method re-computes this phliko whenever a new service of f is encountered, so $R^{sl\_S}(t_{s_i}) = \frac{s}{t_{s_i}-t_{s_{i-s}}}$.

Certainly the last two metrics are more "rate aware", since these take into account the actual rate of each flow when plotting rate diagrams. Even if a flow has no services in a time interval T, this methods will not necessary plot zero rate, but instead will wait until the next service to take a position. The only problem is that this metric might misjudge in the case that a flow which received services in the past, receives no more services in the future in which case the old rate assignment will be used.

These will be the tools that we will use to estimate the time to converge to new rates. Small values for w or s are what we want, to observe fast rate changes, but might be inappropriate in the case of bursty-non smooth-service allocation. In many cases we simply plot the difference of the aggregate service that the weighted max-min fairness would provide with the service provided by our system. Clearly when this plot becomes horizontal – parallel to the time axis(has zero first order derivative)–, then the fair and system rates are equal.

5.3.1 A basic configuration scheme

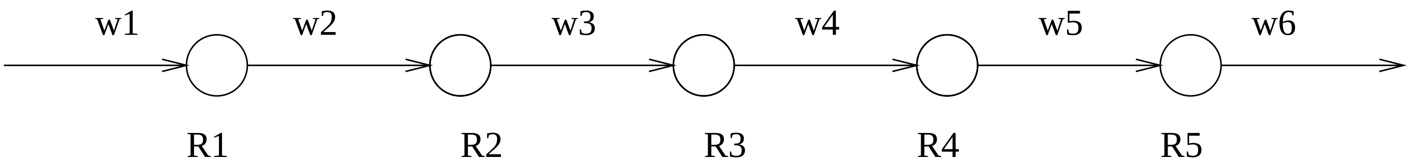


Figure 34: Five resources are being shared by six flows. Each resource is being utilized by each these flows "touching" them. $w1 \geq w2 \geq w3 \geq w4 \geq w5 \geq w6$.

In a previous section, dedicated to algorithms that compute the weighted max-min fair shares of flows, we show that weighted max-min fairness requires all flows to confront at least one bottleneck link[62], on which link the respective flows must place continuous demand. The first most congested flow(f^1) should stabilize in this state, before the other flows that belong in f^1's trajectory, become steady in their fair share. These other flows, might reach their fair shares in parallel with this flow, but might change "direction" after this flow has stabilized. Anyway we

[62] Since we make the assumption that all active flows have infinite demand, the source will never be the bottleneck

expect, the sequence that flows reach their steady state, to follow the order that flows are labeled as bottlenecks, in the weighted max-min fair algorithm. We know that this algorithm terminates in steps proportional to the number of bottleneck links B, so one first rough estimation of the time to converge is $O(B)$ – which in our crossbar topology case is at maximum $O(2 \cdot N)$.

We use as basic example, the simple configurations presented in *Fig. 34* . This configuration isolates one single chain of interactions and can be as large as $2 \cdot N$, occurring as much as $2 \cdot N$ bottleneck links. Therefore we can use this example, without actually losing contact with the general case. In the crossbar topology, we can map this configuration, into a series of flows $i \in [1 \ldots 2 \cdot N]$ using input $i/2 \in [0 \ldots N-1]$ and output $i/2 + i \bmod 2 \in [0 \ldots N-1]$, as shown in *Fig. 35* .

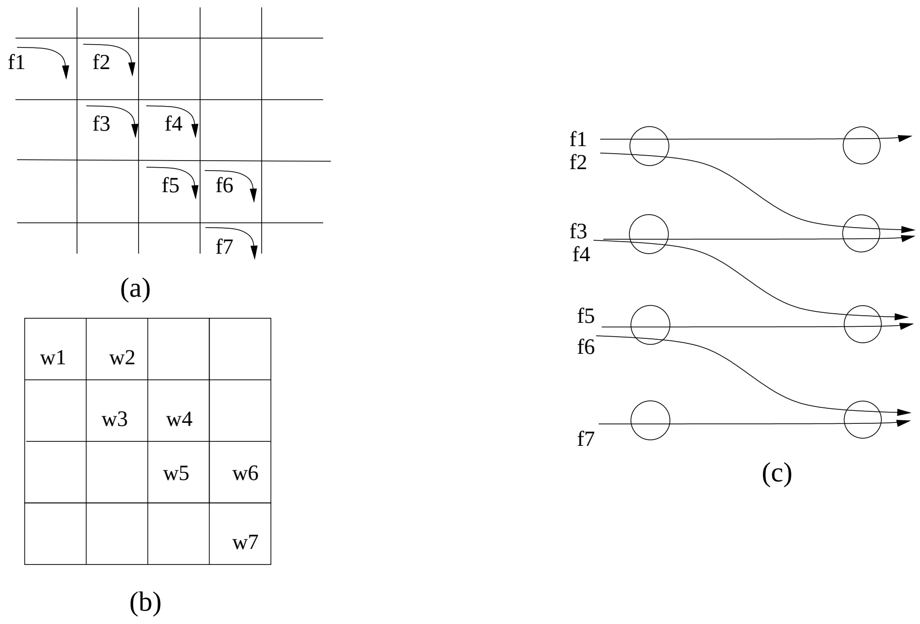


Figure 35: One possible map of *Fig. 34* , into the crossbar topology(a). The weights configuration(b) and the bipartite graph of resources-requests(c)

5.3.2 System calibration time, starting from the scratch

If all flows have the same congestion index, then all flows will be able to find their fair shares in parallel. Since no flow is actually affected by the others behavior, the time to reach the weighted max-min fair shares will be minimal. Interactions start to occur, when the congestion index of flows are diverged. The first idea is to use decreasing weights for the interacting flows in *Fig. 35* , so that the first flow – with the higher weight – will have a trajectory, that comprises all other flows.

Fig. 36 , plots the difference between the ideal fair service and the actual service supplied by $WRFQ$, for the configuration shown in *Fig. 34* . It exbounds the fact, that flows resident in more depth of this chain, take more time to reach their fair shares. Flows can be grouped in couples, according to their stabilization behavior –

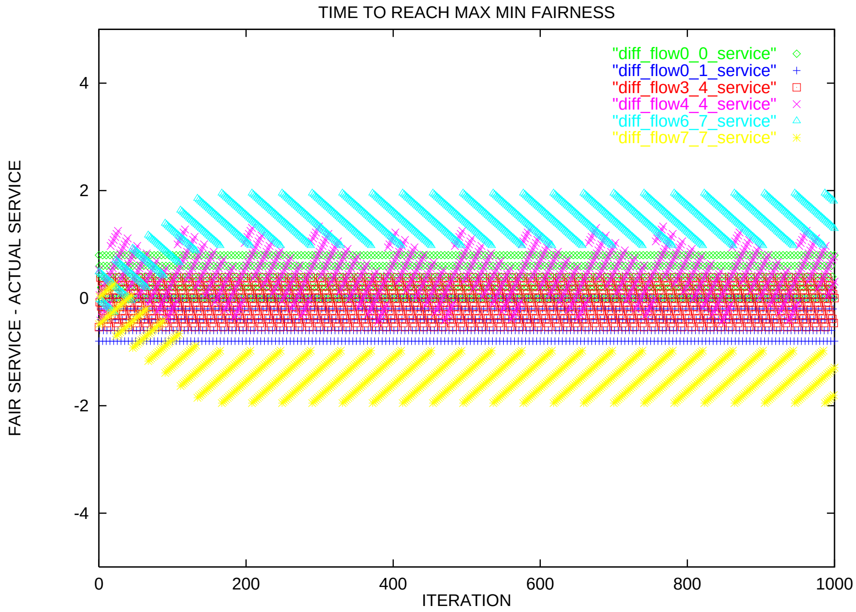


Figure 36: The evolution of weighted max-min fairness through time, for a 8x8 switch. Flows have the same configuration like in *Fig. 34*

these are neighbor flows, with equal utility at the final equilibrium. We can see that couple's in the end of the chain, can take us much as twice the time to stabilize into their fair shares, than flows in the middle. But these(more-distant) flows, restore only a small amount of bandwidth. In *Fig. 36* , we can see that flows with id 6-7 and 7-7 exceed and lag from their fair shares respectively, only for 2-3 cells at maximum. The system has not strictly stabilized before lets say iteration 200, but even before this point the system does not behave *unfairly*. It seems appropriate to say, that these flows are affected by previous flows rates, but not that much[63].

Analysis In the section for flows trajectories(section 4.4.3), we show that a chain of actually interacting flows, will consist of flows with non-increasing congestion index. During the stabilization process, there is number of sub-small-regional-max-min-equilibria that actually occur. We want them to reform, to the final holistic equilibrium of weighted max-min fairness. So actually, what applies in the case of events, that reform a starting equilibrium A, to a new B, applies here as well. There we claimed, that in order for a flow to increase its rate, there should be a flow in the previous – before the event – bottleneck of this flow, that decreased its rate. If a flow decreases its rate, then there should be flow with equal or less utility – before the event again – that increases its rate. Does this configuration has these properties?.

It certainly has the first property, that the interaction chain is in decreasing congestion index. But flow 3 and flow 4, could for example have different temporal

[63]This is again the point, that the rate(=degree) of the unfairness and the duration of unfairness, are inversely proportional.

bottlenecks – referring to sub-equilibria –, in which case the decrease of 3 does not necessary imply an increase of 4, since 4's neighbors in its bottleneck link has not been affected[64]. In this case the decrease of flow 3, will simply appear as under-utilization for the respective link, since flow-id 4 cannot utilize the additional rate. When the "simulation" starts this means, that if flow 3 had not reached its fair share from the beginning – since the crosspoint buffers delayed this enforcement or because the flows that affect 3 have not reached their fair share for the same reasons – and was actually getting, more than this deserved, this will have small impact for the stabilization of the other flows.

0/0.00/0.00	**16384/1.00/0.66**	**0/0.00/0.00**	**0/0.00/0.00**
0/0.00/0.00	**8192/0.66/0.33**	**4096/0.33/0.66**	**0/0.00/0.00**
0/0.00/0.00	**0/0.00/0.00**	**2048/0.66/0.33**	**1048/0.33/0.66**
0/0.00/0.00	**0/0.00/0.00**	**0/0.00/0.00**	**512/0.66/0.33**

Table 2: Weights of flows/rates, before congestion wave reaches flows/rate after congestion waves reaches, flows. Analogy c =2. The wave reaches flow 2 with weight 16384, when its buffer fills. The same for flow 3. Flow 4, is reached when 3 fills his buffer and flow 5, when 4 empties his buffer, etc.

Source/Input	**Output/Input**	**Source/Source**	**Source/Source**
Source/Source	**Output/Input**	**Output/Input**	**Source/Source**
Source/Source	**Source/Source**	**Output/Input**	**Output/Input**
Source/Source	**Source/Source**	**Source/Source**	**Output/Output**

Table 3: Temporal/final bottlenecks for the flows in *Table 2* before/after the final-dominant congestion reaches them – or equivalently with flow 1 active/inactive.

If we want all properties to hold, then in addition to the decreasing weights, some analogy must hold between flows weights. This analogy must ensure, that each flow faces as bottleneck, strictly the link that this shares with the previous flow in the interaction chain. Then a change in $i-th$ flow's rate, will certainly affect flow $i+1$. This must hold at least for those flows, that are supposed to increase their rates after the change event – which reduces in our case, into a neighbor reaching the fair share. In addition, if we demand the decrease of flow i to be completely absorbed by flow $i+1$, then $w_{i-1}/w_i \leq w_{i+1}/w_{i+2}$. A basic scenario that complies with the above, is one in which: $\forall i : (w_i/w_{i+1} = constant)$. This is realized by the crossbar, as shown in *Table 2* and *Table 3* .

In figure *Fig. 37* , we can see that during the stabilization process the rates of the flows are not close at all, to their fair rates. [65]. This happens because,

[64]these can only be affected afterwords – by 4 and not affect 4

[65]This effect happens only when flow 1(0_0) is absent(inactive). If this flow is active, then the phenomenon will not occur at all, since the starting rates (during charging the buffers, before the congestion

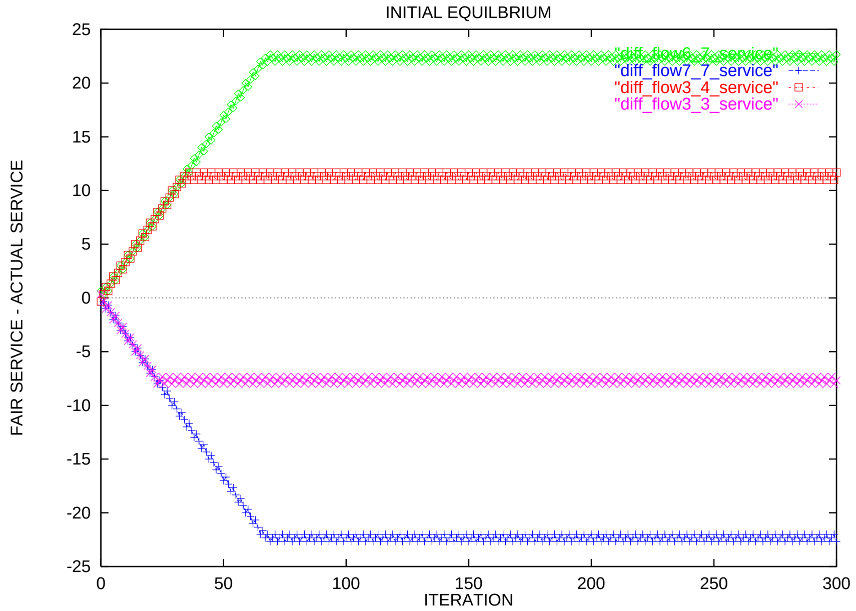


Figure 37: The evolution of weighted max-min fairness through time 8x8 switch. Flows have have the same configuration like in *Fig. 34* (except that flow 0_0(flow 1) is inactive) and in addition $\forall \quad i \quad (w_i/w_{i+1} = 2)$

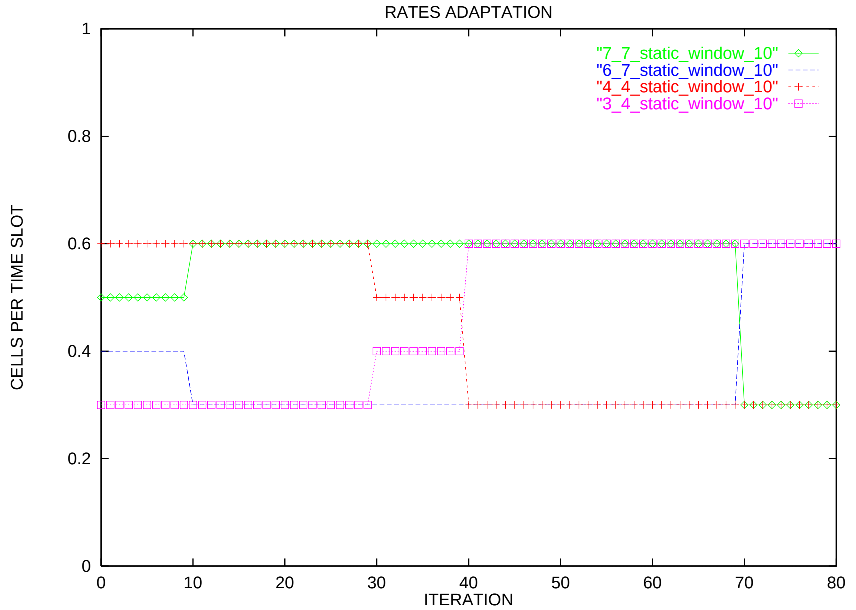


Figure 38: The evolution of weighted max-min fairness through time 8x8 switch, for the same configuration like in *Fig. 37* . The rates are computed with the constant window method and w = 10.

before the buffers are filled and the congestion stops flows from sending data, the input fairness controls the rates. First some highly congested flow (i) must drop its rate due to backpressure, so that the next flow using the same input ($i + 1$), can take advantage and increase its fair share (taking advantage of the absence of i, instantaneously). Then $i+2$, will no longer take advantage of the absence of $i+1$ at the shared output and decrease its fair rate, which will propagate to the next flow, only after the crosspoint buffer switch states(from full to empty or the opposite).

It appears that flows reducing their rate, change bottlenecks and this takes time, since it cannot happen but only serially in a buffered system(fluid or packet). *Table 3* , that marks the bottleneck of the flows, before the congestion wave reaches them and after[66], shows that while the congestion wave propagates, flows change bottlenecks and that those flows that are supposed to decrease rates, when the wave reaches them, faced temporal-bottlenecks at the link shared with the neighbor flow, of greater congestion index(sharing output, in our case) – the prerequisite in order to be affected, according to Lemma 2.

The parameters that control the worst-case duration for the trip of congestion waves There are two parameters affecting the delay of the change propagation, in the above process. The first is the constant in the analogy of the weights and the second is the size of the crosspoints buffers.

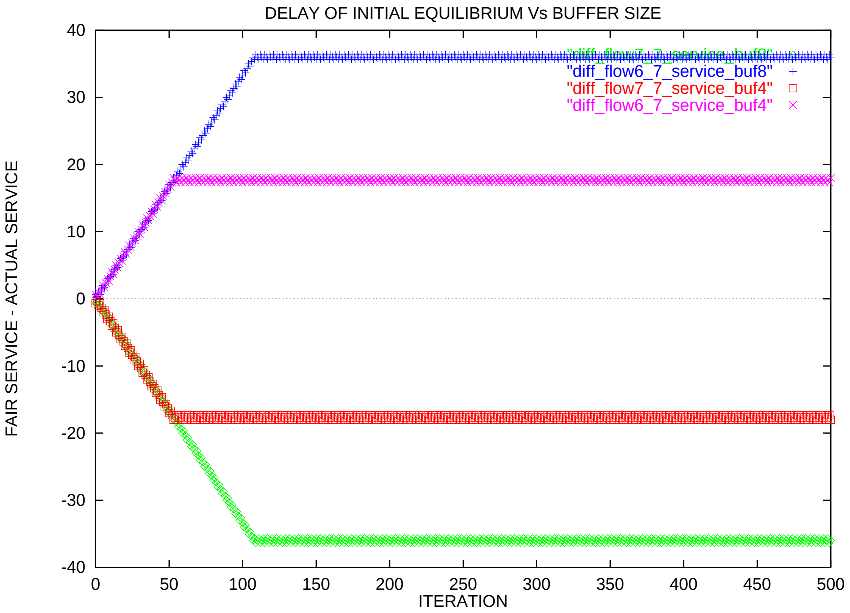


Figure 39: The impact of the buffers size in the delay of fairness enforcement.

wave has reached them) equals to the final rates. In other words flows do not have to change bottlenecks, when the previous flows reaches its fair share. Flow 0_0 controls, weather input or output fairness will dominate.

[66]Before the congestion reaches them, flows have the same bottlenecks with the equilibrium in which, flow 1 is active and has weight 16843

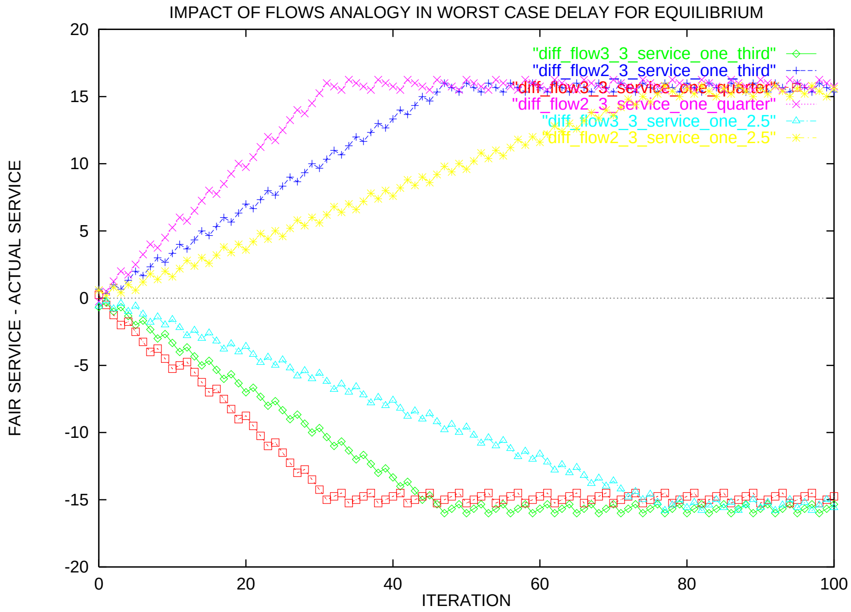


Figure 40: The impact of the analogy of weights to maximum delay

Plot *Fig. 39* , exhibits the effect of the crosspoint buffers size, to the delay in reaching the equilibrium. Here the first diagram, is for a configuration with 4 cells capacity in the the crosspoint buffers and the second for 8 cells. As we can see the plot is consistent with our description of the stabilization process; double the buffers $\rightarrow$ doubles the time to reach equilibrium.

In *Fig. 40* , we depict the effect of the analogy between flows weights, in reaching weighted max-min fairness. We can see that while the analogy of weights closes to one, the delay of reaching weighted max-min fairness increases. This is somehow natural. We have argued, that those flows that decrease their rates, will have to change the state of the crosspoint buffer so that the change-wave propagates. The rate with which these flows change states – empty or fill some crosspoint buffer –, is the (temporal)fair service of this flows, at the less bottleneck resource that these use. This rate depends on the analogy of weights and can be as small as $0.5 + e, e \rightarrow 0$, if we want the phenomenon to happen at all, in the scenario that we examine – if this rate is 0.5, then there is no delay in the stabilization, since all flows must have equal weights. But no matter the analogy, the maximum distance from the fair service, when measured in bits, is almost constant. This holds because if w_i/w_{i+1} influences proportionally the delay of the stabilization, again w_{i+1}/w_{i+2} influences inversely proportionally, the unfairness experienced – in terms of rates–, while the stabilization has not yet occurred. Since we used constant analogy between the weights, their product (=fair_service - actual_service) is roughly independent of the analogy.

For the configuration presented in *Fig. 37* and *Fig. 38* $w_i/w_{i+1} = 2$. The estimation for the delay in the enforcement of weighted max-min fairness, that we computed in the flows trajectories section(section 4.4.3 , section 4.4.4)

, is N times the time to fill a buffer of size B, when this buffer fills up with rate $r\_in = \frac{w_i}{w_i + w_{i-1}}$ and drains with rate $r\_out = \frac{w_i}{w_i + w_{i+1}}$. Starting with the equality $\Delta B = (r\_in - r\_out) \cdot t$ and substituting $w_{i+1} = 4 \cdot w_{i-1}$, $w_i = 2 \cdot w_{i-1}$, the estimate solves into $t = \frac{N \cdot B}{0.33}$.[67]. For $N = 8$ and $B = 4$ the estimate is $32 \cdot 0.33 = 96 iterations.$. The actual value that we observe in *Fig. 36* is 70. This is not inconsistent, since actually in this configuration, there are six buffers instead of eight, that change state. With 6 instead of 8 the estimate is 72 simulation steps.

Summarizing It seems that the existence of chains like the one that we examined above, affects the time to reach the equilibrium. Although for other configurations, this time can be greater – compared to the scenario that we presented above, which involved a maximum chain depth equal to $2 \cdot N$ and completely serial, big changes – it cannot be more unfair when measured in bits, since the unfairness degree(measured in rate) and the time that it lasts, are inversely proportional. We showed that the parameters involved are: the size of the internal buffers, the depth of the "alive"- interaction chains and the analogies of the weights. The type of the servers used at the contention points, does not seem to affect this worst case process in an obvious manner. We now turn on results for a similar problem(if not equivalent), which is the time to recalibrate into equilibrium, after a change in the weights or state. A difference here, is that if we use couples of dual events – first a flow f becomes inactive and afterwords active again – if the servers punish other flows for utilizing the excess bandwidth of flow f during its inactive period, like the VC method does, then unfair results can and will occur.

[67] In the general case with analogy c(instead of 2), the rate that the buffers change state is $\frac{2 \cdot c}{(c+1) \cdot (c+2)}$.

5.3.3 One single event

Since we know the notion of flows trajectories, we know the maximum total affect, of any change in a flows demand(weight - active/inactive). We know that the worst case will be if we affect the most congested flow, or a flow in this neighbor(if the change is an increase of weights). In this section, we will present some examples of this worst-case behavior and some more “common”/regular changes.

