SWITCH ARBITRATION ALGORITHMS IN HIGH SPEED ROUTERS

Gaurav Chandra, James Hsu, Milind Kopikare, and Anamaya Sullerey

EE482B - Interconnection Networks
Department of Electrical Engineering
Stanford, CA 94305

ABSTRACT

A good switch arbitration algorithm is an important component of any high speed router. The objective of maintaining a high throughput while maintaining certain fairness standards makes the switch arbitration problem a non-trivial one. While a number of approaches have been proposed, it is still a fairly open problem with an ongoing quest for satisfactory solutions. In this paper, we attempt to gain an insight into why this is such a difficult problem. We review some key papers that have been published in this area and try to come up with a critical analysis. We also discuss issues that have not been adequately addressed. Towards the end, we also analyze the feasibility of certain schemes for implementing a switching fabric that eliminate the problem of crossbar scheduling altogether.

1. INTRODUCTION

For any interconnect router the most important architectural component is the physical switching fabric. It governs the latency, throughput, hardware complexity and cost of the router. While other possibilities do exist, in high speed routers the most common switch architecture is a crossbar. This is because of its non-blocking nature and high performance. In a crossbar switch, at any point of time, there are a number of flits/packets at each input port that may want to go to the same output port. In general, every flow at an input port has the freedom to choose an output port. *Switch arbitration is essentially the problem of mapping the set of input requests to the set of output grants* .

There are two key measures of the performance of a switch scheduler. First is *throughput*, which is a measure of how effectively we are utilizing our output channels. A scheme that leads to output ports sitting idle at times is clearly not a good scheme. The other measure of performance is *fairness*. Each incoming request should get an equal chance to send and no flow should be starved. In cases where inputs might have different priorities, outputs should be allocated accordingly. The problem then, becomes even more complicated.

Coming up with an algorithm that maximizes the performance with respect to the above mentioned measures is a very difficult task as we explain in section 2. The problem is also getting increasingly more difficult because of the increase in complexity of modern routers. The number of ports in routers have been increasing, especially in internet routers. Crossbar size increases quadratically with the number of ports. While 32×32 crossbars are common these days, people are considering 1024×1024 crossbars. With ever increasing data rates, we have a situation in which computation complexity is increasing and computation time is decreasing. While we may have good algorithms for scheduling, coming up with a solution that can be implemented quickly is of greatest importance. In the absence of such a solution, the switch arbitration stage could end up being the botteneck stage, afffecting the performance of the router.

In this paper we review some of the important work that has been done in the area of switch arbitration. In section 2, we describe the problem in more detail and elaborate upon the key issues. This is followed in section 3 by the description of some key algorithms that have tried to address the problem. In section 4 we briefly describe the approaches that try to eliminate the need of a good scheduling algorithm. We consider the feasibility and pros and cons of such solutions. Finally, in section 5, we summarize the contributions made by these approaches as well as their limitations. We also discuss the problems of scalability of the solutions and the possibility of enhancing them.

2. BACKGROUND

In this section we briefly explain the problem of switch arbitration and why it is a difficult problem. We also present the key terminologies and mention the basic classes of algorithms that attempt a solution.

The switch arbitration problem is the problem of matching the inputs to the outputs. At each input port, one could store the incoming inputs in a FIFO queue. In this case, the problem of arbitration is relatively simple. This is because only N input requests are contending for N outputs. However, we might end up with a situation where there exists a packet in the buffer whose requested output port is free, but because it is not on top of the FIFO queue, it cannot be transfered. This is referred to as head-of-line (HOL) block-

ing. To solve the HOL blocking, each input request at an input port could be stored in a different buffer. This improves the performance but also makes the problem significantly more complicated. Now at each input we have N possible flows (one for each output), and the problem is essentially a complicated bi-partite graph matching problem.

There are two classes of algorithms to solve the arbitration problem. The scheduler may attempt to maximize the number of connections at each decision instant, which is referred to as *maximum size matching*. All inputs are treated equally in this case. These algorithms are simple but have their limitations. One might want to have priorities assigned to inputs based on input queue size or QoS gaurantees. To ensure fairer arbitration, each input port could be prioritized with weights, and this total weight could be maximized. This is referred to as *maximum weight matching*.

A distinction must also be made here between a *maximum match* and a *maximal match*. A maximum match is the optimal solution for any possible input-output combination. A maximal match is one that leaves no input unmatched unless all the outputs it requests are matched. Maximal match algorithms are much simpler to compute. It has also been shown that maximum match algorithms suffer from starvation [1], [10]. For this reason most of the contemporary algorithms attempt a maximal match rather than a maximum match.

With this background, we now proceed to describe some of the important algorithms that address the problem of switch arbitration.