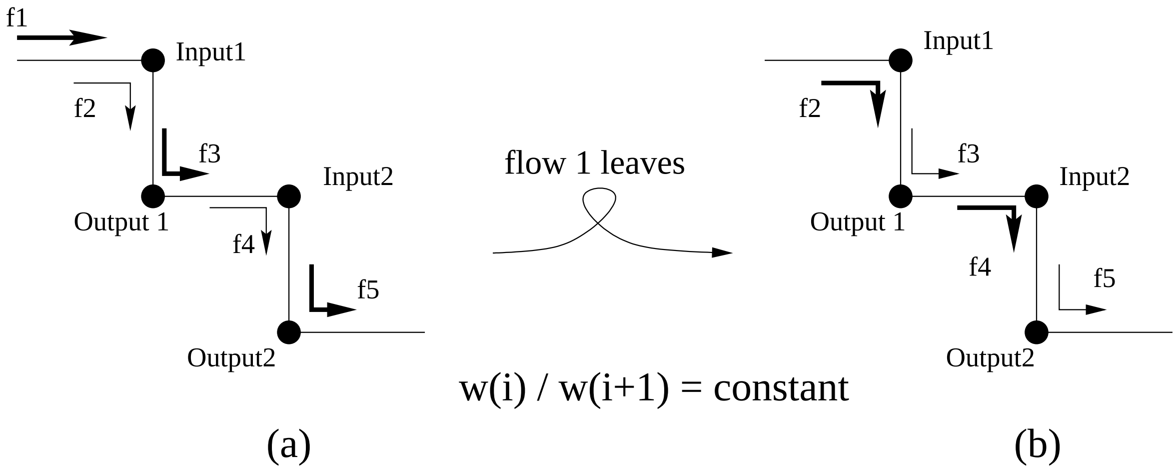


Figure 41: Five resources are being shared by six flows. Each resource is being utilized by these flows “touching” it. $w1 \geq w2 \geq w3 \geq w4 \geq w5 \geq w6$.

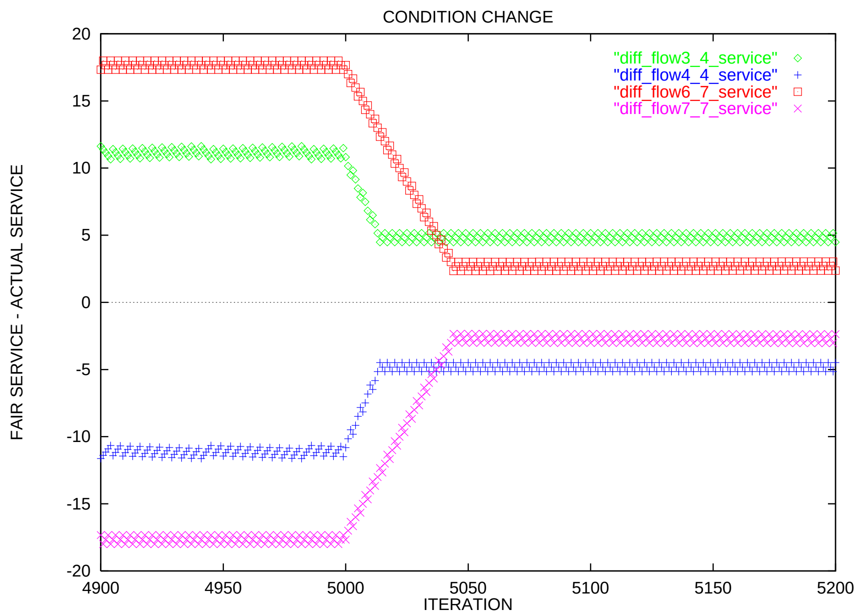


Figure 42: This plot present the results for the scenario in *Fig. 41* , for 8x8 switch with crosspoint buffers size 4 and constant analogy $\frac{w_i}{w_{i+1}} = 1/2$

Figure *Fig. 41* presents an instance of this worst case behavior. Flows have the same properties like that in *Fig. 35* . The most congested index(that is flow

1), change state from active to inactive, at time_slot 5000. Figure *Fig. 42* present the results, which are basically homogeneous with that presented before, for the system calibration process. The results that applied there, about the impact of the crosspoint buffers size, the weights analogies and the length of the interaction chain, apply here as well.

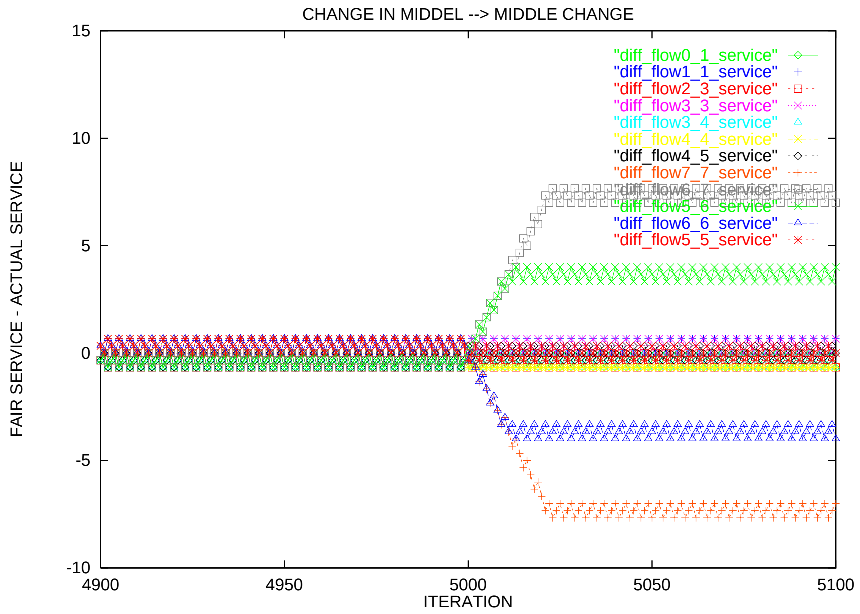


Figure 43: In this plot we have a scenario similar to *Fig. 41* but with event difference that the flow that uses input 4 and output 4 becomes inactive(instead of flow 1), affecting only flows with lower congestion index

To see that a change of a flows state, can only affect flow's that are less congested, in *Fig. 43* we changed the state of a middle flow. We can see that this change, did only affect the subsequent flows in the congestion series.

For clarity we simulated a configuration(*Fig. 45*) which correspond to the scenario, where a flow with small congestion index, can affect the distribution of rates for all flows – since this is a neighbor of a highly congested flow. Theorem 4.11 claimed that this can actually happen. The configuration that realizes this scenario is shown in *Fig. 44* .

Finally we present two cases with more active flows. First we have augmented the basic scheme shown so far(*Fig. 35*), with low priority flows in the empty positions of the table. So all flows are active. Flow 0_0 becomes active at iteration 5000 *Fig. 46* .

The last case represents a more regular scenario. In this example all flows are active, but there is nothing special about the configuration[68]. We simply make a high priority flow inactive and watch the changes *Fig. 48* , *Fig. 49* .

[68]weights follows the hot spot distribution presented in the a previous section

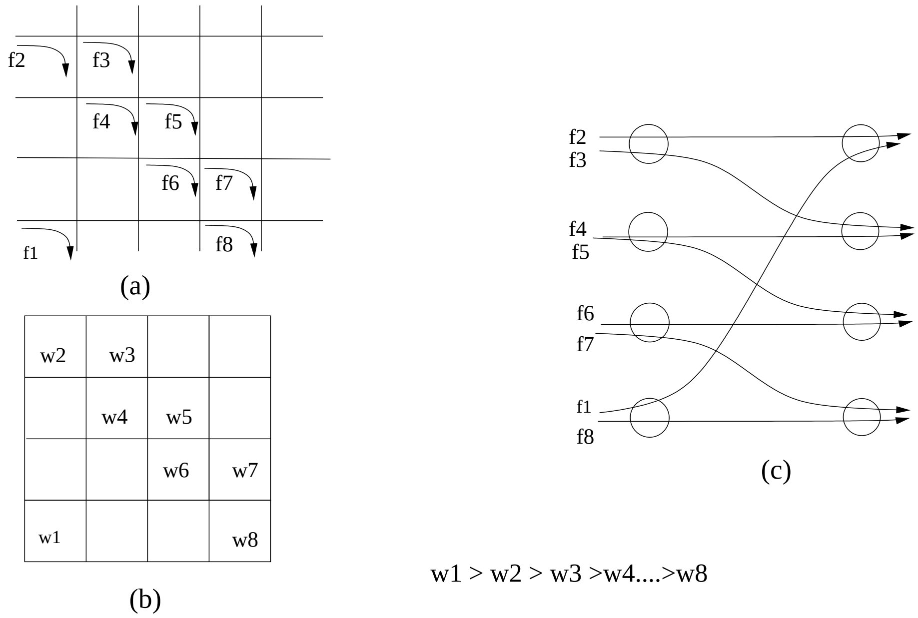


Figure 44: Again all flows are indexed in decreasing weight order. Flow with id 8, is the less congested flow. After the change of this flows weight(upgrade), all flows rates changed

5.3.4 Two events

Here in this section, we will try to see if two(or more) events – overlapping in some way – can have different impact on the system than one single event. Certainly two events are simply one event plus an other one and if we accept this view holistic-ally, then we might argue that nothing different can happen. Here we will simply evaluate this hypothesis. The opposite argument is, that these two events may conflict, or may overlap in the enforcement of the new equilibrium. Before continuing we note that with two events, there are four equilibrium under consideration. The starting one (A), the one that the system would stabilize in, if only the first(arbitrary ordering) event occurred (B), the same for the second event (C) and the final equilibrium (D),after both events occur. Since the problem is quite complicated , without loss of generality, we will stick on configurations similar to that presented *Fig. 35* .

We have described the set of flows that will react in response of a single event. These flows g must belong to the interaction graph, of the flow that changed f_1 and additionally these should be reached from f_1 or one of f_1's neighbors, through a path(chain) of flows(interactions), with decreasing congestion index under equilibrium A. So let the event E_1 occur. If the first event completes before the time of the second event(E_2), then the argument that two is one plus one holds and there is nothing to add here. The problem may occur if the two events, or to be more precise the duration of the intermediate periods caused by the events, overlap. There are plenty of possible scenarios here – which we will try to reduce.

If the flows that caused the events belong on different interaction graphs, then

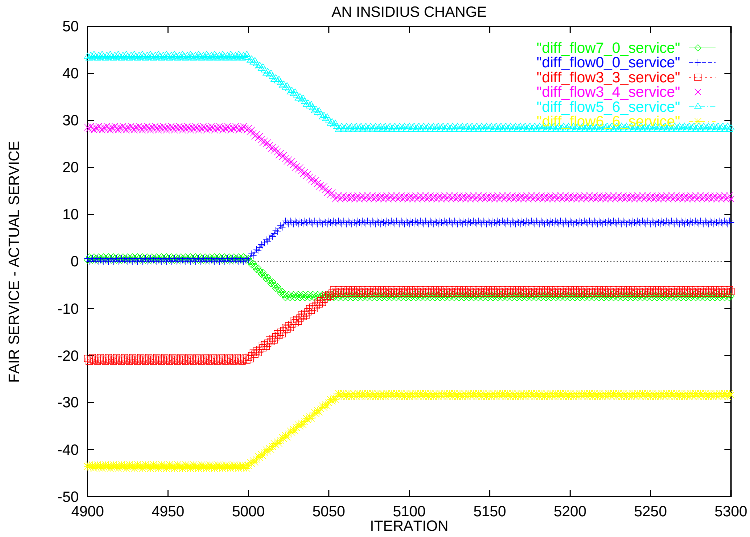


Figure 45: Flow with id 8 changes weight and affects all other flows

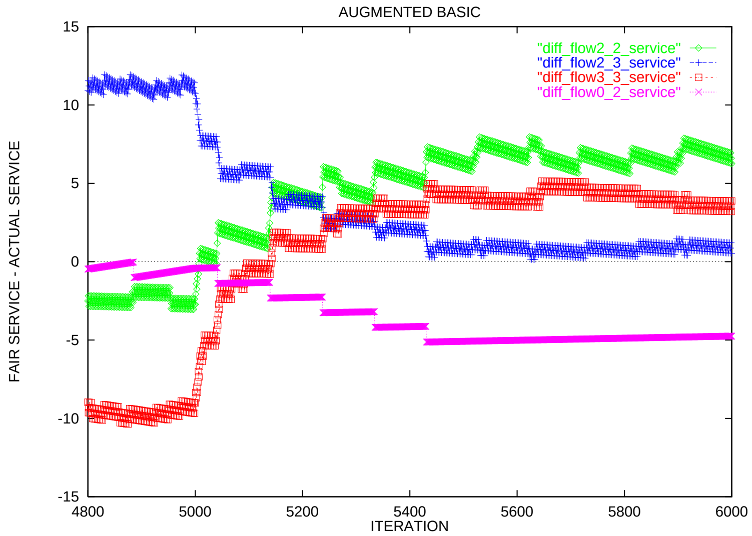


Figure 46: 4x4 switch like previous examples with the addition of low priority flows, in the position where previous nothing

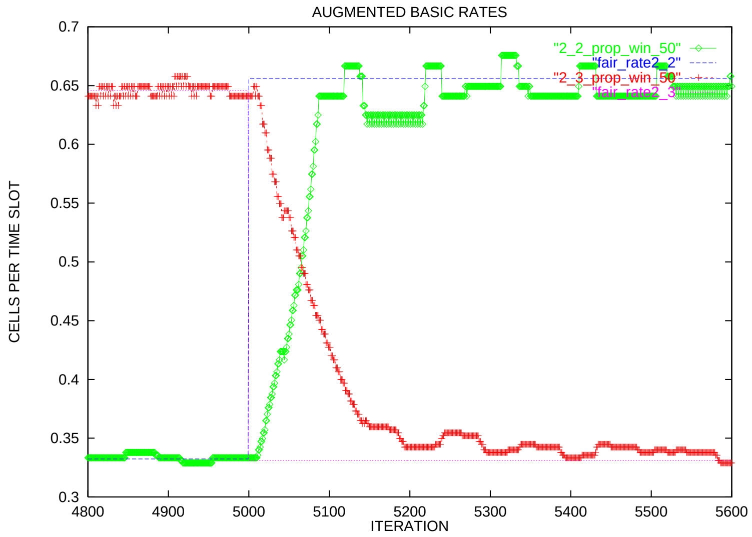


Figure 47: 4x4 switch like previous examples with the addition of low priority flows, in the position where previous nothing. Rates plot

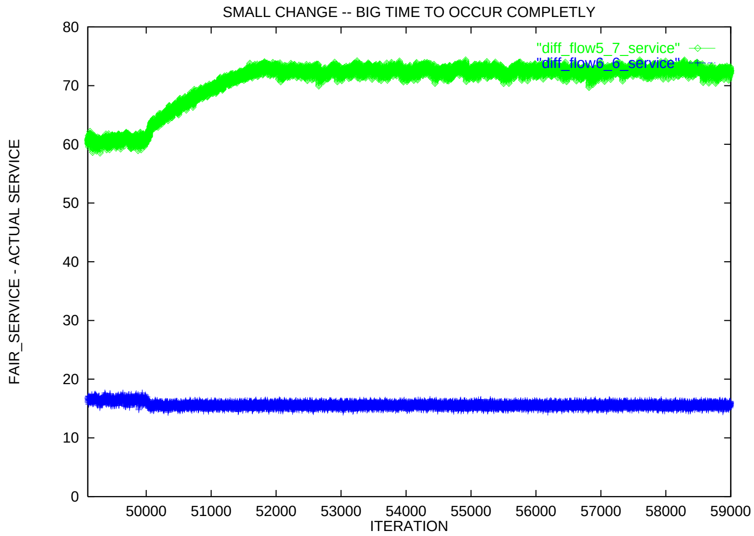


Figure 48: 8x8 switch, common case change

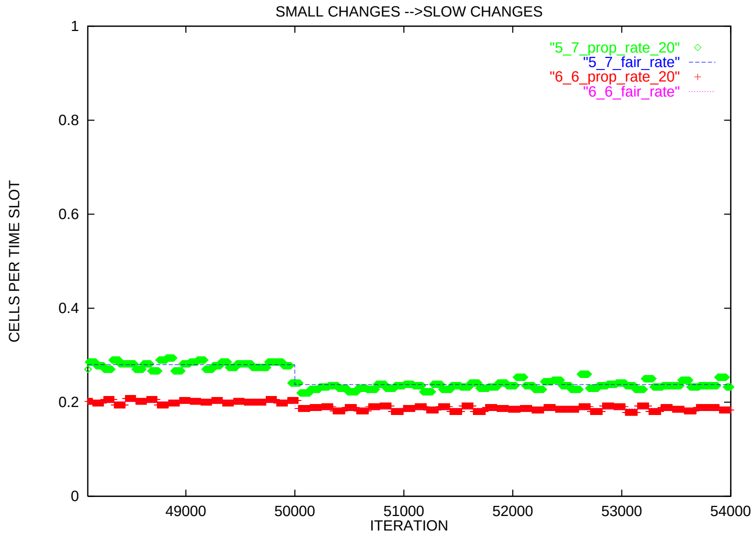


Figure 49: 8x8 switch, common case change

these are completely independent. The same holds if the graphs do overlap, but the trajectories of the two flows that caused the events do not. This can be the case if for example, in the chains that connects these two flows their is a flow j, that has strictly higher congestion index than f_1 and f_2 or their neighbors(this is necessary for the case that the events express change in the degree of urgency(weight)).

Now if the trajectories overlap, then we have the problem. We can here distinguish between three different scenarios. First the trajectory of flow f_1 may contain f_2. It is easy to prove in this case, that the trajectory of f_2 will be completely contained in the trajectory of f_1 The second scenario, is that f_1 resides in f_2's trajectory. The last case, is the one in which the trajectory of f_1 and f_2 overlap into a set of flows, that contains neither f_1 nor f_2.

Stop We will stop here the discussion and the analysis of the possible scenarios – due to time constrains. In what follows, we will simply show one result, where one flow receives two different and conflicting messages from two different interaction chains. In the configuration that we have in mind, the two event flows trajectories have in common only two flows(*Fig. 50* .

Figures *Fig. 51* and *Fig. 52* plot the phenomenon for the more interesting flows. What actually happens is that when flow 2_3 reduces its rate(happens first), flow 3_3 takes advantage and gets services in the output by draining its full buffer. At the same time, flow with id 3_4 increases rate, but this does not affects flow 3_3, which still had cells in the crosspoint buffers and could take advantage of the absence of 2_3. After a while(when the buffers drained) 3_3 “tries” to receive services from the input, but the negative affect of flow 3_4 constrains it. So this flow simply changed bottlenecks. The collision in strict terms did not occur.

Ending this section, we comment that it would be nice to find some configura-

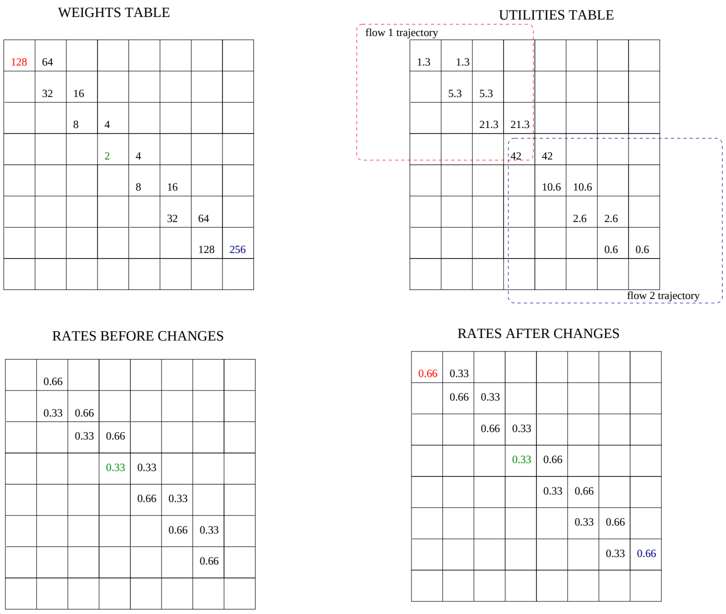


Figure 50: Weights tables for the configuration. The flows with the red and the blue colored weights are inactive and become active at during time slot 5000. The flow with green colored weight, is the flow that belongs to both trajectories. Note that although the flow with utility 42(right to the other 42), has bigger utility than the red flow, this does not belongs to this flow's trajectory, because if this flow was to change, this should rise its rate, but the bottleneck link is not in the interaction chain, that connects it with the red.

tion that would make the crossbar to oscillate, but we have not found any yet, which could be a hint that this cannot happen. It is also interesting to see, which event(s) will finally mainly affect the crossbar, under any of the three changes scenarios that we discussed in the previous paragraphs.

Conclusions:It is a complicate problem We agree, that the analysis presented in this section for events and the stabilization process in general, is quite complicated and probably is not described through the simplest means. But the problem its self is complicate and would require many hours of work, to find the simplest, rational description.

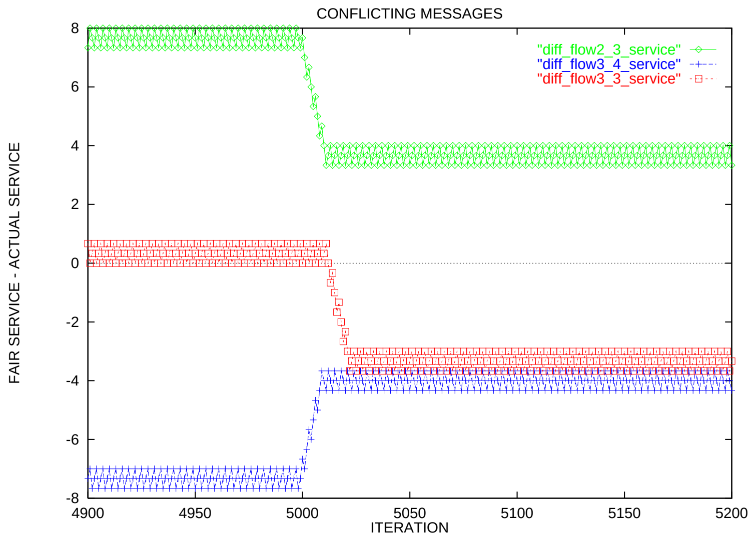


Figure 51: Difference in service for *Fig. 50* for the flows in the neighbor of the struggled flow

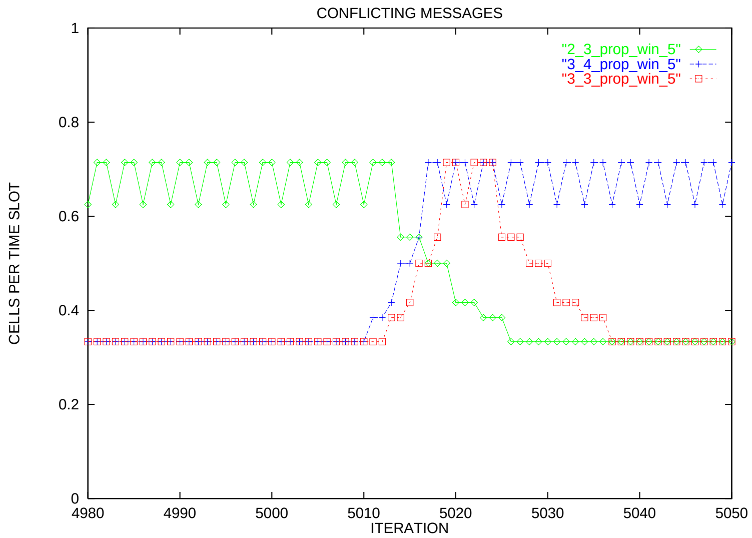


Figure 52: Rates plots for *Fig. 50*

5.4 Saturation Throughput and Output Queuing characteristics

.

5.4.1 Motivation and methods

The fact that the systems allocates weighted max-min fair shares in many cases(in all cases that we have examined), is certainly a very valuable asset of the scheduling discipline. It means that if there is never contention at the inputs, then the system will behave exactly like an output queuing system. But contention at the inputs will occur when the load is high, since these are the places where the main bulk of buffering occurs. In this section we evaluate the saturation throughput of the system and compare it to the ideal of pure output queuing. We also compare the delay allocation(or dealloacation to be semantically correct) on the flows by our system, with the – again – ideal output queuing system.

The arrival patterns that we will examine are:

- Bernoulli generated arrivals, with uniformly distributed destination
- Bursty traffic – traffic comes into bursts, with common destination.

We will also make an attempt to examine some arrival patterns, that relate with the weights of flows.

5.4.2 Bernoulli arrivals

Equal weights When all flows have equal weights, the average long term throughput of all flows is equal, for both weighted max-min fairness and output queuing with RR servers. This means that the delay distribution to flows is expected to be nearly the same under both scheduling disciplines. In *Fig. 53* and *Fig. 55* , we plot the average cells delay, on average and per-flow respectively, for a configuration where all flows have equal weight. Of-course our system belongs to the output queuing family and as we can see for crosspoint buffers equal to four cells, the average delay is almost adjoin, to that of pure output_queuing with infinite buffers at the outputs. But even with one cell buffer size(which translates to N buffers per-output), the performance is very close to ideal. We are tempted here again, to indicate the launch of the systems performance, that will be returned in the small cost of one single cell buffer at each crosspoint.

Any discrepancies compared to the ideal, can be attributed in the bad-synchronization of the input servers – all select the same output and the system temporarily detracts from being work-conserving – and to under-utilization, result-

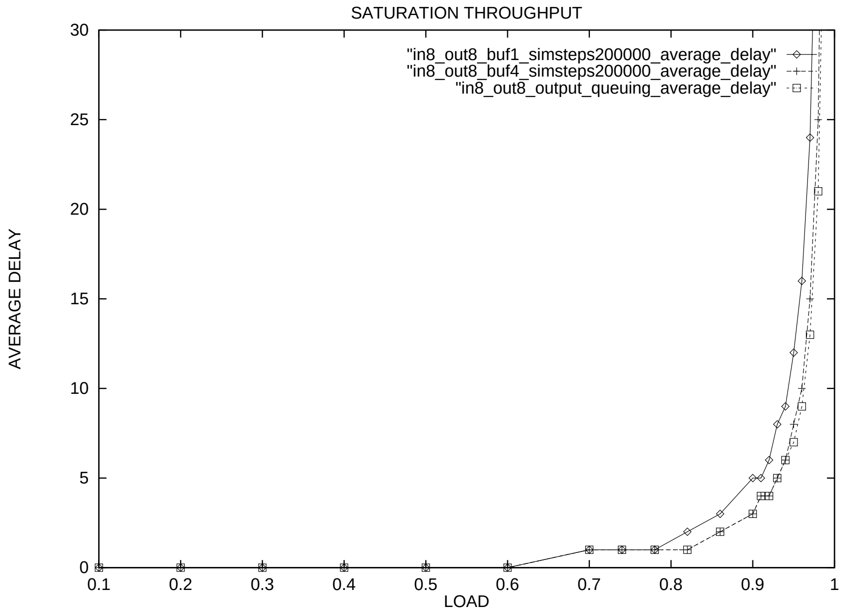


Figure 53: Average delay of all cells for an 8x8 switch with equal weights for all flows and with Bernoulli i.d.d arrivals

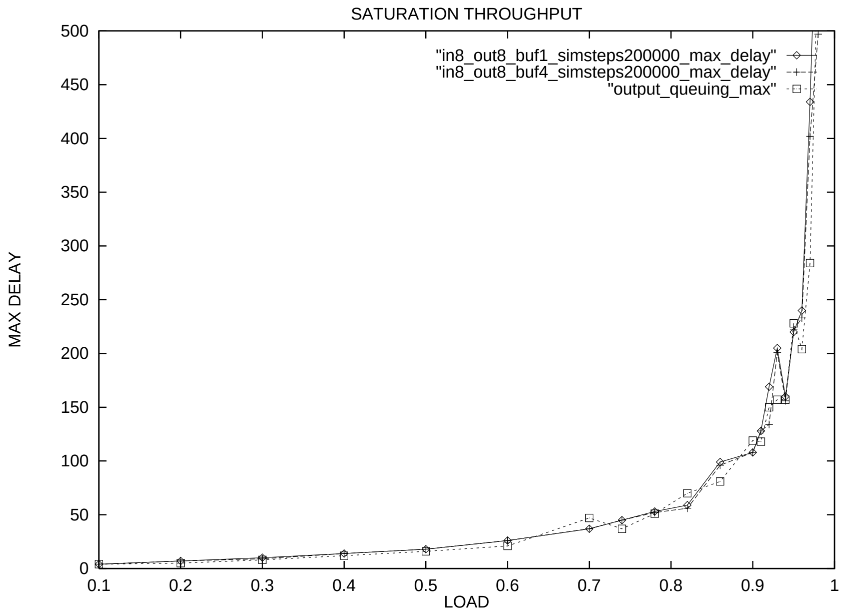


Figure 54: Max delay of all cells for an 8x8 switch with equal weights for all flows and with Bernoulli i.d.d arrivals

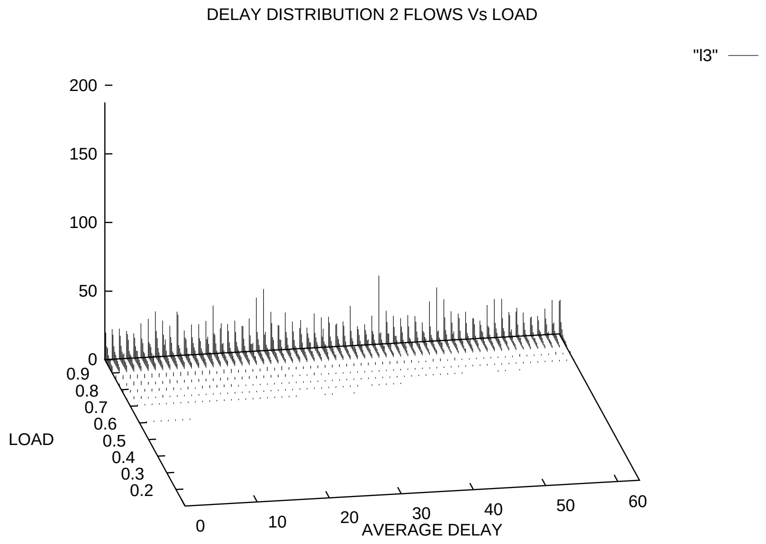


Figure 55: Delay distribution for a crosspoint buffered switch, with 4 cells buffers and Bernoulli i.d.d arrivals

ing from the lack of infinite buffer to push-data close to the outputs, when temporal bottlenecks moves from output to input.[69].

Weights introduction In this paragraph we study the saturation throughput, when weights have been assigned to flows. However we do not couple the weight assignment, with the traffic arrival process. We retain that input traffic is uniformly distributed to outputs. But this enviroment, is somehow "unfair" to the system. It is more natural to simply check, per-flow delays and not average metrics. Since the demand of all flows is $1/N$, the weighted max-min fair allocation is $1/N$ independent of the weights assigned to flows. What weights differentiate is the delay distribution to flows. The problem occurs when input servers are biased against some flows, due to their relative small weight, without actually knowing that all flows will place equal demand. But again, although this increases the delay of some flows, this is the role of weighted scheduling – we may want to reduce the delay of some flows, without caring about other (the last, should either increase their weights, or stop sending with this rate).

This first example, that corresponds to the hot-spots weights configuration, expresses even more this phenomenon. As we can see in figures *Fig. 56* , the average delay increases dramatically, especially for these flows that deserve very small input-server rate.

The fact that the average delay experienced by all flows, increases compared

[69]A freshly eligible flow will be delayed at most N cell times due to contention at the input buffers – adding N cell times incremental worst-case delay to pure output-queuing discipline. Of course if this phenomenon continuous, backlogs will appear and *packets* can experience high delays.

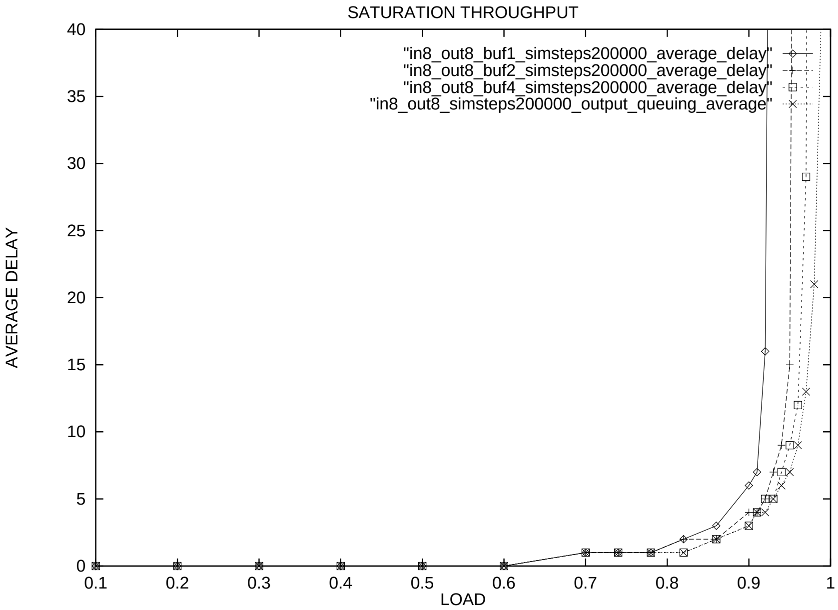


Figure 56: Average delay of cells for an 8x8 switch with the hot-spot weights assignment, Bernoulli arrivals and uniform destination selection.Configuration 6

to output queuing, reveals once again, that the system is non-work conserving. When congestion occurs, contention at the input servers will force propagation of cells to follow input-servers fairness as well and the delay distribution will be in general different compared to output queuing. This is the case when a low priority flow, experiences long delay at the input server, even in the case that the respective output is congestion-free – due to the relative small weights of the flows using the same output, or due to the temporal lack of traffic for this output. So flows are affected by the existance of hots spots, if we adopt statically-weighted scheduling at the inputs, like we do in our system.

Figures *Fig. 57* and show the dramatic impact of this configuration to the system, since for the output queuing system, the average delay ranges in the range [0...250], where for our system, with four cells size for the crosspoint buffers is much greater. We should note here, that the flows that experience these high delays, are simply flows with relativelly small weight at the inputs(in the hot-spot configuration there are plently candidates for this behavior).

In the second example, we have created a uniform topological distribution for the weights of flows. In this case we expect our system to closely emulate the output queuing system. Since the sum of weights at inputs, outputs, is expected to be equal on average, flows will not be mis-treated in inputs when these are treated "equally"(better) at outputs – the cause that produced the discrepancy relative to output queuing, at the hot spot case.

Plots in *Fig. 58* and *Fig. 59* , manifest that in all but the most degenerate cases, discriminative scheduling at the inputs, does not necessary imply decreased

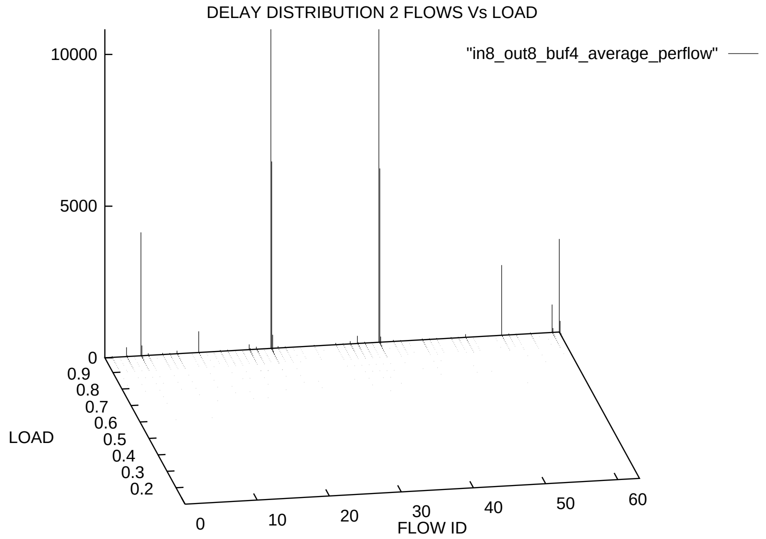


Figure 57: Per flow delay distribution for a crosspoint buffered switch, with 4 cells buffers and Bernoulli i.d.d arrivals. Configuration 6. Hot-spots configuration of weights.

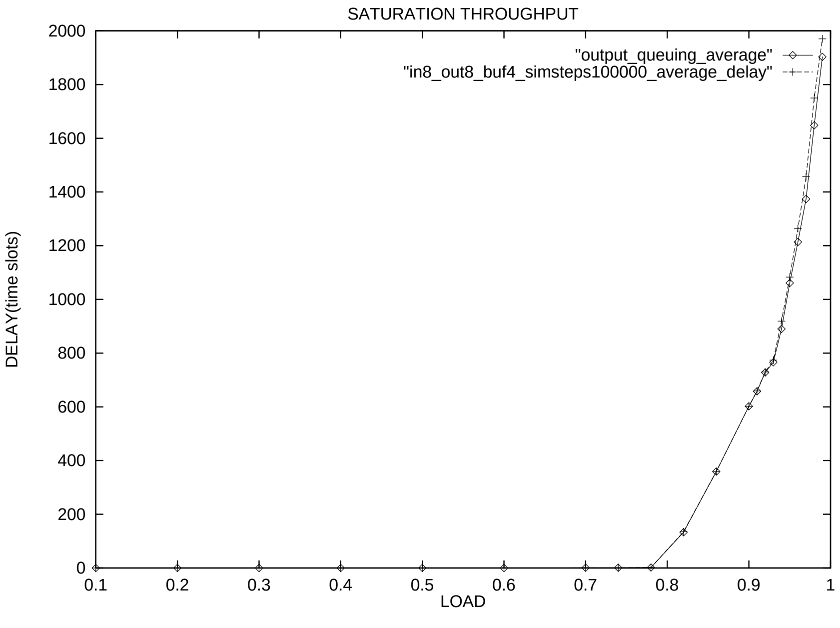


Figure 58: Average delay, for a 8x8 switch configuration. Bernoulli i.d.d arrivals. Configuration 7. Uniform distribution configuration of weights

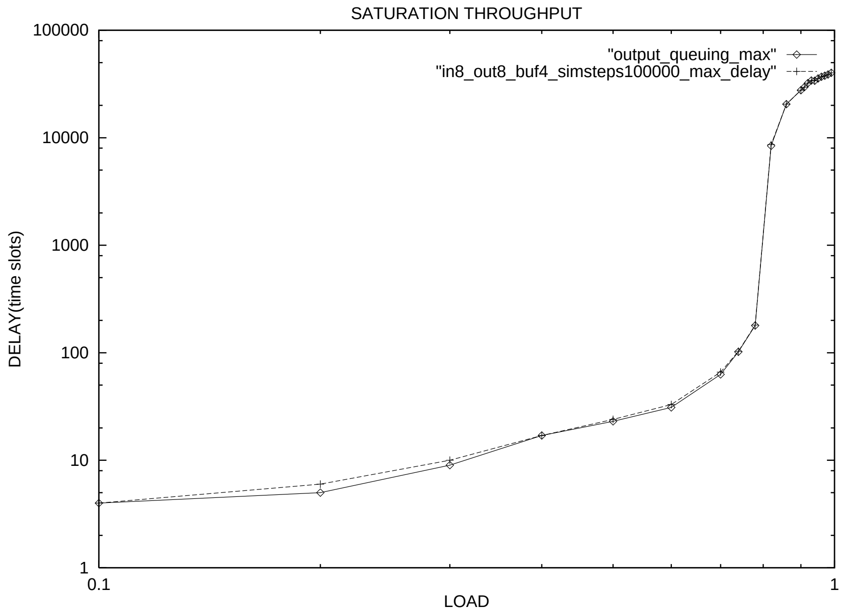


Figure 59: Max delay, for a 8x8 switch configuration. Bernoulli i.d.d arrivals. Configuration 7. Log-scale. Uniform distribution configuration of weights

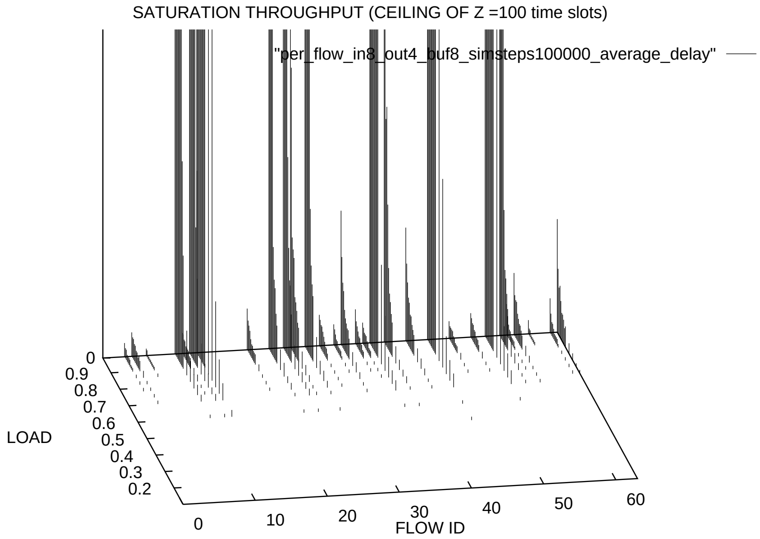


Figure 60: Per flow average delay, for a 8x8 switch. Bernoulli arrivals i.d.d arrivals. Configuration 7. Uniform distribution of weights. The plotted Z axis(delay), ranges from 0 to 100. Actually, the delayed flows, experience much higher average delay(~ 150000).

throughput. Both the output queuing system and $WRFQ$ behave poorly, simply because some flows have increased delay due to their small weight, dragging average delay to high levels.

Per micro-flow fairness In the paragraph for the alternative/possible schemes on weights assignments(section 2.3), we made clear that our analysis will not stand upon any hypothesis on the "nature" of weights. In the previous example, we regarded the weights of flows and their arrival rate, as two independent processes, which resulted in observing very long delays. Here we will pick an assumption on the weights formulation, that will enable us to couple the weight of a flow, with its arrival rate.

We have presented an application of weights for sustaining per micro-flow fair service in systems, that aggregate micro-flows into greater entities. Then the weight of each aggregate flow, should be the sum of the ingredient, active micro-flows. We can fairly then decide, that each aggregate flow has an arrival rate proportional to the number of micro-flows that this carries – that is proportional to this flows weight. Here we take this stand to analyze saturation throughput. A critical point here is to make sure that we do not over-occupy any output link. That is for the sake of the experiment, we have to ascertain that the load presented upon any output link will be affordable. Since the per-micro flow fairness view of weights couples the relative weights of flows in the inputs, with the arrival probability of the flows and we demand that arrival rates to be feasible with regard to outputs capacity, it follows that output fairness should be equal to input fairness. Configuration 6 presents this case, where the sum of weights is equal at all input and output links.

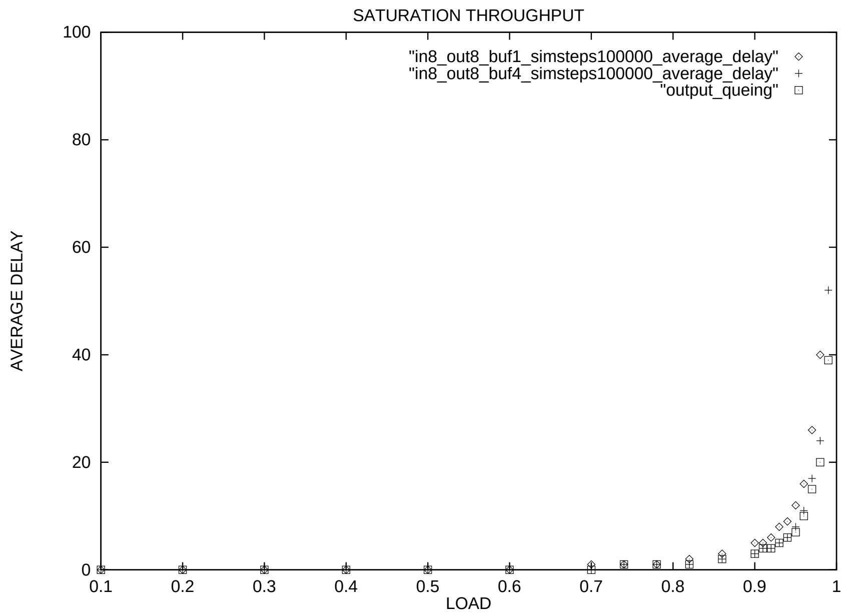


Figure 61: Saturation throughput for Bernoulli arrivals and weights dependent destination selection

We can see that in this case(*Fig. 61*), the buffered crossbar behaves very close to output-queuing, much like the equal weights case. This is not surprising; since both input and output servers allocate the same rate to flows, the phenomena of flows experiencing input congestion, without actually being congested at the output, will not occur.

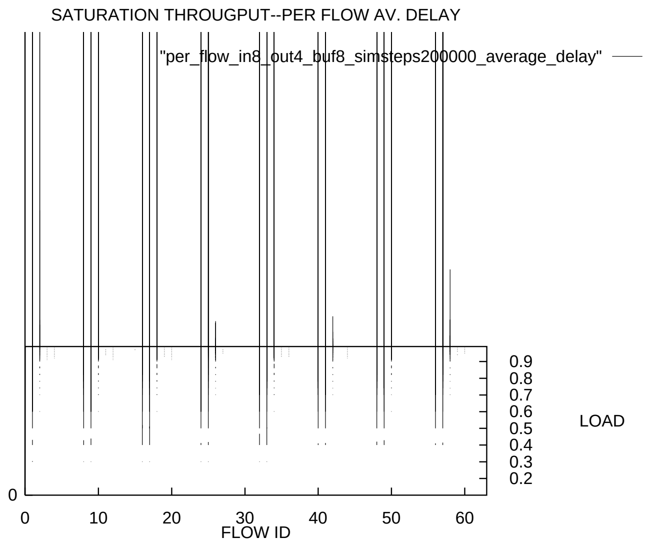


Figure 62: Per aggregation-flow delay, under a hot spot assignement for the weights. Smaller outputs are oversubscribed, since the arrival rate depends on the flows weights and big weights are topologically placed, on outputs with small index(0,1..).. Z axis has been chopped to 1000.

But what happens in the case, that some outputs are over-subscribed? Will other flows be affected? Under the weights scheme that we consider in this paragraph, flows with big weight have greater arrival rate. If we use the hot spot distribution of weights, then, using the semantics of the per-micro flow fairness scheme, there are simply more micro-flows destinated to some outputs(the hotspots). The scheduler, will pass greater priority to the hot-spot aggregations, but since the outputs will be oversubscribed, we expect the congested micro-flows, to experience great delays. Other micro-flows, destinated to less congested outputs, should no be affected. As we can see in *Fig. 62* , flows destinated to small outputs experience delays, while well behaving flows(or aggregations destinated to non-congested outputs), realize minimal delays – in *Fig. 62* , these are the flows inbetween the highly delayed flows[70].

[70]The output of a flow, can be computed with the arithmetic modulo operation: output=FLOW_ID mod 8

5.4.3 Bursty traffic arrivals

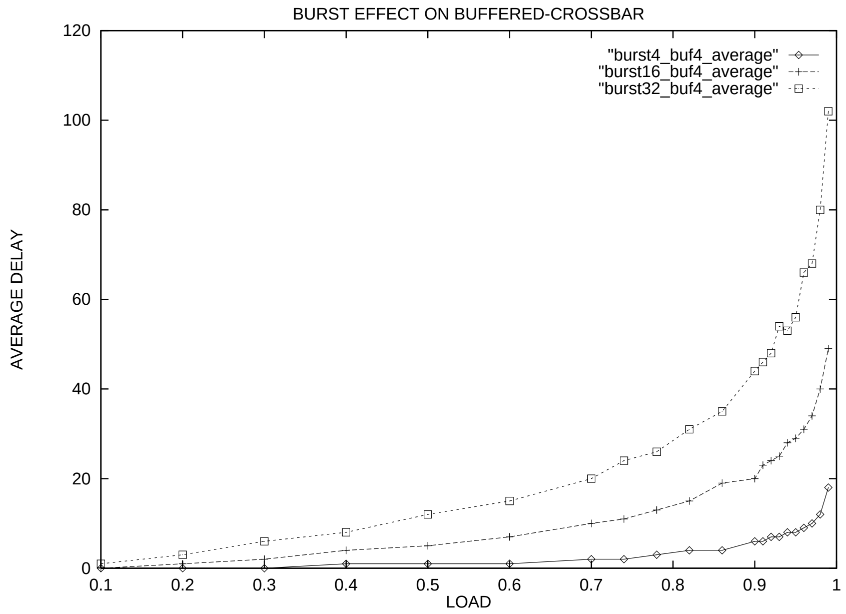


Figure 63: Bursty traffic effect, on average delay for crossbar with buffer size 4. Burst size 32,16,4 with deviation 8, 4 and 1 respectively. Equal weights

Traffic in the Internet is not always uniformly distributed on time, like the Bernoulli arrival model suggests. There are plenty of reasons advocating for a bursty arrival model. First sources – that is applications – can place bursty demand for communication. The *feedwise* communication model is a good representative term of this behavior. Additionally congestion fluctuation, naive – non smooth – service discipline at routers and hot-spots, all advocate for a bursty arrival pattern.