3. SCHEDULING ALGORITHMS

3.1. Parallel Iterative Matching

Anderson et al have addressed the need for high speed switching in router crossbars with a scheduling algorithm based on parallel iterative matching (PIM) [1]. The goal of this paper was to build a local area network that supports high performance distributed computing. The primary advantage of PIM is that it is a distributed algorithm. Rather than having a centralized scheduler that allocates the resources, each input and output has a separate grant arbiter. This enables the computation of a good match with high speed. PIM by itself is not perfect, as it can be unfair and there is no guarantee on latency or throughput. But, it is a good algorithm in the sense that it prevents starvation and converges quickly to a maximal match.

In this paper, the authors demonstrate the performance degradation caused by head-of-line blocking (HOL). It is shown that only a throughput of 58% of the link bandwidth is achieved when the destinations of incoming cells are uniformly distributed among all the outputs [1]. The PIM algorithm and other more recent algorithms (such as *i*SLIP which is similar to PIM) fix this HOL problem. This is done by using non-FIFO input buffers, where an input may transmit data to any one of its outputs for which it has a cell. The algorithm then converges to a maximal match in one to several iterations, depending on the traffic. The authors argue that apart from being time consuming, maximum matching can lead to starvation, thus justifying the use of a maximal match.

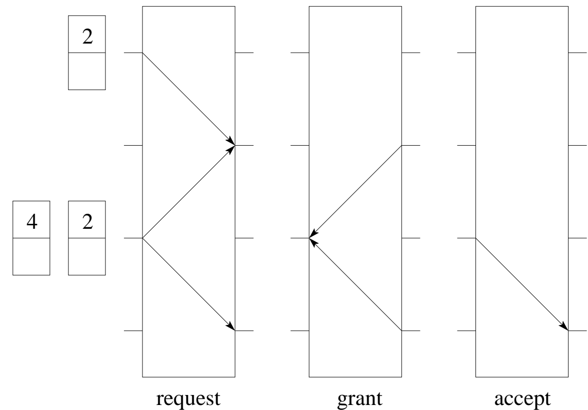

Fig. 1. Parallel Iterative Matching: One Iteration.

The PIM scheduling algorithm works as follows:

1. **Request**. Every unmatched input sends a request to every output for which it has a queued cell.

2. **Grant**. Among all received requests, an unmatched output chooses one randomly with equal probability.

3. **Accept**. Each unmatched input, then, randomly accepts one granted request with equal probability.

As is apparent in step 1 of this algorithm, HOL blocking does not exist, since each input is routed to every output that is requested in the buffer. The randomness in choosing the grant in step 2 ensures that the algorithm will converge, and that no input-output flow is starved. Each iteration of the PIM algorithm eliminates at least 3/4 of the remaining possible flows [1], and this is why it converges so quickly. Figure 1, which is reproduced from [1] shows an example of the algorithm through one iteration.

In simulations that the authors have performed, they found that after just 4 iterations, they are able to obtain a maximal matching over 98% of the time. Furthermore, it has been proved that a maximal match can be found, on average, in $O(logN)$ iterations [1]. However, they also acknowledge that in the worst case, only 50% of the maximum pairings may be matched.

Although PIM uses random arbitration in selecting which input to grant, it does not always provide fair and predictable network performance. In order for a queued cell to be scheduled, it must receive a grant from its output and its input must accept the grant. Thus, higher throughput will be given to connections that have fewer contending connections, since

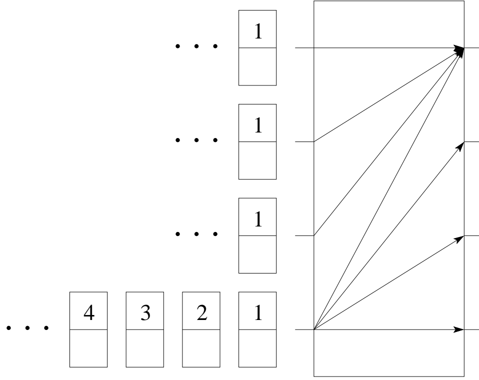

Fig. 2. Unfairness with Parallel Iterative Matching. Higher throughput is given to connections with fewer contending connections. In this case, the connection from input 4 to output 1 suffers.

the throughput is effectively split equally between the contending connections. Figure 2 shows an example of this.

Since the PIM algorithm cannot provide fairness or guaranteed throughput, the authors of this paper have made enhancements to the PIM algorithm. PIM by itself, while fast and effective at keeping links utilized, does not provide the required services [1]. By using static scheduling, they are able to provide bandwidth and latency guarantees. This is done by reserving bandwidth in terms of slots/frame. The network grants a request if it can find a path and each switch builds a schedule for transmitting cells in parallel. Statistical matching, which is a generalization of PIM and is discussed in greater detail in the article, makes more systematic use of randomness in making and accepting grants than does PIM. The algorithm mirrors PIM, except that there is no request phase, and provides fairness at the expense of greater hardware complexity.