Here in this section, we analyse the saturation throughput of the buffered crossbar, in the case of bursty traffic. We assume that flows packets, appear into bursts of an average length with a small uniform deviation. The average length and the deviation are common for all flows. In order to sustain an average incoming rate a, while still cells arrive into bursts, of average length b, the decision to "transmit"(in the switch inputs) a new burst – this decision is taken only when no burst is "active" –, is taken randomly with probability equal to $x = 1 - (1-a)^{\frac{1}{b}}$[71].

In figure *Fig. 63* and *Fig. 64* we can see the effect of traffic burstiness on the saturation throughput for the $WRFQ$ and for a pure output queuing system respectively. In general bursty traffic will introduce delays, since the demand can be short-term highly conflicting. We can see in the *Fig. 64* , that the average delay almost doubles when the average burst does. This result is very close to what we

[71]If there exists such a probability x, that governs the arrival process, then the probability of *not* transmitting a cell at a given time slot, will be equal to the probability of having decided not to transmit a cell during any of the b previous slots, so $1 - a = (1 - x)^b$. We should note here, that this method actually resulted in slighly smaller arrivals rate than anticipated(in the order of minus 5%).

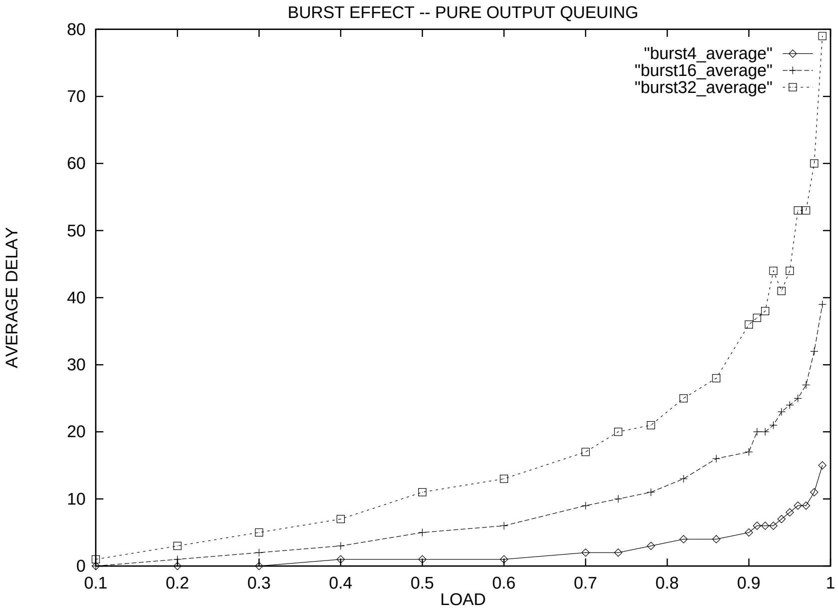


Figure 64: Bursty traffic effect on average delay for output queuing. Burst size 32, 16, 4 with deviation 8, 4 and 1 respectively. Equal weights

intuitively presented in the section for transient traffic(section 4.4.6). There we argued that the time to transmit a feasible traffic stream of time-length T, can be as large as $2 \cdot T - \frac{1}{N}$, where N is the number of inputs. In the pattern arrival presented there, a packet may experience delay equal to $d_{max} = T - \frac{1}{N}$, when the burst size is $b = \frac{T}{N}$. So the maximum delay is $d_{max} = \frac{b-1}{N}$, which is proportional to b.

The plots that compare our system with output queueing, are presented in figures *Fig. 65* and *Fig. 66* . We can see that for buffer size close to 4, the system actually behaves very much close to pure output queuing, but realizes discrepancies that increase with burst size.

A final (check)point We have simulated, the worst-case feasible arrival pattern for the pure output queuing discipline, mentioned in the section for probabilistic sources(section 4.4.6). As we intuitivelly proved there, the delay of our system in transmitting the traffic, under any size for the crosspoints, is equal to that of pure output queuing, equal to $2 \cdot T - \frac{1}{N}$. More specifically, our simulation was for an $4x4$ switch and 1000 time slots. During the first 250 steps, all sources sent to output 0, afterwards to output 1 etc. Both the output queuing system and $WRFQ$ under any size for the crosspoints, cleared the backlog out, at time slot 1750.

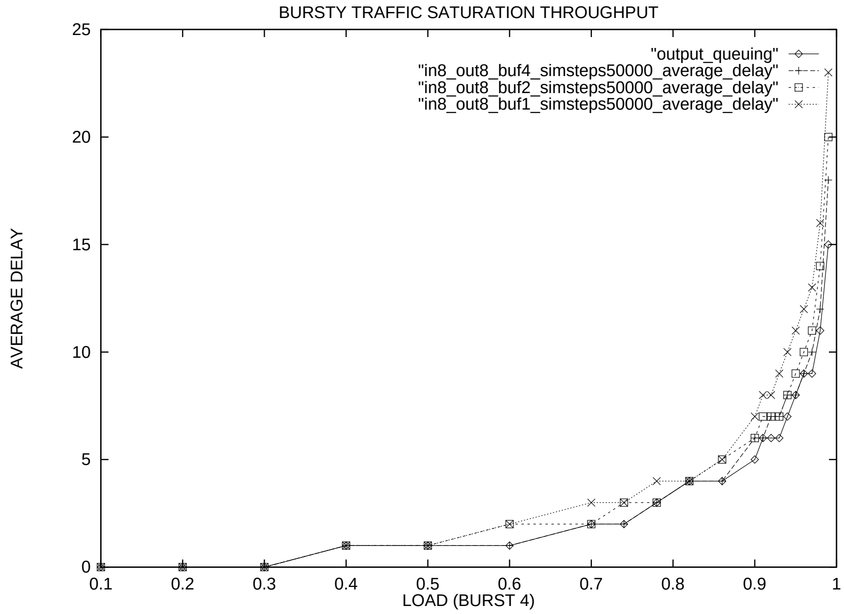


Figure 65: Saturation throughput for bursty traffic arrivals and uniform destinations selection. Burst size 4, deviation 1. Equal weights

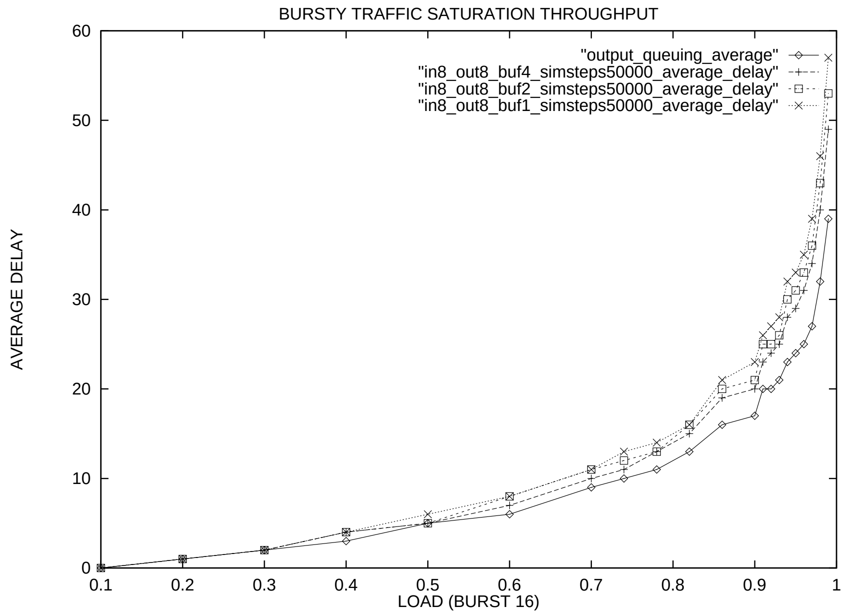


Figure 66: Saturation throughput for bursty traffic arrivals and uniform destinations selection. Burst size 16, deviation 4. Equal weights

5.5 Different choices for the input and/or output servers

In the section about the WFQ-related servers, we show that there is a considerable number of alternatives solutions, in the design spectrum. Some (WF^2Q,WF^2Q+,WFQ) are proved to be better in terms of scheduling properties, while other are more cost-effective, by roughly keeping the desired properties, while reducing the hardware-complexity costs(WRR/WFQ). Additionally some service disciplines, have strictly different properties (VC, $FCFS$, LQF, LRU). In this section we experimentally evaluate/compare some of the different variants weighted-based discriminative-methods, in the decision of what servers to use for the buffered crossbar.

The methods/technics that we will examine, can be split into two categories, based on the objective that we use to compare them. For one, we will try to evaluate if there is anything to gain by using more elegant approximation of the GPS service – other than the WRR/WFQ, that we used so far. In this task, our objective remains the weighted max-min fairness, so we will compare the corresponding $PAWD$ efficiency metrics. We will evaluate the WRR/WFQ, the WF^2Q+ and the $VClock$ techniques. Additionally we have simulated a system, that uses the virtual clock method in the input servers and WRR/WFQ at the outputs, which we name in the plots as input_virtual_clock[72].

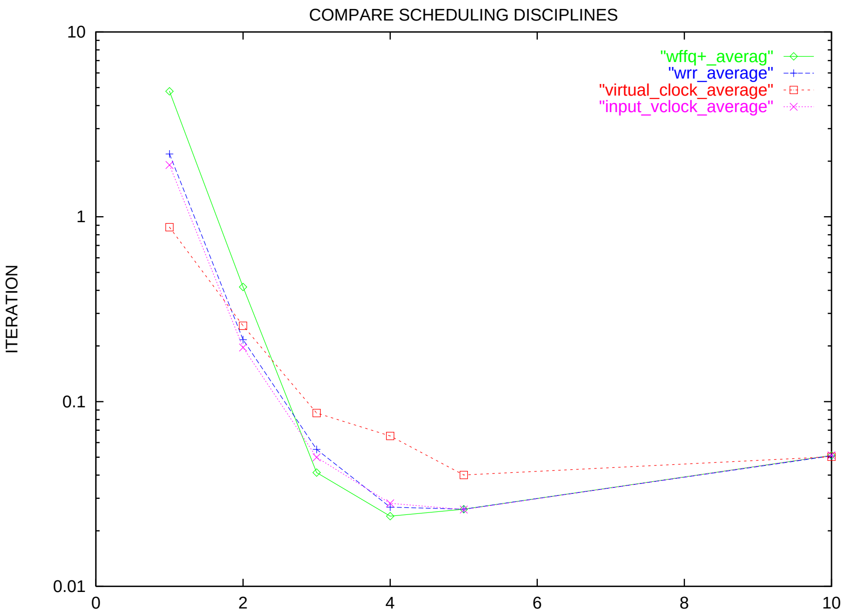


Figure 67: A hot-spot configuration

[72]Remember that we argued that virtual clock, may be appropriate for the fairness objective(in the constant-state sources case), since this gives high priority to a new eligible flow, so flows that were previously backlogged will be served fast. Of course, in the case of transient source, we expect much worse behavior under VC, primarily due to punishing flows for receiving excess bandwidth.

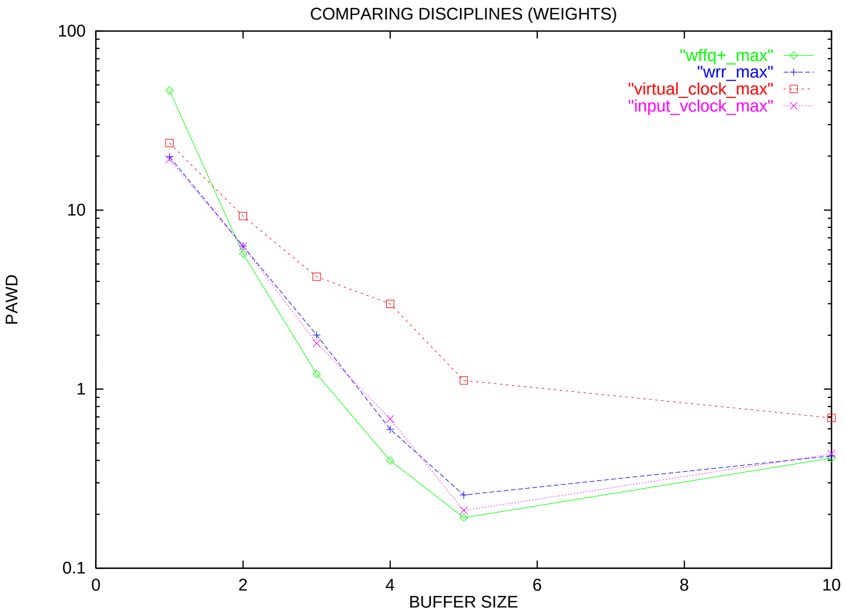


Figure 68: A hot-spot configuration

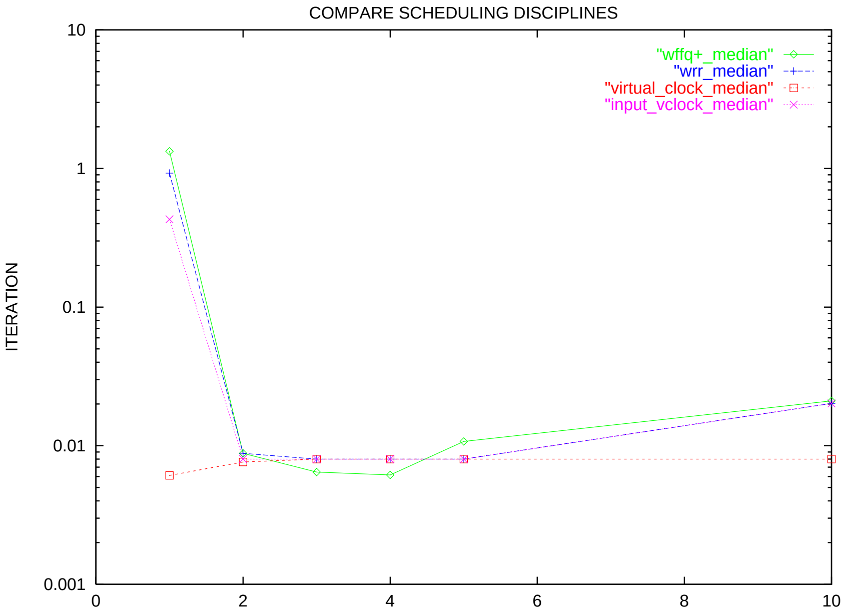


Figure 69: A hot-spot configuration

In figures *Fig. 69* , *Fig. 68* and *Fig. 67* , we can see that virtual clock behaves well, even for the one buffer case, yielding almost the best performance. This is somehow consistent with that virtual clock philosophy, which gives priority to previously inactive flows – with one cell buffer at the crosspoints, flows oscillate between active and inactive states, all the time. WF^2Q+ yields better performance than WRR/WFQ – although in the one buffer space, it did not do that well. It is a noticeable result, that with one buffer space, the three GPS-approximation, performed inversely to their complexity-precision. We have actually explained the reasons of this behaviour. WRR/WFQ by not emulating the GPS model well enough, favors previously inactive flows, which is somehow appropriate when the crosspoint buffer size is small. Of course WRR/WFQ will not punish flows for receiving excess bandwidth, in the same extent as VC, but simply gives a small temporal priority to newly eligible flows, by not evolving the virtual time function, at times that according to GPS, it should.

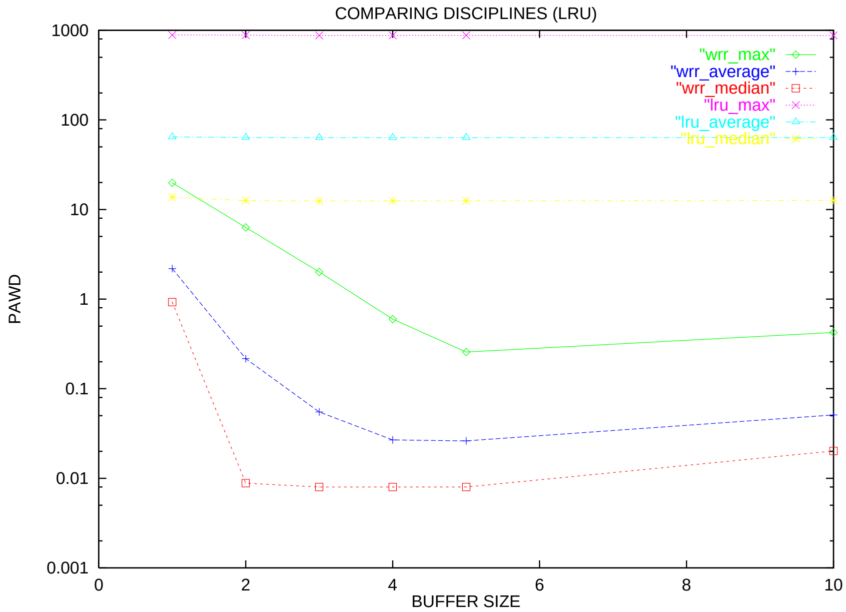


Figure 70: A hot-spot configuration

In figure *Fig. 70* we compare a system that uses an adaptive scheduling method in the input servers (LRU) with one that uses WRR/WFQ, under the weighted max-min fairness objective. This is not very fair of-course, since we do not expect the LRU system to even close the objective. For completeness though, it worths plotting it, since as we will see later, the choice between *adaptive* or GPS-*like*, turns to be a good trade-off for the input servers.

Finally in *Fig. 71* we present one more result, comparing the main $GPS-like$ alternatives under a uniform weights distribution, in the view of weighted max-min fairness. Here the results are more close together, compared to the hot spot case presented the previous figures. Again virtual clock, has the lowest median

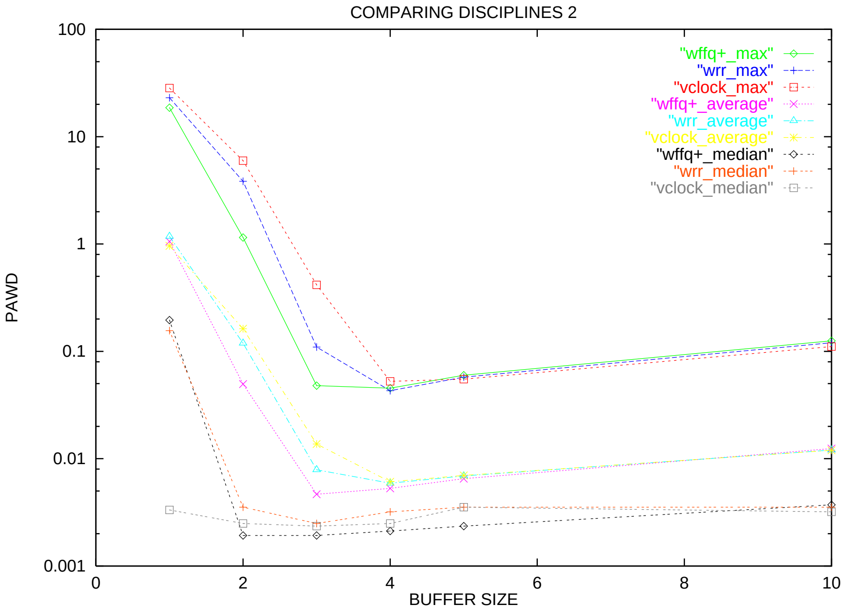


Figure 71: A uniform case compare

discrepancy than the other methods.

The other alternative/possible scenario that we will examine, is to use other types of service disciplines in the input servers, like first come first served($FCFS$), longest queue first(LQF) and longest waiting time(LRU), but keep the GPS-related servers at the outputs. Many designers ([MagBufFullCap][PackSwitDes]), have found that these techniques are more applicable for the input servers, since the contention in the inputs is not expected to be so severe that will detract the disciplines ability, in applying QoS. So these, either do not care for discrimination, or they care to sustain high throughput, which is not always possible as we have seen, when using GPS-like servers at the inputs. For example, since the up-stream node is responsible for the arriving order to the switch, it seems legitimate to use the $FCFS$ discipline for the input servers. Or in order to prevent starvation, the longest queue first(LQF), or the least recently used(LRU)[73]could be considered. LRU differs with round robin, in that it always selects the *least recently served*, whereas in round robin the cycle order of selections is predefined – the one reduces to the other when all flows are continuously active. These methods are sometimes call adaptive, since they formulate ("adapt") the priority of a flow, based on the "running environment", usually under an objective hypothesis-purpose.

All these adaptive servers, need a sorting engine or a heap, to find the minimum previous serving time(LRU), or the minimum arrival time ($FCFS$) or the maximum queue length(LQF) and have roughly similar implementation cost with a simplified WFQ engine.

[73]In [PackSwitDes] they consider the OldestQueueFirst (OQF), which differs with the LRU in that a queue is aging only when this is active.

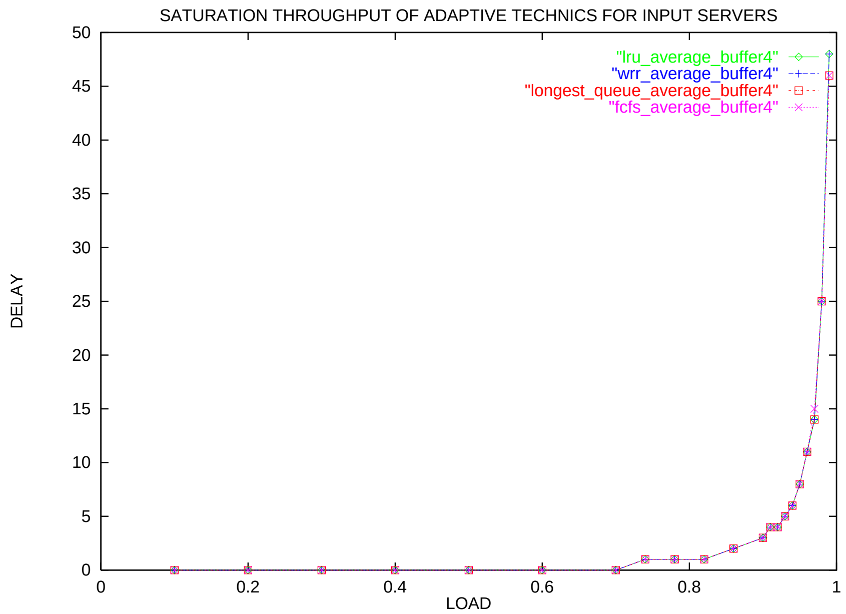


Figure 72: A uniform weights case with some flows inactive. Bernoulli generated traffic arrivals with uniform destination(but affordable load). Plot of average delay

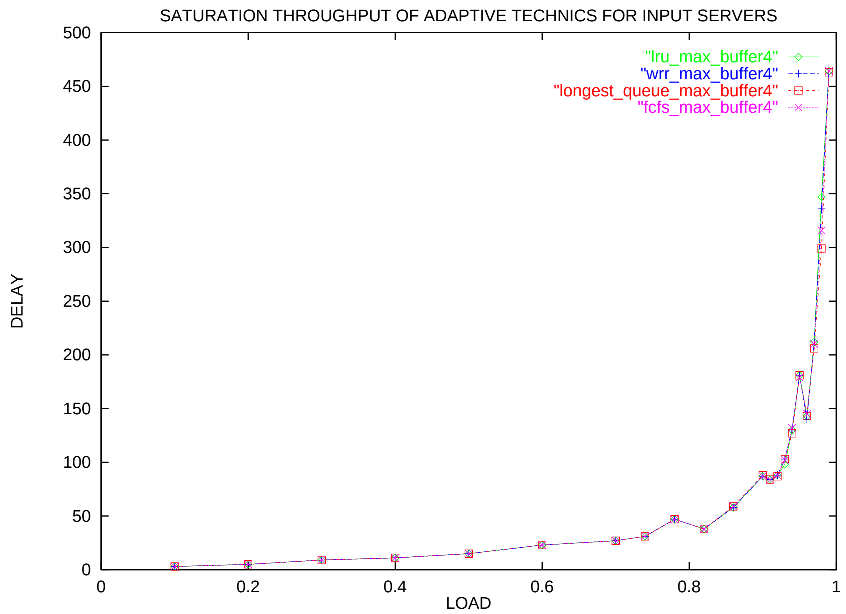


Figure 73: A uniform weights case with some flows inactive. Bernoulli generated traffic arrivals with uniform destination(but affordable load). Plot of max delay.

In figures *Fig. 72* and *Fig. 73* , we compare the average and max delay, of the adaptive methods described above, with WRR/WFQ. These variants are used in the input contention points and we keep the WRR/WFQ discipline at the outputs. All methods are configured with the same size of crosspoint buffers(4). We can see in these plots, that under the assumptions of Bernoulli generated arrivals, uniform destination distribution and if no weights are assigned(all active flows equal), all methods perform equally well. Next we relax the Bernoulli arrivals assumption.

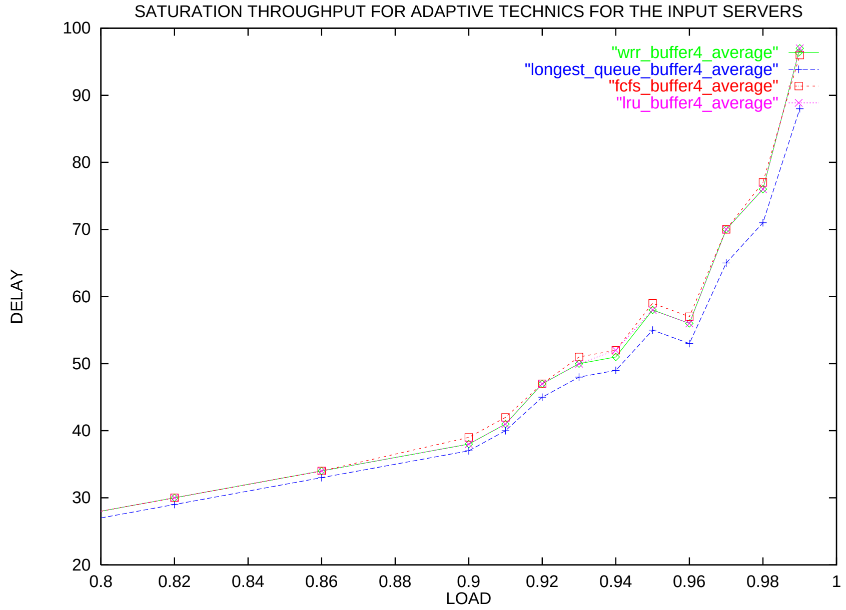


Figure 74: A uniform weights case with some flows inactive. Traffic comes into bursts of 32 cells. Plot of average delay

Adaptive methods are expected to perform better when the environment is not "regular", since this is why these were devised in the first place. *Fig. 74* and *Fig. 75* , demonstrate the advantage of LQF in minimizing average delay, when traffic comes into bursts. The configuration is exactly the same with *Fig. 72* , but the arrivals of cells is now in bursts of cells with identical destination.

You should not think that we have forgotten to compare the $GPS-like$ disciplines, in the delay objective. WF^2Q+ is supposed to provide the smaller WFI index[74] and consequently the lowest delay bounds. It may worth using this discipline, only because of this characteristic. Plots *Fig. 76* and *Fig. 77* compare these different alternatives/variant-methods, with different crosspoint buffer sizes(1 and 4), but under uniform weights(the same with the previous plots) and non-bursty traffic assumptions. Virtual clock seems to be the worst choice($4b-VC$ is worse, even than $1b-WRR/WFQ$), whereas WF^2Q+ is as expected the best, but not considerably better than WRR/WFQ.

[74]Delay in serving a newly eligible flow

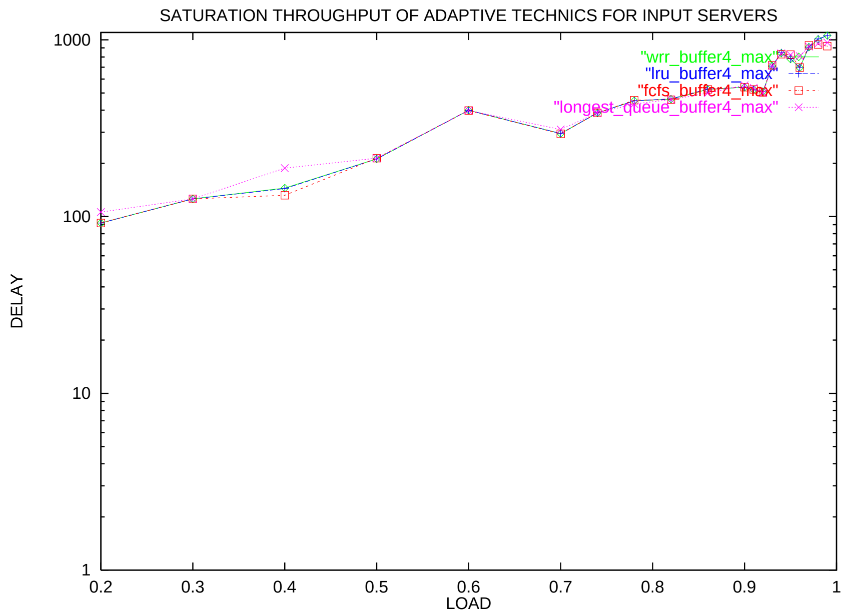


Figure 75: A uniform weights case with some flows inactive. Traffic comes into bursts of 32 cells. Logscale of max delay

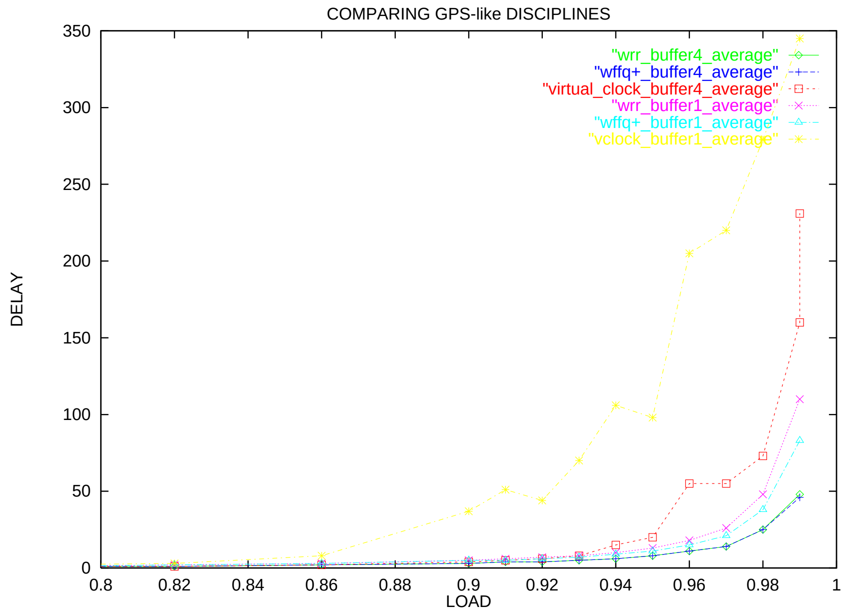


Figure 76: A uniform weights case with some flows inactive. Bernoulli generated traffic arrivals with uniform destination(but affordable load). Plot of average delay

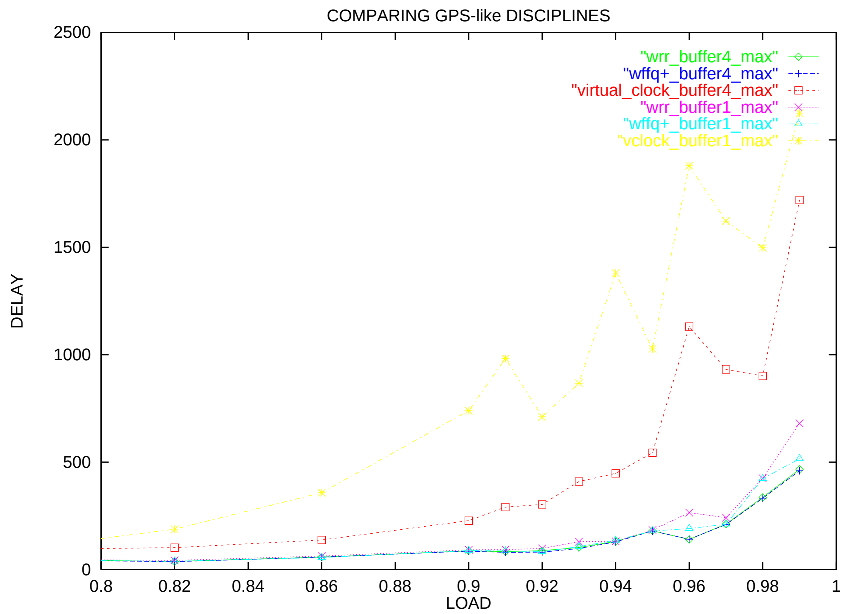


Figure 77: A uniform weights case with some flows inactive. Bernoulli generated traffic arrivals with uniform destination(but affordable load). Plot of max delay

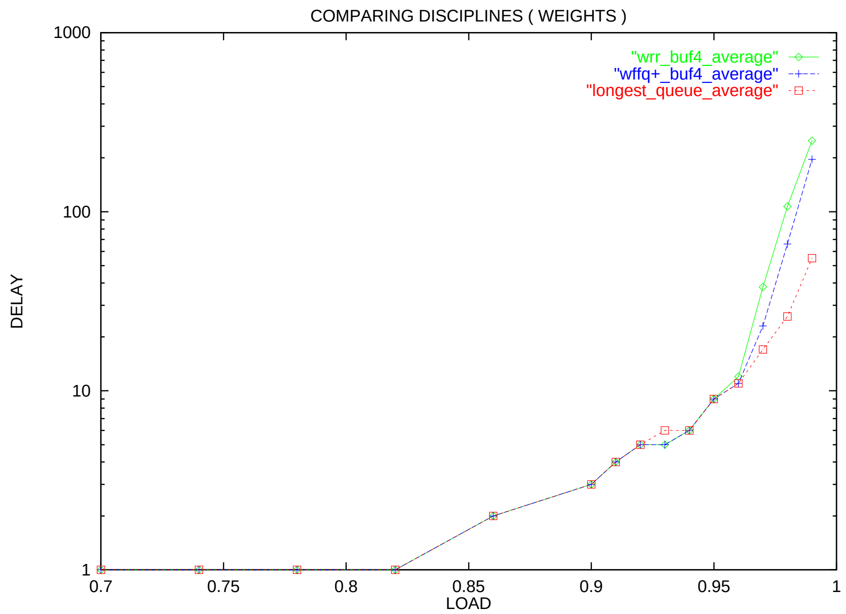


Figure 78: A hot-spot, topological distribution of weights case, with some flows inactive. Bernoulli generated traffic arrivals with uniform destination(but affordable load). Plot of average delay

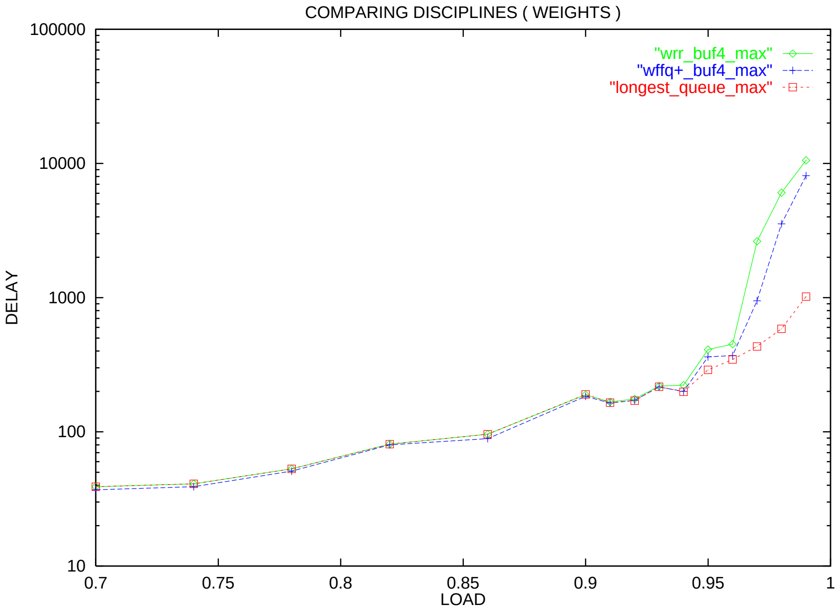


Figure 79: Same as *Fig. 78* . Max delay plot.

Finally plots *Fig. 78* and *Fig. 79* , exhibit the delay behavior when weights are considered. We consider the WF^Q+,WRR/WFQ and the LQF disciplines. The arrival pattern is drawn independently of the weights and this makes the LQF to be discernably better. The input servers, do not know that all flows will have equal arrivals and by serving them in proportion to their weight, will decrease the backlog periods of the outputs $\rightarrow$ increase the average delay. This is so evident here, for the hot spot configuration, where additionally all input servers synchronize in sending to the spotted outputs, creating fluctuation in the service and low utilization. The placement of demand to the output servers by the inputs is irregularly distributed, whereas arrivals are actually uniform. Big crosspoint buffers(4 cells in this configuration) "legalize" the input servers to do so, even more.

We end here this section rescheduling more deep discussions of the trade-offs that have been disclosed, for the discussions section. In section 10 you can find some additional plots on saturation throughput, comparing the LQF with WRR/WFQ, under the bounding delay – for some flow – objective. Before we move to discussions, we present some final experimental results, evaluating two novel methods for assigning weighted max-min fair shares to flows, in a buffer-less crossbar.

5.6 Scheduling algorithms for a buffer-less crossbar – the difficult problem

Here, in this section, we discuss some methods which aim to solve the bipartite graph matching and the demand for fair scheduling together – that is they try to solve "scheduling crossbar problem". We have already seen such attempts (PIM,$SLIP$,$FIRM$, $StableMarriage$). Here we discuss if any of these methods or something close to these, can provided weighted max min fairness.

5.6.1 Theoretical discussions on proposed iterative schemes – in the view of weighted max-min fairness

Although $i - SLIP$ seems to be the most cost-effective solution, here we discuss these methods primarily with theoretical interest, out of technology burdens. In this view $i - SLIP$ does not provide analogical scheduling, since it does not discriminate traffic streams. The most obvious way to include more sophisticated scheduling in the $i - SLIP$ context, is by upgrading the input and output servers from Round Robin, to something like WRR, much like in FIM – but FIM uses an adaptive method in the inputs, to accept the grants.

During each scheduling period, the output servers would select which flow to grant – considering as active only these inputs that have sent a request –, by comparing their next service times and selecting the one with the minimum. But there are some problems. For one, if we try to sustain the desynchronizing characteristic of $-SLIP$, simply by incrementing the next service time for selected flow only if the grant was accepted, much like $i - SLIP$ does, we would simply fail. So we need more than one iterations to sustain a respectable throughput – that is necessary, in the weighted max-min fairness view.

The problem of creating an iterative algorithm that will close to the ideal service, is that the schedule must be computed serially – one iteration will simply not do. Even $i-SLIP$'s trick to accomplish good utilization with one iteration, is that after a complete schedule the next one will be ready for the next iteration – since all output servers will be ready to grant a different flow. This is a consequence of the round robin discipline and cannot be sustained if we generalize to weighted round robin. This means that schedulers decision will be "directed" and dictated, either the grants(when talking for the input servers), or by decisions taken in previous iterations(when talking for the output servers).

Consider a flow f with a relative high priority index in the input server compared to its competitors in this server. This flow has a bottleneck in the output though, since all flows in the respective output have high priority index. This scenario is shown in *Fig. 80* . The output server, will grant equally all flows. We would like when the output grants this input f, the input to accept the outputs

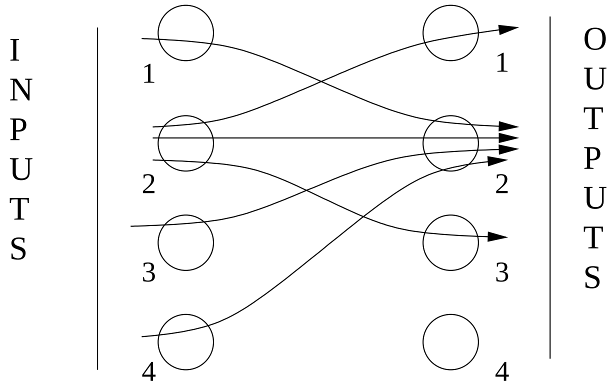


Figure 80: Flow from input 2 to output 2, faces bottleneck at the output(where this should receive 1/4 of the link). But also faces contention in the input. Then an iterative discipline can mistreat this flow. This holds for PIM ([SLIPMcKeownAndersComp]) and we argue that this holds for every weighted iterative discipline).

grant immediately. But what might happen, is that when the output grants the congested input(f), the input accepts some other grant. The output will then grant an other input(the opportunistic services will be distributed to others in time), without "punishing" these others later(keeping up with the GPS-fluid philosophy). So these others will get more service more than this input contented (but not congested) flow.

In the buffered crossbar this is not the case, since we do not oblige the input to synchronize with the outputs. When the output "thinks" it is time to service f, it will have available a history f's input services, waiting in the buffer. Since we assume that f faces bottleneck at the output, then we expect that in-between the times that the output "thinks" its time to serve f, the input will have foresee, that f has a cell in the crosspoint. Anyway with these arguments we want to excuse our selves for not simulating this system. But in any case it seems rational. Reaching down to the bottom, the argument is that iterations cannot exactly substitute the buffers mechanism —- although both methods deal with the common underlying – synchronization-problem. In a similar fashion, buffers cannot substitute iterative methods in sustaining instantly high-throughput or work-conserving scheduling [75].

5.6.2 Iterative WRR – an expensive discipline

Next we evaluate a very expensive iterative technique for finding a complete crossbar match, with discrimination among traffic. This is based on a central scheduler and utilizes similar ideas, with the GPS-scheduling in the case of single resource. We consider an iterative, two dimension WRR, engine that implements a WFQ variant, and which keeps the *next_service_times* for each flow(route), much like in

[75] Buffers and independent scheduling cannot reassure aggregate work-conserving discipline, while expensive iterative methods can.

the one dimension case (section 3.1). The pseudo code for $IterWRR$, is shown bellow:

Iterate until no more flows active or scheduled

- *find min next_service_time out of all active flows*
- *consider as inactive, all flows using the same input or output with min*
- *update next service time of the scheduled flows by adding their respective service_intervals and update virtual time to the min next service time*

This algorithm will terminate in N(number of outputs) steps, since it finds one edge of the match in each iteration – the cost paid for being work conserving. It can also guarantee a minimum rate to each flow, equal to $r_f^{min} = \frac{w_f}{\sum_{g \in F} w_g}$ and probably a delay bound, depending to the fair engine used[76]. The sum of minimum guaranteed rates over all (N^2) flows $f \in F$ is: $\sum_{f \in F} \frac{w_f}{\sum_{g \in F} w_g} = 1$. So if we use the above formula to assign rates and delays on flows, we will simply under-estimate the switching capacity of our system and pretend that it has only one output.

The real rate that this algorithm will apply to backlogged flows, can be computed as follows. Certainly each flow f will receive rate $\frac{w_f}{\sum_{g \in F} w_g}$. This is the rate assigned, if only one iteration was used. If we want to include the second iteration in our computation, then we have to deal with all possible decisions that the algorithm may have taken during the first. For each possible decision of the algorithm in the previous iteration, a new candidate set is formed. So $r_f^2 = r_f^1 + \sum_{g \in F} r_g^1 \cdot \frac{w_f}{\sum_{j \in F, j \wedge g = 0} w_j}$, which by substituting the first iteration terms expands into:

$$r_f^2 = \frac{w_f}{\sum_{g \in F : g \wedge f = 0} w_g} + \sum_{g \in F} [\frac{w_g}{\sum_{j \in F} w_j} \cdot \frac{w_f}{\sum_{j \in F, j \wedge g = 0} w_j}]$$

You can easily verify that $\sum_{f \in F} r_f^2 = 2$, as expected. Continuing in the same fashion, the rate of f after N iterations of the algorithm will be:

$$r_f^N = \sum_{i=1}^{N} \left(\sum_{\bar{j} \in \binom{N}{i}, \forall j_o, j_l \in \bar{j} : j_o \wedge j_l = 0 \cap j_l \wedge f = 0} \left(\left(\prod_{j_l \in \bar{j}} r_{j_l}^{i-1} \right) \cdot \frac{w_f}{\sum_{k \in F, k \wedge \bar{j} = 0} w_k} \right) \right)$$

In the view of weighted max-min fairness, intuitivelly this algorithm has the characteristic of favoring flows with high priority or good alliances and one expects to close to the objective – we will present later some results. But flows in this algorithm can propagate far in virtual time, if these are scheduled not because of their high priority index, but because the competitors of its competitors have high priority and by so the contention that these face clears out, during the first iterations. If these implicit alliances go away(become inactive), this flow will simply be mistreated for receiving fair service in the past. To avoid this we need a common reference, “a count of fair service” for all flows, that will grow in proportion to the normalized service received by all active flows – much like virtual time does in WFQ implementations.

[76] This discipline has some additional properties that could differentiate thinks, from the simple one resource case

The problem here is, that there is no obvious way to define such a metric, since different resources(outputs or inputs) face different congestion. How can we relate the service of a non-contented flow, with a flow using a congested resource? The difference in normalized (=fair)service received by these two flows during an active period, can grow unbounded in proportion to the backlog period. Now if an other flow becomes active that contents both flows, where should this flow be inserted in virtual time? We want to sustain the GPS memory-less property and treat all flows independently of the past. But the virtual time of the non-contented flow will be far in front of the contented one, so it seems that we cannot solve this problem. The only solution would be to prevent these "timers" from diverging and the only way to do this, is not to upgrade the virtual time of the non-contented flow – but what could be the criterion for this decision? Note that this problem will not occur in the one resource, since all active flows receive the same normalized fair service during active periods[77]. You might suggest to use as common reference the *real* time that has elapsed and upgrade flows next service times relative to this common reference – instead of the previous next service time. This will certainly keep the virtual times of flows close, but additionally will lead to unfairness even in the competition of one resource, since actually with this scheme we cannot differentiate between flows using the same resource.

	$IterWRR$	Buf. CrossBar (buffer size 4)	Buf. CrossBar (buffer size 1)
Average $PAWD$	6.3036	0.043	3.047
Max $PAWD$	59.841	27.94	1.098
Median $PAWD$	0.013	0.012	0.059
Average $PAWD$	0.047	0.192	2.31
Max $PAWD$	0.246	1.65	20.43
Median $PAWD$	0.032	0.08	1.17

Table 4: Comparing $IterWRR$ with the buffered Crossbar under the weighted max-min fairness objective.

Table *Table 4* demonstrates the applicability of the technique in the backlog case. For most configurations, this multi-resources scheduling discipline, proves to behave quite well, at least for these periods of time when flows are continuously backlogged and all start competing as equal(all flows start with the same next_service_ time).

[77] In our view the underlying idea of a viable solution, would be to keep track of the fluid model of the crossbar topology with GPS servers and serve flows with increasing service order in this model. Of course this requires methods and analysis.

5.6.3 Permutations WRR – THE expensive discipline of theoretical interest

In the one resource scheme, the unit of competition is a flow. In the crossbar case, we can think of a greater unit of competition, cartels consisting of flows alliances – each possible permutation of inputs to outputs. One other idea, which has been successfully utilized in [Serp2DRR] in a primitive form, is to consider these permutations as the candidates, into a fair queuing engine.

We evaluated this hypothesis, using an extremely expensive algorithm($PermWRR$). We computed all possible permutation of inputs to outputs, we labeled them and assigned to each one, a weight equal to the sum of weights of the ingredients flows. We then passed these competitors into a fair queuing engine, identical to WRR/WFQ-variant (see section 3.1) and simulated the service that these received.

If we consider for the moment that all flows are backlogged, then each flow f will receive service equal to the service received by all the permutation, containing this flow (M_f). The total number of all permutations M is $|M| = N!$, whereas the number of permutations containing f is $|M_f| = (N-1)!$. In a WFQ discipline, each backlogged competitor c receives service with rate at least equal to $\frac{w_c}{\sum_{c_i \in C} w_{c_i}}$. In the scheme we consider here, a flow f will receive service with rate equal $\frac{\sum_{m \in M_f} w_m}{\sum_{\mu \in M} w_\mu}$.

Now the sum of all permutations aggregate weights is:

$$W_M = \sum_{\mu \in M} w_\mu = (N-1)! \cdot \sum_{f \in F} w_f$$

since each flow belongs to exactly $(N-1)!$ permutations. Similarly the sum weights of all permutations containing f will be:

$$W_{M_f} = (N-1)! \cdot w_f + (N-2)! \cdot \sum_{g \in F\ ; g \wedge f = 0} w_g$$

It follows then, that the rate received by flow f will be:

$$r_f = \frac{W_{M_f}}{W_M} = \frac{(N-1)! \cdot w_f + (N-2)! \cdot \sum_{g \in F\ , g \wedge f = 0} w_g}{(N-1)! \cdot \sum_{j \in F} w_j}$$

.

This result is quite intuitive. It tells us that the rate of a flow will grow when the weight of this flow increases, or when the weights of the no-competitors(alliances)of this flow increases. But can we use this formula to compute the weighted max-min fair rates – or equivalently will the $PermWRR$ algorithm assign the weighted max-min fair shares to flows? For many instances this is true. For example if all flows have the same weight and all flows are backlogged, the above formula reduces to $r_f = \frac{1}{N}$[78]. You can also find, that simple configurations with different weights will result in the assignment of the weighted max-min fair rates. But since

[78] This is why the 2DRR algorithm had good properties

this algorithm may schedule permutations that contain some in-active flows, it is straightforward to prove(by counterexamples) that this algorithm will not assign the weighted max-min fair rates to flows, in the general case. The question of weather some “fix” will make the algorithm to do so remains.

	$PermWRR$	Buf. CrossBar(buffersize 1)	$IterWRR$
Average $PAWD$	0.17	0.14	0.17
Max $PAWD$	0.4	0.4	0.4
Median $PAWD$	0.2	0.1	0.2
Average $PAWD$	11	0.14	0.08
Max $PAWD$	28	0.66	0.2
Median $PAWD$	13	0	0.1

Table 5: Permutation WRR, Comparing Iterative WRR and Buffered Crossbar.

WEIGHTS

1	1	1
2	2	2
2	2	2

MAX-MIN FAIR

0.5	0.5	0.5
0.5	0.5	0.5
0.5	0.5	0.5

PERMUTATION WRR

0.5	0.5	0.5
0.5	0.5	0.5
0.5	0.5	0.5

(a)

1	0	1
1	0	1
1	0	0

0.33	0	0.5
0.33	0	0.5
0.33	0	0

0.28	0	0.38
0.28	0	0.38
0.23	0	0

(b)

Figure 81: The configuration of table *Table 5*

In *Table 5* we can see the results from two different runs, that correspond to the plotted in *Fig. 81* configurations (a) and (b), respectively. In both cases, the permutations algorithm yielded results consistent with the formula above. In case (a) this resulted in max-min fair shares, whereas in case it did not – one obvious reason is that it scheduled permutations containing inactive flows, like flow from input 3 to output 1.

5.6.4 Conclusions/the (un?)supported argument

The intention of this paragraph, was to make a point of the cost that must be paid if we want a non-buffered crossbar to produce weighted max-min allocations. We think that this should be now obvious. Once again, sophisticated QoS and high utilization seem to conflict, somewhere in the middle of the crossbar network-topology/schedule-area and on the QoS specifications. A more profound result, would be that the unbuffered crossbar under any scheduling discipline, cannot assign weighted max-min fair shares, that accurate as the architecture the $WRFC$ does. Although we have not been able to prove it, the question remains open.

6 Discussions/Conclusion

.

6.1 Weighted Max-Min Fairness discussions

The results that we presented related to the weighted max-min fairness objective, are quite pleasing. The system actually converges to weighted max-min fairness when backlog traffic is waiting for service in the input queues of the system. This is critical for backbone routers, that might use the proposed architecture, as it is or within a more complex system. This architecture can also be used in parallel computers, where excellent performance is desired. The switch as proposed, is perfect in interconnecting multiple computing nodes, if point to point interconnection(pure output queuing), is out of the discussion table. Weights can be realized in the application level, like we proposed in the networks objectives section (section 2) and the weighted max-min fairness objective can guarantee the welfare of the application(s) running, especially if good methods are found to form the weights based on the urgency for coummnication, of the applications.

Most of the results that we have plotted, advocate, that a size for the crosspoints near 4 internal packets, will do the work very accurately, for switch scales smaller than 32x32. With only one worth of cell buffer space, the system will not behave like we want and might under-utilize the switch capacity. The type of the servers used at the crosspoints is not actually of much concern, since we do not expect these servers to coordinate in very bad configurations – like the theorem in section 4.4.5 hypothesized. So a simple WRR/WFQ-like discipline at the contention points is adequate, although if this is an (l, k) fair at all then certainly $l, k = O(N)$. The virtual clock discipline, although works better for the one buffer case, is simply not acceptable, since we must give up with other characteristics of the scheduling discipline – like the memory-less property, which is important when flows switches states between active and inactive.

The WF^2Q discipline, is expected to have the lowest (l, k) values as we have commented in the respective section(section 3.1). But although WF^2Q+ works a little bit better in the fairness objective than WRR/WFQ, we would not pay the cost of using this discipline, simply for this small marginal gain. Other reasons may advocate for the deployment of WF^2Q+ servers – like the smoothness in the allocations that it provides, that reminds the service characteristics of RR schedulers, in the no-weights case.

The importance of the enforced discipline on actual networks/or parallel machines, is actually verified in the sections for the transient source(section 4.4 , section 5.3) and flows trajectories. There we showed that the weighted max-min equilibrium, although can delay in time at the worst-case in proportion to the size of the crosspoints and to the size of the switch, this interval is not that long. Especially in the case that simple marginal changes occur in the system, the average delay will be small and inversely proportional to the unfairness experienced during this time. That is the system behaves more or less like an elater; the longer it is pushed from its state of equanimity, the more urgent it is to find its (new)balance again. You cannot suggest to use smaller crosspoint buffers than 3 or 4 in order reduce the system inertia, since then you might not reach the equilibrium in the first place, especially for scales bigger than 16x16. Only manifacturing reasons, can legitimelly oblige you to do so.

6.2 Saturation throughtput discussions

In the sections on throughput(section 4.4.6 , section 5.4) we showed that again a size near 4 cells at the crosspoint buffers, can provide throughput near the ideal, in the case that no weights are used to represent priority. When weights are used and these are formed independently of the arrival pattern(under networking-favored-terms, no policing mechanism is applied on flows), then the performance – in terms of aggregate throughput – of the system degrades, especially in the case that weights are not evenly distributed on resources. The gain is, that you can configure weights, so that you reduce the delay of a flow into anything you want. For one more time we fall upon the argument, that providing QoS and high utilization, is somehow conflicting in input buffered switches. But this is not a legitimate argument against the architecture. Either if weights represent the number of active flows, or the urgency of an application, we expect that this will not hold, since in the first case the arrivals will be in proportion to the weight(we showed that under this assumption the performance is again near the ideal) and for the second, under a good weights assignments mechanism, only some packets will have big urgency degree and anyway weights could change in the purpose of clearing backlogs and not creating them.[79].

We also show, that adaptive methods for the input servers, like longest queue first, can provide better saturation throughput than discriminative methods, but again only when the weights are decoupled from the arrival process. It makes some sense, to use these servers at the inputs, but only if you do not care for the weighted max-min fair obejctive, or for the ability to assign arbirtary small delays to some privileged flows. Anyway with adaptive disciplines at the inputs, you cannot guarantee arbitrary small delay to flows in case that this is desired, since you will have to include a term that grows with N. Additionally LQF, may favor not well behaved flows that continuously sent packets – as long as the backpressure does not protest – and in this way will increase the queuing delay, of more well behaved sources. Generally, the average throughput of the system increases if adaptive methods are used, so the decision for which alternative to adopt is certainly application depen-

[79]the weight of a packet could for instance change the weight of all packets in the respective queue – even for those in front of it.

dent. The good think is, that both disciplines can use the same $schedulingengine$ and by these means, an actual system, can be designed with both disciplines and switch from one to the other, on the fly.

Comparing WF^Q+ with WRR/WFQ, it is certain, that if input servers provide smooth allocation, the backlog/active periods of the output servers increases and in this way the average delay decreases. In this view WF^Q+ should provide much higher performance. But the good news are, that the backpressure method, as it is, obliges input servers to provide somehow smooth allocation and the advantage of WF^Q+ is not that evident, in this system. Probably this discipline could be used at the output contention points, when additional service-elegance is desired.

6.3 Design Complexity

The architecture, that we have analyzed, is quite modular and simple(a look at the figure *Fig. 3.2* , will convice you). All crosspoints have the same structure, and all input and output servers similar implementation and similar/symmetric behaviour. We have seen, that single resource servers are about to become a commodity in hardware design, nowadays. Every crosspoint buffer, can be placed on the same chip with the every crosspoint/transistor, in which case it only needs to run, at twice the external links speed – at most, one read and one write at each cycle. The logic that manages the acceses to these buffers, can be made compact, to fit in the same chip also. In case that a shared memory is used to keep internal memory in autonomous chip(s), then you can work with already proposed solutions. But do not expect, to increase memory utilization in this way, compared to the alternative of distinct buffers at the crosspoints, since then you might challenge the protection of flows and the independence of input servers scheduling. Ofcourse other more complicate schemes, might share a part of memory among flows and still enusre protection/isolation and independence in scheduling. In addition, the pipelined structure of data-forwarding, increases systems delay tolerance and generally enables faster designs.

6.4 Concluding Remarks

The generic architecture that we proposed and analyzed, can be used under various schemes. This is done for example in [DGPS], where delay bounds are considered. Essentially this is one more GPS property of single resource servers, that has been generalized to input-buffered systems, through this architecture. But in providing this efficiency in [DGPS], they actually grant that the throughput guarantee is the minimum of the rates in the input and output servers. By these means, they cannot assign-guarantee, all the available capacity to flows. Here we have proved, that excess bandwidth will be fairly distributed to unsatisfied flows, with no internal speed-up/output buffers requirements, for our and their common, core architecture.

So $WRFC$, can provide near optimal performance, by paying only the cost of placing small buffers at the crosspoints. With the increasing rates that silicon area is available nowadays, there is only a small marginal cost to be paid if we place these buffers in the same chip with the actual crosspoint(s) – in which case, these only need to run with "speed", twice of the external links[80]. This solves all scheduling problems in input buffered switches – that otherwise might tyrannize us for another decade. Compared to the scheme proposed in [PackSwitDes], where internal buffers are shared among many inputs, to increase the efficiency in the internal buffer utilization, our scheme is more simple and in our view provides the desired isolation to flows.

As correctly pointed in [DPGS], this architecture enables the use of variable size packets internally to the switch, since the distributed servers can take their decisions completely independently and asynchronously. This save part of the speed-up that must be paid in any switch that segments packets in the input-interface, to fulfill Anderson proposal of decouling forwarding from scheduling. The speed-up that must be paid in this case is near two. In the context of the implementation costs, the scheme is highly modular and simple, providing the basis for relative cheap and efficient implementation. The pipelined forwarding technique, reduces the delay sensitivity of the system[81].

[80] one enqueue+one dequeue, at each time slot

[81] Many of the advantages of the buffered scheme, can be explained in terms of pipelining efficacy, which is known from processors design.

7 Future work/open issues

There is a number of open issues related to the performance of the proposed scheme. For one a more complete proof that the system will reach weighted max-min fairness, is certainly such an issue. This might require a proof that (l, k) fair servers actually exist, or that WRR/WFQ or/and WF^2Q or/and WFQ are (l, k) fair – with what values for l, k? –, since we have not actually proved that this is the case. An other issue is, the minimum size of the buffers that can guarantee this objective enforcement. A third issue is the development of a more precise characterization for the delay and the unfairness in bits occured in the (re)stabilization process. Although the one that we provided is good, probably is not that well described.

In our view their is an additional point, of weather better service disciplines exists – probably defined in the context of the buffered crossbar, or similar –, that could sustain weighted max-min fairness, under less buffer space; that the Virtual clock did so, by preferring previously inactive flows, is a good hint. The extension of this architecture to handle variable packets internally and the investigation if any performance deficits might be present in this case, could additionally be part of future work.

The two last points/questions are more theoretical. The one has to do with a proof that non-buffered crossbars cannot sustain the weighted max-min fairness, since this discipline is not in the feasible scheduling region, of permutations scheduling. We do not know if this can be actually proved, but it worths to cope with for a while. In this target, the simulation of expensive but promising crossbar scheduling disciplines, is a good first step.

The last and continuously open issue is to find an other, more ecomomic than the crossbar($O(N^2)$) switching/scheduling fabric, that can sustain the weighted max-min fairness objective. This is a very difficult problem, but probably the investigation on the properties that this system/topology should possess, is tractable – for a first examination on other topologies/characteristics, with internal buffering look in [SapKatBenes].

References

[DGPS] Donpaul C. Stephens Master Thesis, *"Implementing Distributed Packet Fair Queueing in a scalale switch architecture"*, In INFOCOM'98, pages 282–290, San Francisco, CA, March 1998.

[ChQoS] Fabio M. Chiussi and Andrea Fancini, Data Networking Systems Research Department Bell Laboratoris, Lucent Technologies, *"Providing QoS Guarantees in Packet Switches"*, Proceedings of IEEE GLOBECOM'99.

[ShNetObj] Sandeep Bajaj, Lee Breslau Scott Shenker *"Is service Priority Useful in Networks"*, In Proceedings of the ACM Sigmetrics '98, Madison, Wisconsin USA, June 1998.

[UtMaxMin] Zhiruo Cao, Ellen 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

[FaCongCont] Jeonghoon Mo and Jean Warland, *"Fair End-to-End Window-based Congestion Control"*, Available as. http://citeseer.nj.nec.com/337047.html

[HahRRMaxMin] Ellen L. Hahne, AT&T Bell Laboratories, *"Round-Robin Scheduling for Max-Min Fairness in Data Networks"*, IEEE Journal on Selected Areas in Communications, 9(7), September 1991. http://citeseer.nj.nec.com/hahne91roundrobin.htm

[HaydenMaxMin] H. P. Hayden, *"Voice Flow Control in Integrated Packet NetWorks"*, Technical Report LIDS-TH-1152, MIT Laboratory for Information and Decision Systems, 1981.

[JaffeMaxMin] J. M. Jaffe, *"Bottleneck Flow Control"*, IEEE Trans. Commun ,vol. COM-29, pp. 954–962, July 1981.

[Kate87] M. Katevenis: *"Fast Switching and Fair Control of Congested Flow in Broad-Band Networks"*, IEEE Journal on Selected Areas in Communications, Vol. SAC-5, No. 8, October 1987, pp. 1315-1326.

[KaSC91] M. Katevenis, S. Sidiropoulos, C. Courcoubetis: *"Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip"*, IEEE Journal on Selected Areas in Communications, vol. 9, no. 8, October 1991, pp. 1265-1279.

[ParGalPGPS] 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.

[DemKesShenWFQ] A. Demers, S. Keshav And S. Shenkar, *"Analysis and simulation of a fair Queuing Algorithm"*, Proc. of ACM SIGCOMM, 1989, pp. 3-12.

[ZhangVC] L. Zhang, *"A new architecture for packet switching networkprotocols"*, MIT PhD Thesis. 1989.

[BenhZhangWFWF] Jon C.R, Bennett, Hui Zhang, *"WF^2Q: Worst-case Fair Weighted Fair Queuing"*, In Proceedings of INFOCOM '96, San Francisco, CA, March 24-28, 1996 http://citeseer.nj.nec.com/zhang96wfq.html.

[Keshav97] S. Keshav: *"An Engineering Approach to Computer Networking"*..

[HierFairQ] Jon C.R. Bennett and H. Zhang, *"Hierarchical Packet Fair Queuing Algorithms"*, In Proceedings of SIGCOMM'96, pages 143–156, August 1996.

[AnderPIM] Thomas E. Anderson, *"High Speed Switch Scheduling for Local Area Network"*, ACM Transactions on Computer Systems, November 1993.

[SLIPMcKeownAndersComp] N. McKeown and T.E. Anderson, *"A quantitative comparison of scheduling algorithms for input queued switches"*, published 1997. http://citeseer.nj.nec.com/mckeown97quantitative.html

[SerpFIRM] D. Serpanos, P. Antoniadis *"FIRM: a Class of Distributed Scheduling Algorithms for High-Speed ATM Switches with Multiple Input Queues"*, In Proceedings of IEEE Infocom 2000.

[Serp2DRR] R. LaMaire, D. Serpanos, *"Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues"*, IEEE/ACM Transactions on Networking Vol.2, No. 5 (Oct. 1994), pp. 471-482.

[StilVarmWPIM] D. Stiliadis and A. Varma, *"Providing bandwidth guarantess in an input-buffered crossbar switch"*, n Proc. IEEE INFOCOM '95, April 1995.

[FIM] Nan Ni, Lax N. Bhuyan, *"Fair scheduling for Input Buffered Switches"*, citeseer.nj.nec.com/482342.html.

[Kumar98] V. Kumar, T. Lakshman, D. Stiliadis, *"Beyond Best Effort: Router Architectures for the Differentiated Service of Tomorrow's Internet"*, 1998. http://citeseer.nj.nec.com/kumar98beyond.html.

[Linear] Anthony C. Kam, Kai-Yeung Siu, *"Linear complexity algorithms for QoS support in input-queued switches with no speed-up"*, Technical report, d'Arbeloff Laboratory for Information Systems and Technology, Massachusetts Institute of Technology, Cambridge MA, 1998, http://citeseer.nj.nec.com/kam98linear.html.

[LOOFA] P. Krishna, N. Patel, A. Charny, R. Simcoe, *"On the Speed-up Required for Work-Conserving Crossbar Switches"*, In IWQoS 98, 1998.

[In2OutSpeedup] S. Chuang, A. Goel, N. McKeown, B. Prabhakar, *"Matching Output Queuing with a Combined Input Output Queued Switch"*, IEEE Journal on Selected Areas in Communications Volume 17, Number 6, June 1999.

[HarteKatevCompTree] Kostas G.I. Harteros *"Fast Parallel comparison circuits for scheduling"*, Technical Report ICS/FORTH ID Number 304, ftp://ftp.ics.forth.gr/tech-reports/2002/2002.TR304.Fast_Comparison_Circuits_for_Scheduling

[DGPS1] D. Stephens, Hui Zhang: *"Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture"*, In INFOCOM'98, pages 282–290, San Francisco, CA, March 1998. http://citeseer.nj.nec.com/stephens98implementing.html.

[MagBufFullCap] Tara Javidi, Robert Magill and Terry Hrabik, *"A High-Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric"*, Proceedings of IEEE.

[Kat-534] Manolis katevenis *Lectures on Packet Networks Architecture.* http://archvlsi.ics.forth.gr/ kateveni/534/

[PackSwitDes] Cyriel Johan Agnes Minkenberg Thesis, *"On Packet Switch Design"*.

[Stephens99] D. Stephens, J. Bennett, H. Zhang, *"Implementing Scheduling Algorithms in High-Speed Networks"*, To Appear in IEEE JSAC, 1999. http://citeseer.nj.nec.com/stephens99implementing.html

[IoanKateHeap] A. Ioannou, M. Katevenis. *"Pipelined Heap (Priority Queue) Management for Advanced Scheduling in High-Speed Networks"*, Proc. IEEE Int. Conf. on Communications (ICC'2001), Helsinki, Finland, June 2001,pp. 2043-2047.

[SapKatBenes] G. Sapountzis, M. Katevenis *"Benes Fabrics with Internal Backpressure:First Work-in-Progress Report"* , Technical Report FORTH-ICS/TR-303,Institute of Computer Science, FORTH, Heraklio, Crete, Greece, March 2002; http://archvlsi.ics.forth.gr/bpbenes

8 APPENDIX A

.

8.1 Why dynamic weights – or weight per packet

As it may be clear from the above paragraphs, the final objective of network engineering as a whole, translates into a perceptual and context depended criterion of *best* servicing network demand. Here we analyse how this criterion could be formed in the general case. Our objective class hides of-course some initial hypotheses. So the objectives that it contains, share some common characteristics, in terms of initial hypotheses, regarding the conceptualization of the criterion, of *best* servicing demand. These hypotheses are build upon assumptions on the marginal utility that *applications* gain/or loose, when these are serviced by a network. It is made general enough, so that it can be parametrically instantiated, under different view(s) on the nature of efficiency.

8.1.1 Flows, applications and *demand*

In the target of formulating a class of objectives, we need to consider a basic element of demand. Usually flows are considered to provide this base, but here we will try to convince you that this approach is either simplistic, or full of assumptions. Firstly we review the notion of the flows and try to exemplify the reasons which in many cases, have correctly enforced a flows-driven approach of networking. Then in the view of applications, we reconsider.

Flows The networking view of flows is very wide, depending on what you consider as *source* and *end* at each time or place. So we may consider as a different flow a point to point communication between two nodes, or an end to end applications message exchange. Flows are a way of *grouping* data packets in term of temporal and spatial locality factors. A flows is a stream of data(packets), that stems from a source-node and is destined to a end-node. This stream of data-packets shares at least some commons characteristics:

- common routing.
- common QoS needs.
- delivery in order?

By these means flows can be used to make faster routing decisions and to demand constant QoS. In addition flows can be merged(*aggregation*), or splitted by the network and then regrouped(*inverse multiplexing*), so that these take better advantage of the network resources.

In the applications view flows are a helpful tool, for sustaining multiple communications “online”[82] and a mechanism to split the communication interface of the application in independent components, which are serviced according to the application needs. Parallel application for example, may use different flows for communicating control or data signals, different flows for communicating events-flags and different flows for communicating the actual data. Although both the networking and the application view utilize the same characteristics of flows, we will use the application view for formulating the class of objectives – which we have promised –, primarily because the application view can stand independently of a network and also because applications provide a framework on which the demand and the utility gained when serviced by the network, can be more intuitivelly understood.

Flows defined as a unidirectional stream of data, do not provide the means for realizing demand in a dynamic environment. Not all packets of a flow, are equal. This would require that a each packet sent, belongs on a different flow. What flows mainly provide in the application view, is a set of common *static* QoS needs and probably routing. In the context of the ends of a parallel/distributed application, which coexist and coordinate though a common infrastructure of changing overall demand, a simple and static partitioning of communication is not always adequate to provide the means for representing/or clustering demand, in terms of these quality of service needs, which when considered by the network, will make the application better off. That is, the first and the last packet of a flow, need no, or better should no share the same static service, since this highly depends on the state of the application ends, which again is dependent on the dynamic environment of the interaction.

We can distinguish two basic types of communication between application flows. The first type is *feed-wise*, because we can distinguish the roles of the application ends into a a server feeding and a client accepting data[83]. The other is *lockstep-wise*, since the communication needs handshaking[84]. That is under *lockstep-wise* communication, both ends need to receive some signal before transmitting the next one. Note that depending on the level of abstraction on which you realize networks, you may recognize both types of communication as coexisting on different layers[85], but here we focus on the application level.

Normally application communication patterns, are a mix of these two basic types. That is some packets needs handshaking(or a reply) and some need feed-wise communication. We cannot expect that the parallel application will statically partition communication, so that each flow is either of the first or second type, simply because this is a statically intractable problem, in the general case. The marginal utility that an application gains, when being serviced by the communica-

[82]Are used in a way, similar to the way(s) streams are used in software programming.

[83]Examples are the emerging streaming applications, ftp services, or...

[84]Parallel programming barriers, synchronization flags and acknowledgments are typical examples. Also protocol handshaking (HTTP), can be represented by the lockstep wise type

[85]and usually the lockstep, if we get down on the details

tion medium should, by the same means, be dynamically formed.

Priorities and marginal utility Every flow of an application may have a priority index, which could be based on the nature of the application, or the money paid for capacity. Moreover depending on the application state, this priority index may change from time to time, that is this priority need no be the same for all the packets of the flow. Remember that the state of the application, depends on *this*[86] and on the other(s) application end(s), on the bottlenecks on the path of the communication(that changes) and on the *nature* of the distributed application(s).

If it is to relate, the marginal utility that an application gains when one of its packet is being serviced, to the effectiveness of the network, the marginal utility must reflect both the dynamically formulated priority index and the actual service received.

We will first formalize our view on the marginal utility that applications gains when one of its packets is being served, then we will sum up these marginal utilities to find an application's flow utility metric and then we will relate the utilities gained by the individual application-flows, to a effectiveness metric of the network overall.

8.1.2 The application flow, perspective of utility

If we consider the networking time as a series of discrete time intervals, which represent the transmission time of single packet between two nodes, then clearly an application flow may or not have a packet for transmission, on the beginning of every time slot. This may depend on the other end(s) of the applications, the service that this flow received in the previous time slots and of course on the application by its self. Moreover, we may consider that a priority index is created by the application at the beginning of each time slot, for each packet to be transmitted which also depends on the service received in the previous slots, the application nature and the other end(s) state. Adopting a rather greedy view on behalf the applications, we don't count here for congestion indexes that may form this priorities index, in the view of a specific network configuration – similar to what happens with TCP/IP, where unsatisfied flows simply withdraw their demand. Each packet belonging to an application flow *wants* to receives the best, in the goal of prioritizing the application and enhancing the communication with the other end.

We denote by $\mathcal{MU}_{t_i}^f$ the marginal utility that an application flow realizes, at the end of the time interval t_i. $\mathcal{P}_{t_i}^f$ is the priority of the packet to be serviced at time t_i and $\mathcal{B}_{t_i}^f$ is the bandwidth that this packet "received" from the network.

If this packet is serviced by the medium, we consider the marginal utility $\mathcal{U}_{t_i}^f$, to be positive and proportional to the bandwidth $\mathcal{B}_{t_i}^f$ that this packet received, in its path from source to drain – which is the other application end–, and inversely

[86]We consider uni-directiona flows, so *this* is the source.

proportional to the priority of this packet. If at the beginning of a time slot, there are no packets for transmission, then we think of the marginal utility to be equal to zero[87].

$$\mathcal{MU}^f_{t_i} = \mathcal{B}^f_{t_i} \ / \ \mathcal{P}^f_{t_i}$$

if flow f has a packet for transmission on the beginning of time slot t_i, or

$$\mathcal{MU}^f_{t_i} = \mathcal{O}$$

if there are no packets, for transmission on the beginning of time slot $\sqcup_i$.

Now given a finite discrete time interval $\mathcal{T}$, the utility that a flow received will be the sum of the marginal utilities gained at each time slot within this interval. That is

$$\mathcal{U}^f_{\mathcal{T}} = \sum_{\mathcal{T}} \mathcal{MU}^f_{t_i}$$

Some comments can be made about this formula. First the marginal utility depends on the priority. This priority index should be perceived in the context of the application welfare. The application wants to send a packet, so that it can for instance continue execution and by so this packet has a high priority index. On the other hand, the utility of the application may not be affected on the time that this packet is send or the bandwidth that it was allocated and by so this priority index is small. Or the application flow may want to receive small jitter and by so it sets a medium priority on each packet(and reforms it dynamically with small perturbations), so that the jitter goal is realized.

The other issue which needs some discussion, is the bandwidth parameter $\mathcal{B}^f_{t_i}$ (or marginal bandwidth) that each packet receives from the network. It should be clear, that this bandwidth is actually realized in many time slots(RTT), as many as is required for the packet to reach the other end and some information to come back. But since this packet transmission is started – although not committed, since the transmission may fail somewhere on the road–, we assume that the bandwidth that this packet receives, is somehow *divinely* perceived by the application. The final issue has to do with the formulation of the priority $\mathcal{P}^f_{t_i}$ and also has to do with the realization of the instant bandwidth. How can this priority index be formulated at every time slot, based upon the service of the previous time slots when this service can only be estimated[88].

Anyway what we want to put here is an effectiveness metric, which does not have to be absolutely realistic, but certainly ideal in the view of our hypotheses. Note that although this definition as it is formulated so far, assumes the existence of some entity which holds a priory knowledge of the future service(RTT(s)...), or

[87]If a packet is not served this time slot, although it wanted service, we could think that the utility of the application flow is reduced by an amount proportional to the priority index, but our definitions of marginal utility through dynamic priority, can accommodate this cost, since we expect priorities to grow when packets are delayed.

[88]The question of how could such a priority index be based on the relative priority of the competing flows, when it is not clear at start, who will be competing is not an issue here since we perceive priority to be application based(a way to index demand levels) and not as a deterministic index for the qualitative of service (although it should be used as hint)

external demand, it comes more natural, in the case of point2point communication through switching fabrics. The objective class formulation, is completed in section 2 of this thesis. Here we primarily wanted, to make clear why the theoretical formulation was driven to consider dynamic weights.

9 APPENDIX B

.

9.1 Virtual Time function Plots for WFQ like servers.

In this section, we plot the progress of the virtual time functions, which is used to estimate, the progress of work in the fluid model. By the term work, we mean the normalized-service received by all fluid-flows. That is when a flow with weight 2, competes with a flow with of weight 1, both receive the same normalized service during any time interval T, where both are backlogged, which is 1/3 of the total work provided by the server during T. So the unit of work is 1/3 per time_slot, which is the incremental utility that each flow receives at each time slot[89].

WRR/WFQ computes this amount of work based on a properties of the packet system. That is, it considers as the normalized amount of fair service received by all flows, the next_service_time of the flow under service. If two or more flows, have the same minimum_next_service_time, then the virtual time will not progress until all flows are served. Of-course in the fluid model, there will be progress in the virtual time(normalized service). So a flow, that becomes eligible in this time interval, might steal some fair bandwidth from the other flows.

More complicate virtual time function, such as the one used by WF^2Q+, although do not emulate the fluid model, make more elegant approximation of the virtual time. In the WF^2Q+ scheme, the virtual time at time slot $i+1$, is the maximum of the minimum virtual_start_time at time $i+1$ and the virtual time at time i plus 1/(total_weight_of_active_flows_in_the_packet_system). In general GPS related disciplines differ in two thinks. The first is the packet selection policy. In this area you can find disciplines using SmallestFinishFirst (WFQ, WRR/WFQ-variant), SmallestEligibleFinishFirst (W^2FQ, W^FQ+) SmallestStartFirst. The second is the way, that the current virtual time is computed, which is crucial for the reschedule of previously inactive flows. Here you can find alternatives, ranging from full fluid model emulation, to others using the smallest finish time or the smallest start time of the active flows in the packet system, or hybrid methods like that used by WF^2Q+.

For those interested, we plot some diagrams from the execution of WRR/WFQ and WF^2Q+, in single resource configurations. In *Fig. 82* we show the reinsertion of inactive flows into fair competition, under the WF^2Q+. Notice that while a flow is inactive, its virtual_start_time is constant. Also notice, that when flows become inactive, the virtual time function increases its slope, since the remaining active flows, receive more normalized fair service at the fluid world, when

[89]You can think of it, as the service that each sub flow – a flow with weight w carries w sub flows – receives. Then all sub-flows in the system receive the same actual service. Or as the service, that each flow receives, for each degree of urgency that this carries – again equal for all flows.

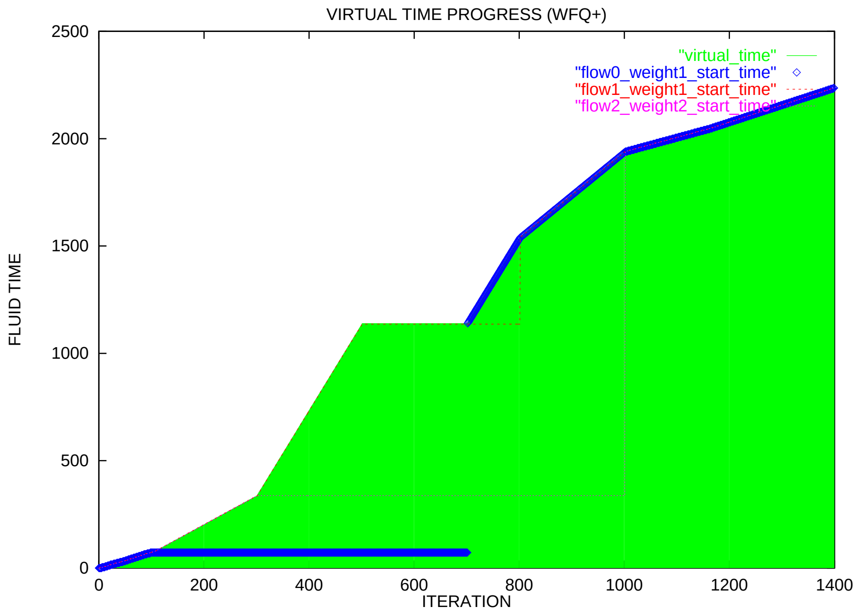


Figure 82: Flow 0 has weight 1, flow 1 has weight 1 and flow 2 has weight 2. You can compute the slope of the virtual time, based on the weights of the active flows.

there are less flows to compete with.

Fig. 83 plot an execution for a configuration identical to that described in *Fig. 6* , under the well-behaved WF^2Q+. Remember that flow F, receives 0.5 of the bandwidth and is smoothly served every second time_slot, while all G_i flows receive 0.05 of the bandwidth and are served every 10 time_slots.

Fig. 84 shows the difference between WRR/WFQ's and WF^2Q+'s progress of virtual time functions, for the same scenario with *Fig. 83* . This logscale figure, demonstrates one more characteristic of virtual time functions, specifically that in more coarse-scales, the virtual time progresses continuously through time, when in more fine-grained scales, the function is actually not continuous. The same holds for the service actually received by flows. In coarse scale, the service closely reassembles that of the fluid model, where when examining small time intervals, the percentage distance/discrepancy can reach infinite.

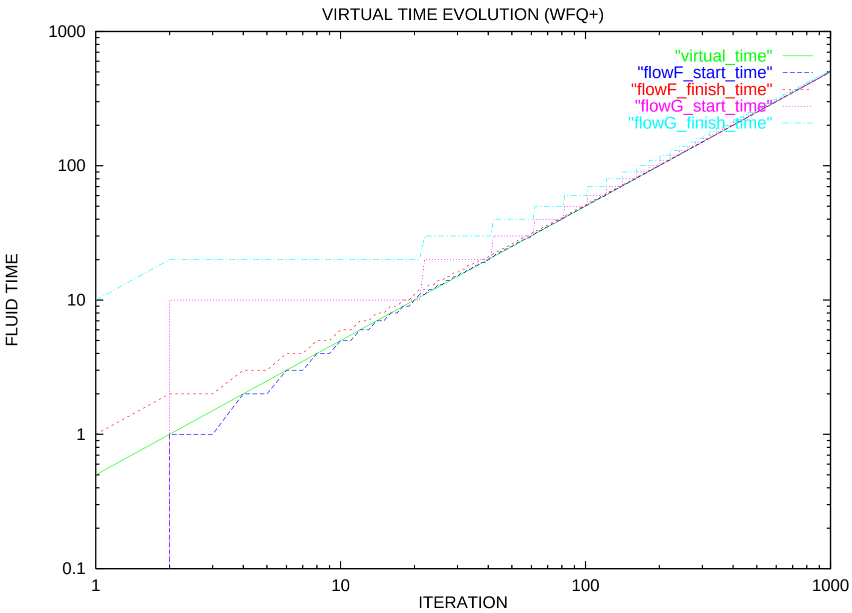


Figure 83: This is the example of an execution of WF^2Q+ with the configuration in *Fig. 6* . Note the times where flow F is inactive, because its virtual start time is smaller than virtual time, so this flow is not served although this has the smaller virtual finish time

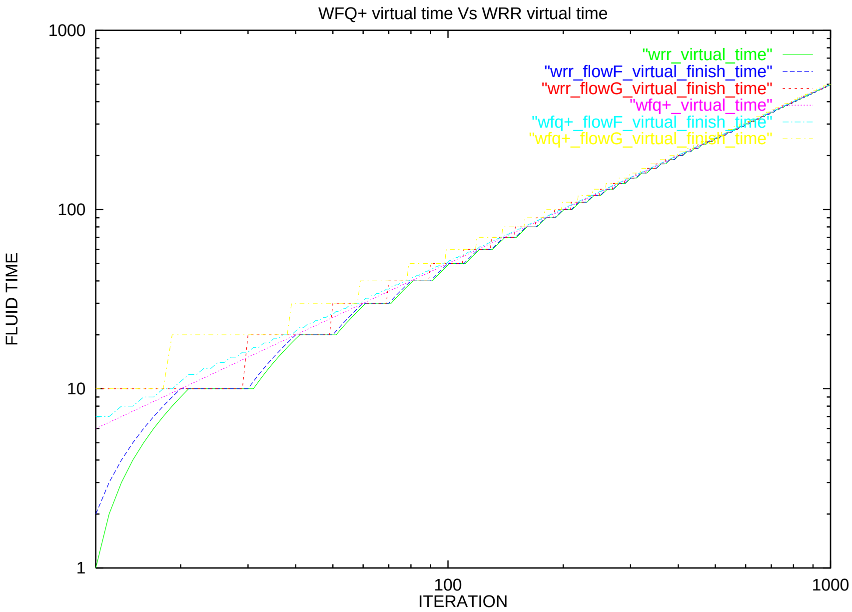


Figure 84: Here we plot together the virtual times of the WRR/WFQ and WF^2Q+, for the example *Fig. 6* . You can see that at times that the virtual time of WRR/WFQ is constant, the WF^2Q+ virtual time progresses.

10 APPENDIX C

.

10.1 More close examination to simulation results, on Weighted Max-Min Fairness

In appendix C, we present more figures that might reveal to an observing reader, hidden properties for the system under examination.

10.1.1 Scaling . . . performance

First we evaluate, if our simulation results on weighted max-min fairness, “scale” to greater switches.

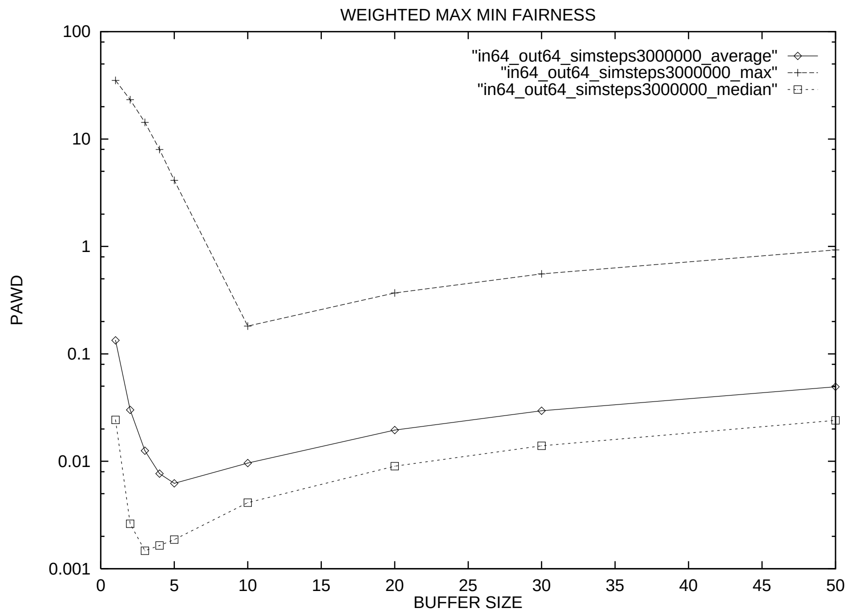


Figure 85: A uniform weights assignment, for a 64x64 switch.

Fig. 85 and *Fig. 86* , plot the $PAWD$ for 64x64 switches, under different topological distribution for the weights. Compared to smaller scales, we can see that the in order to minimize, the maximum discrepancy, we need to deploy greater buffer space, which is consistent with the theoretical analysis on the convergence of the system, since (l, k) values, certainly increase with scale. So it seems, that a (sub-?)linear increase of the buffer space, can sustain the performance characteristics of the architecture at larger scales.

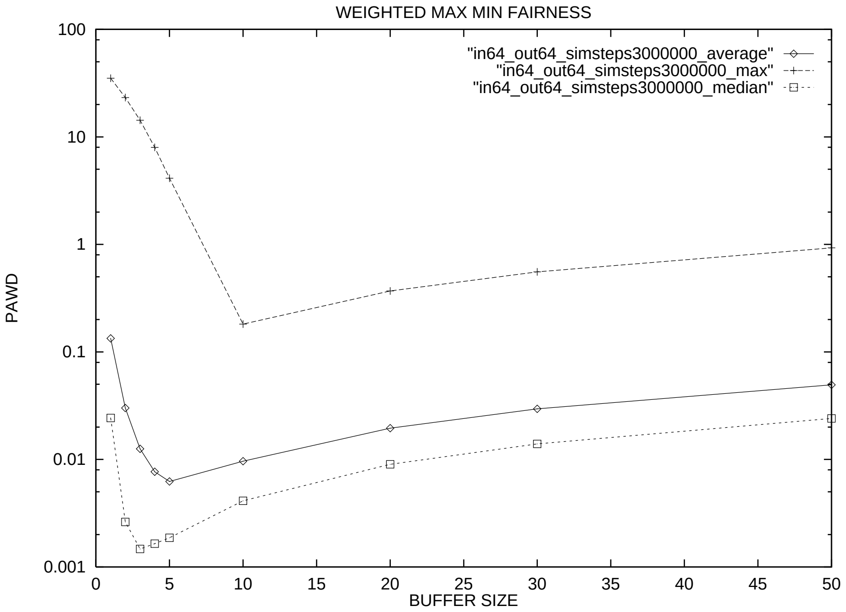


Figure 86: A hot-spot weights assignment, for a 64x64 switch.

10.1.2 More close examinations 1

In this section we present the evolution of weighted max-min fairness, using the $PAWD$ metric, in per flow basis. The results are for configurations presented in the simulation section.

The plots in *Fig. 87* and *Fig. 88* , correspond to the hot spot(configuration 5) of the simulation results, where flows had weight proportional to the output of destination. Both figures plot the evolution of $PAWD$ metrics for the individual flows, with one and four crosspoint buffers size respectively. We can see that discrepancies occur in flows regions. This is rational, since if one flow fails to reach weighted max-min fairness, all the flows in the trajectory of this flow might be affected.

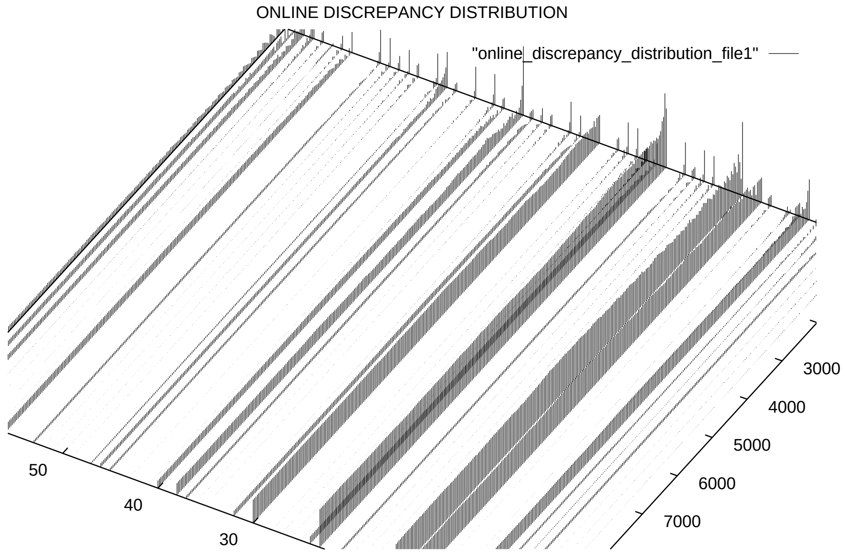


Figure 87: Online distribution of the $PAWD$ for individual flows, with 1 cells buffer space at the crosspoints: configuration 5, hot-spot.

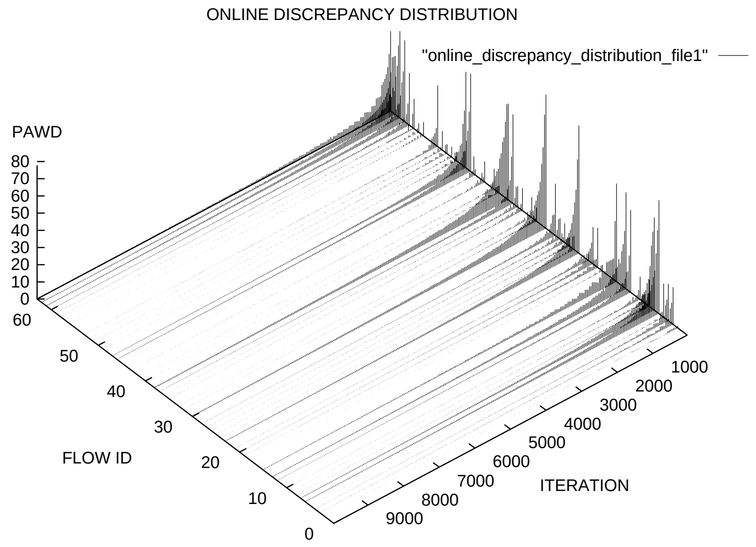


Figure 88: Online distribution of the $PAWD$ for individual flows, with 4 cells buffer space at the crosspoints: configuration 5, hot-spot.

10.1.3 A more close examination 2

In the second subsection, we present an example of a simple configuration, where using only one buffer size at each crosspoint, is not adequate for sustaining the objective. We have seen that weighted max-min fairness, requires buffer space to be enforced. Here, in this paragraph, we will focus on a simple configuration and we will try to observe what actually happens in the system and one buffer space, turns out to be not adequate, in sustaining the fair allocation.

1	11	11
3	21	21
1	21	47

Table 6: Service intervals of the flows. Config 6

In *Table 6* we present the service intervals configuration for a 3x3 switch. The i row represents the service intervals of the flows stemming from input i, while the j column holds the service intervals of the flows destinated to output j. Remember a flows service interval is defined to be inversely proportional to this flow's weight.

0.84 / 0.42	0.07/ 0.48	0.07/ 0.56
0.07/0.14	0.11/0.25	0.11/0.29
0.93/0.42	0.04/0.25	0.01/0.13

Table 7: Input/Output fairness for configuration 6.

0.42 / 0.42	0.28/ 0.28	0.28/ 0.28
0.14/0.14	0.35/0.35	0.5/0.42
0.42/0.42	0.35/0.35	0.21/0.21

Table 8: max-min fair rates/ allocated rates under $WRFC$, with buffer 1.

In *Table 7* we show the input and output fair rates of flows, when the respective servers are considered in isolation, whereas in *Table 8* we present the weighted max-min fair rates and the allocated rates by the system with one size of buffer at the crosspoints.

As we can see – and in *Fig. 89* –, the flow from input 2 to output 3(ID 5), failed to reach its fair rate. All other flows did make it, which implies that the missed services of flow 2_3 resulted in decreased occupancy. This is exactly what happened in the system; 717 unoccupied cycles for output 3 where simulated, when under ideal fairness all outputs would remain continuously busy. As we can see from the tables, flow 2_3 should utilizes great amounts of excess bandwidth, but due to unlucky sampling this flow did not make it to remain backlog at the

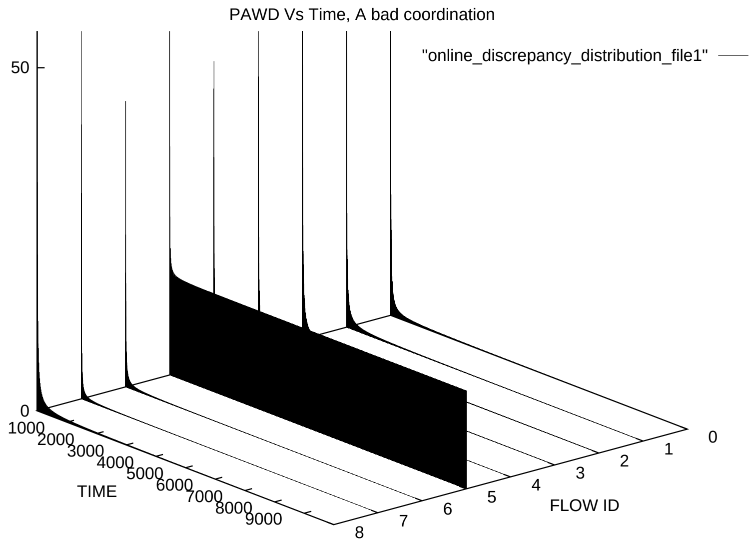


Figure 89: The $PAWD$ metrics for the above scenario. Flow 5 does not reach weighted max-min fairness.

input server – which is it's backlog node under weighted max-min fairness – at the correct moments and missed its opportunistic service, scheduling chances. In fact we have computed during the simulation that this flow is backlogged(active) at the input server, only for 80% of time. What actually happened, is that the input and the output servers have coordinated in a way, that the input finds inactive this flow when it wants to service it, because the output has not serviced it "in time". We cannot really eliminate these phenomena, without using more buffering.

At *Fig. 90* , we can see that the service trace, is periodic – what else would it be. There are periods where both the input and output servers delay the service of the newly eligible flow 5 as much as possible – meaning when the corresponding disciplines, break scheduling "ties", where no actual preference for some flow is present and by so the selection could be random(is predefined in the simulation model) –, equal to 3 time-slots(= the number of contenting flows). Thus each server, blocks, the respective at each case complementary server, from giving more(opportunistic, but fair)services to flow 5 – there is only one buffer space.

10.2 A more close examination, on saturation throughput and delay guarantees.

In this final section, we present results that simply confirm, that LQF, although simetimes better in minimizing average delay, cannot be used to assign arbirtary small delays to flows.

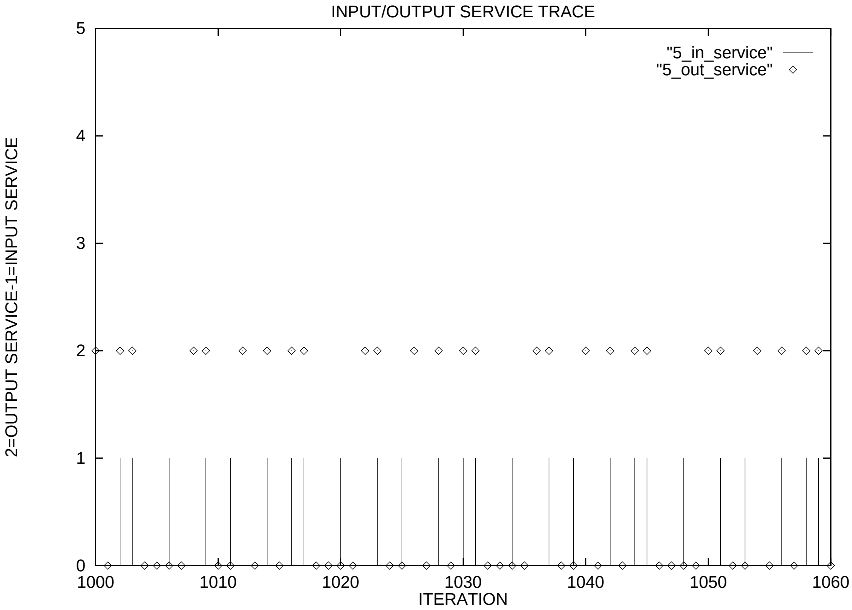


Figure 90: a A trace of the services received by flow 5. With points at ylevel 2, we mark output services and with impulses till ylevel 1, input services. The configuarion is for one buffer space.

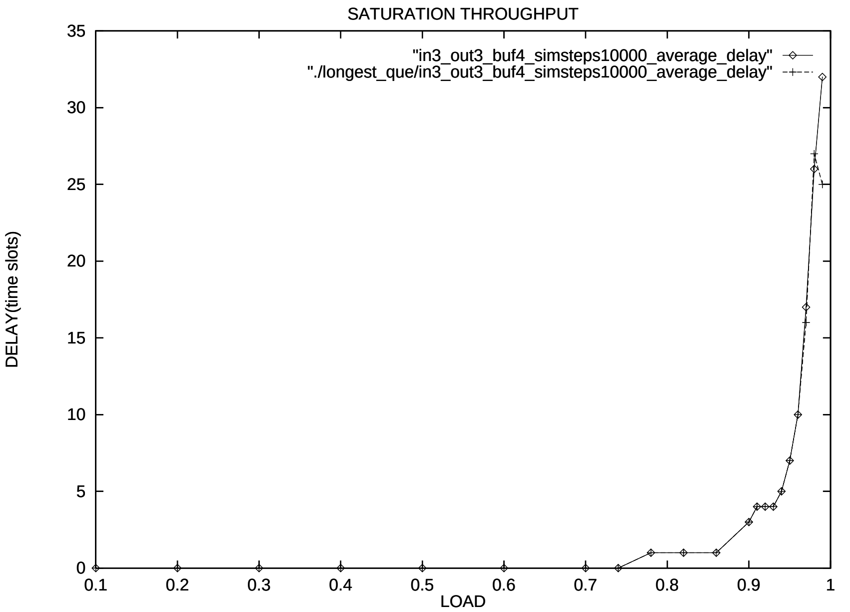


Figure 91: Saturation throughput, for the configuration shown in *Table 9* , with 4 cell buffers space at the crosspoints. Bernoulli i.d.d arrivals, equal for all flows.

18/1	18/1	1/18
18/1	18/1	18/1
18/1	18/1	18/1

Table 9: Service intervals/weights of the flows.

Table 9 , presents the configuration that we have created. Flow with id 2, from input 0 to output 2, is guaranteed 0.9 shares at both the input and output servers. So this flow should realize minimal delays, when its arrival rate is less than 0.9. In *Fig. 91* , where we plot the average delay of all flows under the WRR/WFQ and LQF disciplines for the input servers, all flows have equal arrival rates(=0.33). Although in this scheme both disciplines behave almost identically – with the saturation throughput objective in mind –, LQF performs a little better in this configuration and much better in most configurations. But the delay distribution to flows, is not what we may have anticipated.

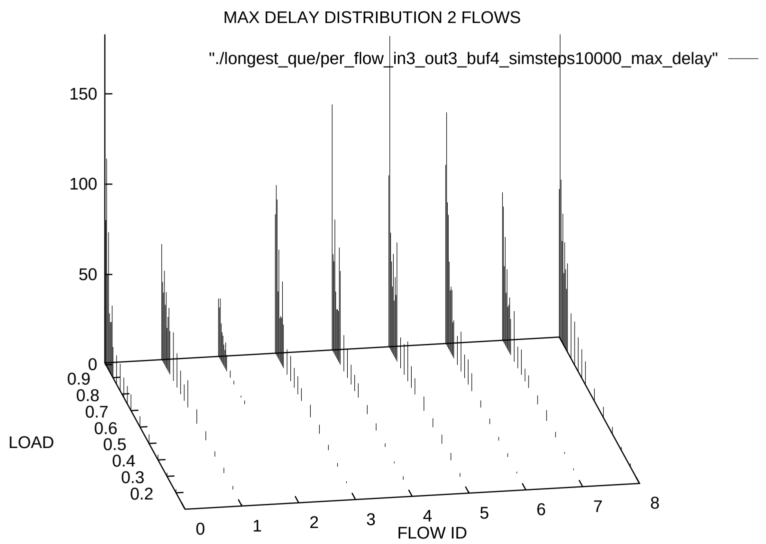


Figure 92: Per flow max delay, under the LQF.

As we can see in *Fig. 92* and *Fig. 93* , only WRR/WFQ can minimize the maximum experienced delay of flow with id 2. LQF generally gives priority to "bad" behaving flows, independently of their weights, as long as the backpressure, permits to do so. If a flow has small weight, then we expect that output fairness will stop input servers from servicing this flow, even under LQF and some form of discrimination/fairness will dominate. But this is a complicate analysis and does not apply in all cases; if, for example, the respective output of this "demanding" / "misbehaved" flow is contention free, then this flow will affect other, more urgent, input neighbor flows. As we have said earlier, the delay bound under LQF will

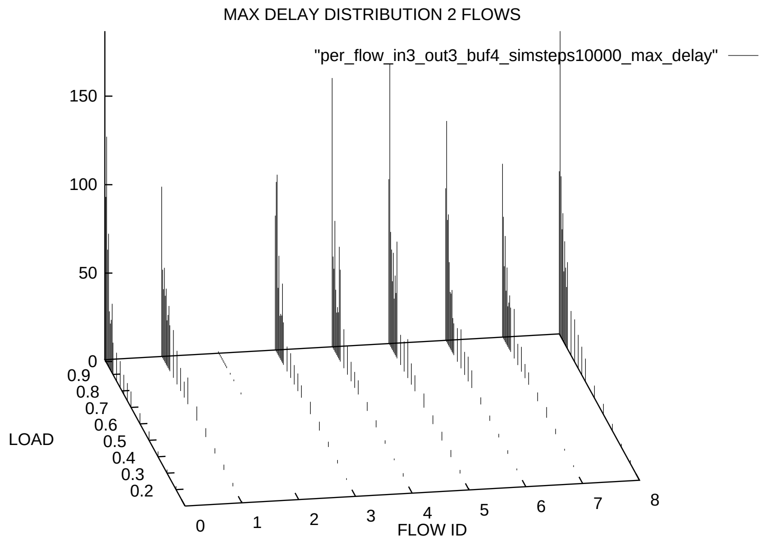


Figure 93: Per flow max delay, under the WRR/WFQ.

certainly contain, at minimum, a $O(N)$ term.

Anyhow, LQF is near ideal in maximizing occupancy for an input buffered switch with internal buffering([MagBufFullCap]) – that is a system more or less like ours – when traffic comes independently of weights. This should be no suprising to occur, in real "shaped-traffic" networks, when the sub-optimality of the weights assignment scheme does not really realize the intented target, or even in schemes where the rate/allocated-capacity of flows is considered independent of weights and weights only stand for determining the urgency-degree/experienced-delay of distinct packets.

11 APPENDIX D

.

11.1 ADAPTIVE-WRR/WFQ in the buffered crossbar context

.

11.1.1 Motive/Rationale/Description

We have seen, that when adaptive methods are applied in the input servers, then the system provides better saturation throughput in the general case, than when WRR-like disciplines are used. We have disclosed some of the reasons in previous sections. Here we propose the adoption of a hybrid scheme, that is both and adaptive and discriminative. This alternative and innovative-variant service discipline, that "fits"/is-well-defined in the context of the buffered crossbar, has as follows. We will describe it here through the WRR/WFQ terminology.

Description It needs/uses/utilizes two "selection rounds".In the first round, flows that have queued cells at the crosspoints are excluded from the selection. So the algorithm tries to find the flow with the minimum next service time *that has no cell queued at the crosspoints*. If nothing is found in the first round, then all flows are included, and in the second round a normal selection is made. This is the only part, where the two algorithms differ. The logical sequence of steps of this algorithm, is the following:

1. *D*rag flows that their next_service_itme is left behind virtual_time, to virtual_time.
2. *S*elect the active flow with the minimum next_service_time, that has no cell in the crosspoints.
3. *I*f nothing is found in the first step, select the active flow with minimum next_service_time.
4. *U*pdate virtual time to the time of the selected flow and update the next_service_time of the selected flow, by adding the respective service_interval.

Although two logical runs of the WRR/WFQ engine are present, a simple OR gate telling us if the crosspoint buffers of all active flows are full, will inform us which set of flows to regard as eligible from the start. This greatly reduces complexity – almost equal of the simple WRR/WFQ scheme–, especially if we use a binary tree of comparators for the find minimum, set operation([HarteKatevCompTree])

– a heap organization of the eligible flows next_service_times, would require, N deletions in the worst-case, which is prohibit.

Rationale It is not difficult to under-stand the rationale, that drove us to think this alternative. We certainly want discrimination in the inputs, in order to sustain differentiation globally in the system, but on the other hand we do not want to keep output servers inactive when there is no reason to. We have seen how this situation can occur through differentiation(= bursty/continous services to some output by one input server) *and* bad-synchronization(= “conspiracy”/independent-probabilistic-events, where all inputs repeatedly select the same (congested)output, when other outputs could have been selected – section 4.4.6 . This hybrid scheme, compromizes between these two objectives, in a very effective manner – the first simulation results are very promising –, as we shall see next.

11.1.2 Comparing Results

Next we plot two results for this discipline. One related to weighted-max-min fairness and one related to saturation throughtput, comparing the scheme, with traditional WRR/WFQ.

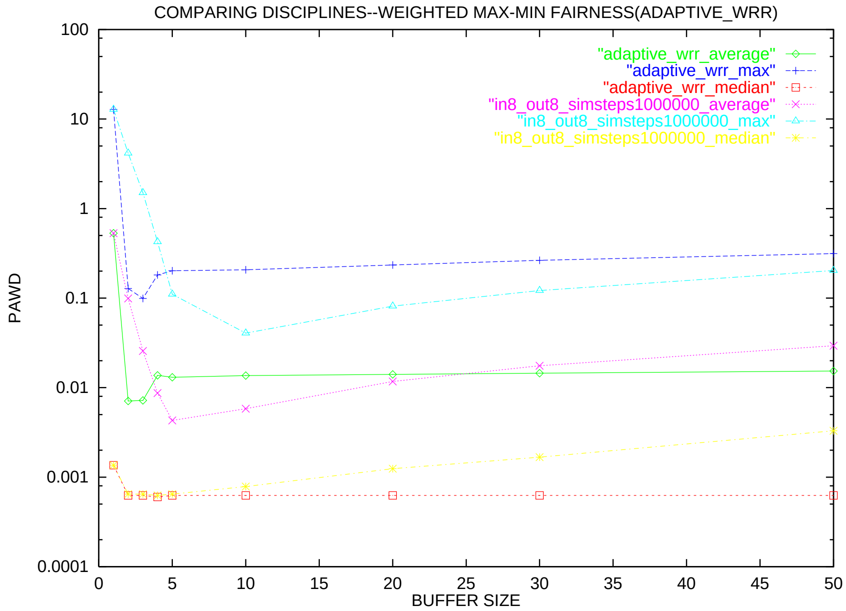


Figure 94: Aggregate $PAWD$ metrics, under the two schemes(WRR/WFQ at both input, outputs servers, Vs $AdWRR/WFQ$ at inputs, WRR/WFQ at outputs). This configuration is for a hot-spot like distribution of weights. The other parameters are shown in the figure, through the labels that correspond to the traditional WRR/WFQ system(in8_out8_simsteps1000000*). Y-axis logscale.

We can see that the system, closes the fair objective well enough both with $AdWRR/WFQ$ and WRR/WFQ. As a matter of fact, for small buffer size, the $AdWRR/WFQ$ works even better, since the injection that we applied to WRR/WFQ will have as effect, flows that face bottleneck at the input, to grab any scheduling chances at the outputs as soon as possible – these will have empty crosspoint buffers, so will these be “always”, elegible in the first selection round.

One second observation that we made running experiments with $AdWRR/WFQ$, is that the occupancy of the crosspoint buffers, does not grow over one cell. This is a good characteristic, since for one it again tells us that the requirements on internal memory are reduced, and in addition, and probably even more importantly, the system inertia is reduced dramatically. We have presented an estimate for the time to converge: $O(\frac{N \cdot Buf\_Size}{min(old\_rate)})$. In this estimate, we made the assumption, that the size of the crosspoints are full, or should become full, in order to approximate the Weighted Max-Min objective. This is true with WRR/WFQ, or WF^2Q+, but is not true under $AdWRR/WFQ$ – only 1 buffer space is used, by most flows.

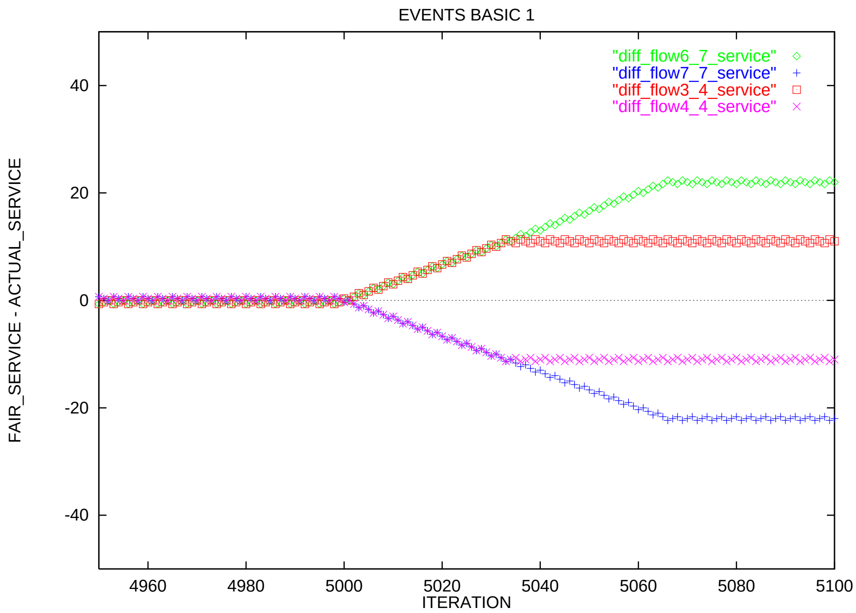


Figure 95: Time to converge, after a marginal change in the condition of flow 1, for the scheme presented in section 5.3 (*Fig. 41*), with WRR/WFQ discipline at both inputs and outputs. Crosspoint buffers size 4.

Fig. 95 and *Fig. 96* , are for the same configuration, with crosspoint buffering worth 4 cells, for each flow/crosspoints. We can see theat under $AdWRR/WFQ$, flows take much more less time(/4) to find their new fair shares and the actual unfairness, measured in bits, is reduced also at about 4 times. This is a direct consequence of the ligh occupancy of the crosspoints, under the second scheme.

Note that with this method, you can assign arbitrary small delays (probably greater than 1), to flows, like with the WRR/WFQ scheme. If the delay in the output server is bounded as much as you want – through analogous weights formulation–, you can bound the period that the flow will not be eligible in the

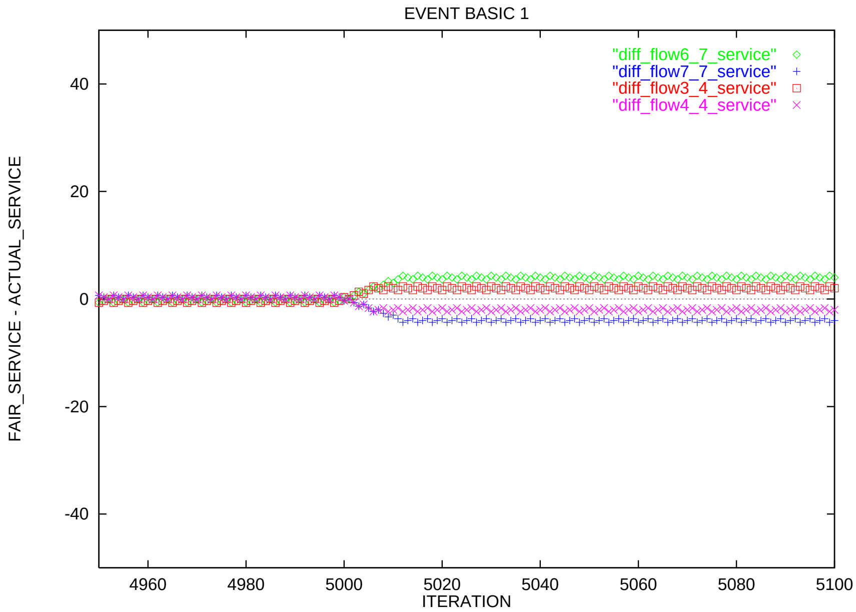


Figure 96: Time to converge, after a marginal change in the condition of flow 1, for the scheme presented in section 5.3 (*Fig. 41*), with $AdWRR/WFQ$ discipline at inputs and WRR/WFQ at the outputs. Crosspoint buffers size 4.

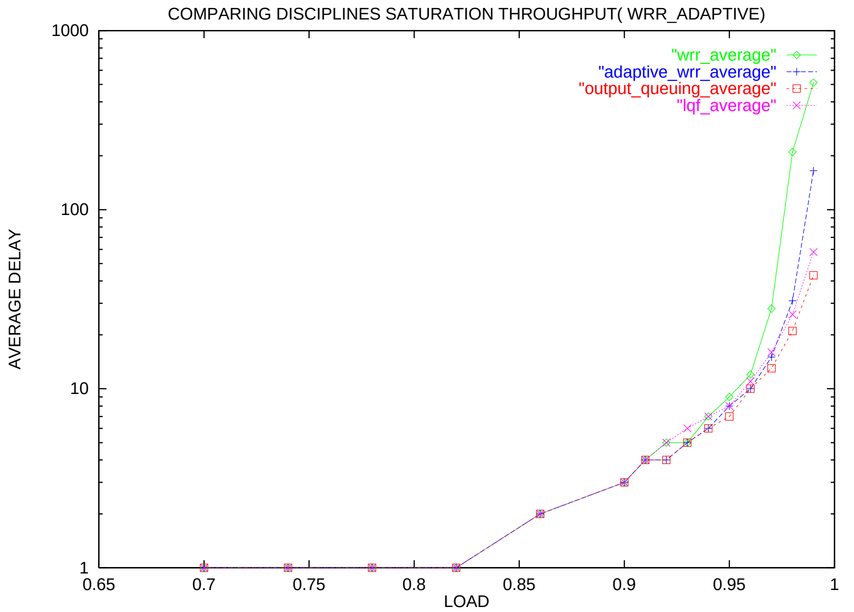


Figure 97: Average delay, for a 8x8 system, hot-spot(irregular) distribution of weights, Bernoulli id.d arrivals, independent of weights. Y-axis in logscsale.

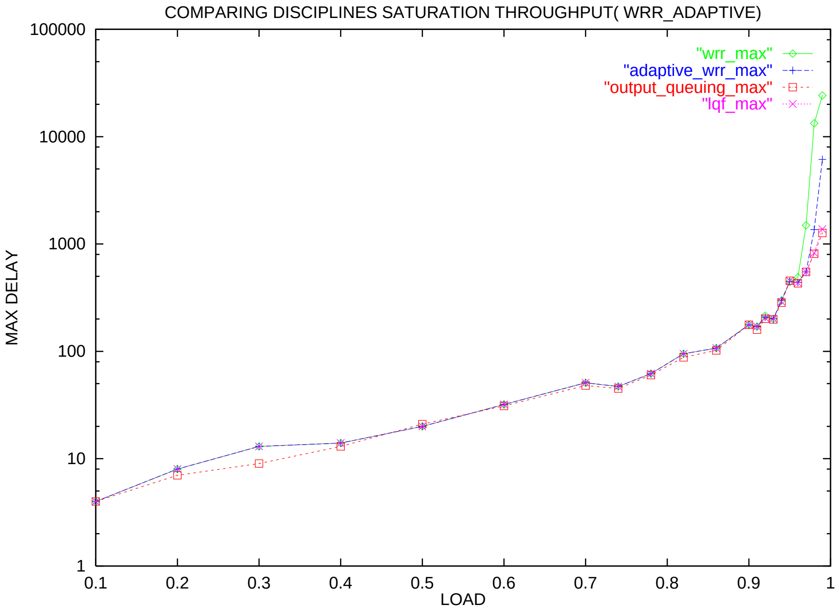


Figure 98: Max delay, for a 8x8 system, hot-spot(irregular) distribution of weights, Bernoulli id.d arrivals, independent of weights. Y-axis in logscale.

first round. Certainly all these factors, complicate the analysis of the system (you can find an analysis, of how the buffered crossbar can provide delay bounds – an other GPS desired property, retained by the buffered crossbar – in [DGPS].

This solution, seems like a good alternative in the desing spectrum, since this does not "miss" the weighted max-min fairness objective and on the same time, provides smoother allocations in the inputs $\rightarrow$ higher utilization of the output links $\rightarrow$ less average delay. Of-course, this enhancement, does not behave that good as LQF, but works *much* better than WRR/WFQ. The same behavior, is expected when bursty traffic is examined. Of-course under equal/uniformly-distributed weights and Bernoulli id.d traffic, this efficancy will not be disclosed.

11.1.3 Bad/Unfortunate (Hidden)Results / Discussions / Conclusions / Problems/open-issues – alternatives and future direction(s)

Ending, we should comment, that we have found scenaria, where the fairness objective is not reached(maximium, average discrepancy almost 10%), if we exclude in the first round the flows with not-emptied crosspoints. But this is again resolved, using a higher threshold for the exclusion process – two cells, or in general something proportional to the switch size. A final point, is that symmetrically we can define the same discipline for the outputs – there exclude in the first round the flows with small buffer – threshold – occupancy. Combining the two methods, we reach something close to maximum utilixation – local maximum since the choices are still independently considered –, still retaining analogical scheduling characteristics. From our current point, the point is, that we had almost forgotten or underestimated or misunderstood the adaptive characteristics/dimension, that any efficient administrative/resources-handling system should realize, if it rea]ly wants to be named efficient or fair. The analogical/weighted scheduling is definitelly important and necessary, but this simply/alone, cannot solve all our "problems".

So our direction/motive is to find scheduling disciplines that will approximate the model of fluids – as defined in the section 4.2.4 – as much as possible. In this view we would like to define a packet-scheduling scheme – most probably we are in the "correct direction with the internal buffers and all these –, that will be as much close to the fluid world as possible(see section 4.2.4). Although many schemes might converge to weighted max-min fairness, there can certainly be some discrimination/differentiation in-between these, and probably one can/will be proved to be optimal – something like the WF^2Q superiority, for the single resource case. We expect this hypothetical superiority to be better expressed during/in short-term intervals/decisions, and/or through higher performance under lower internal memory requirements[90].

[90] In this view we might be obliged – either simply for the sake of analysis or for insight and/or techniques – to "emulate" the fluid model, like the virtual time techniques do in the single resource case. Of-couse all these remain throughts, and we do not take/expect any grant for simply stating them here.