PIM was one of the earliest algorithms to use the idea of parallel iterative matching. Since its publication, there have been newer algorithms that are based on the same idea. These try to improve the performance while reducing the complexity of required hardware. *i*SLIP, for example, is a variation of PIM with round robin arbitration to select which requests get granted.

3.2. *i*SLIP

In this section we examine *i*SLIP, another algorithm that attempts a maximal size matching. *i*SLIP has been proposed by Nick McKeown in [10]. It is similar to PIM in the sense that it tries to converge to a solution iteratively. The important difference from PIM is that it replaces the randomizers of PIM with round robin arbiters. Motivation for this is primarily from the author's claim that randomization is difficult and expensive to implement at high speeds in hardware. Another important difference in terms of performance is that *i*SLIP helps to *desynchronize* the arbiters with respect to each other, and hence achieves better results for a single iteration. The authors of [10] claimed that *i*SLIP has better performance as well as lower cost as compared to PIM. We shall examine these claims towards the end of this section.

*i*SLIP has essentially the same three steps as PIM with only slight modifications. The steps are :

1. **Request**. Each input sends a request to every output for which it has a queued cell.

2. **Grant**. If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the granted input if and only if the grant is accepted in Step 3.

3. **Accept**. If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo) N to one location beyond the accepted output.

With a round robin arbitration as mentioned above, *i*SLIP deterministically achieves a fair arbitration because of certain properties. Lowest priority is given to the most recently made connection because of the way pointers are managed. Also, no flow is starved because in the output arbiter, pointers are only updated if a successful match is made in Step 3. The significant improvement over PIM occurs, though, because of the fact that each output arbiter favors a different input and a large match is achieved in a single iteration. This happens because when a successfully matched output arbiter moves its pointer to one position beyond the input that it is connected to, it must be the $only$ output that moves its pointers to that position. In other words, the set of output pointers that move, do not clash with each other in the next cell time. In this regard, the arbiters are $desynchronized$.

To compare the performance of *i*SLIP with PIM, we reproduce here two figures from [10]. Figure 3 shows the performance for a single iteration while Figure 4 shows the performance for four iterations of these algorithms. The performance is modeled for random uniformly distributed Bernoulli traffic. Also shown in the figure are the results for maximum size matching, FIFO queuing and perfect output queuing. Perfect output queuing only has queuing caused by contention for the output links and is hence the upper bound on the performance. The figures also include results for another algorithm called iLRU, but we omit its details here because of its relative insignificance. We also reproduce the simulation results with bursty traffic and four iterations of these algorithms in Figure 5.

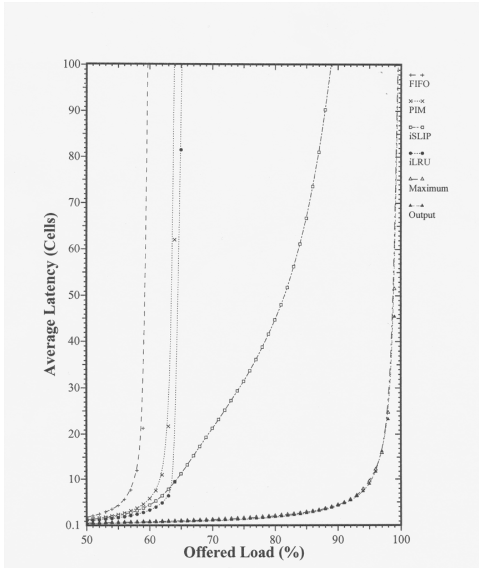

Fig. 3. *i*SLIP with one iteration and uniform traffic. iSLIP outperforms PIM as seen in this plot.

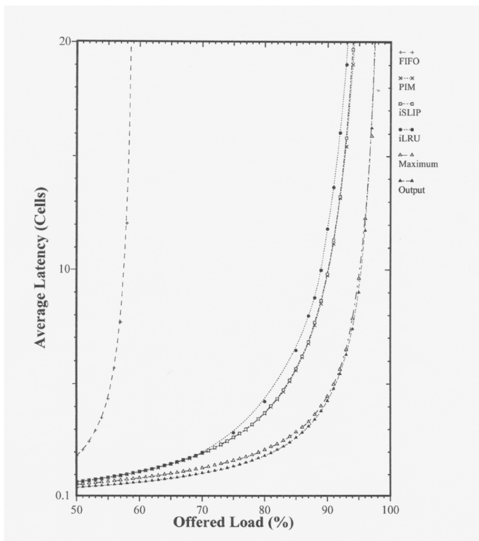

Fig. 4. *i*SLIP with four iterations and uniform traffic. The performance of PIM and iSLIP are identical.

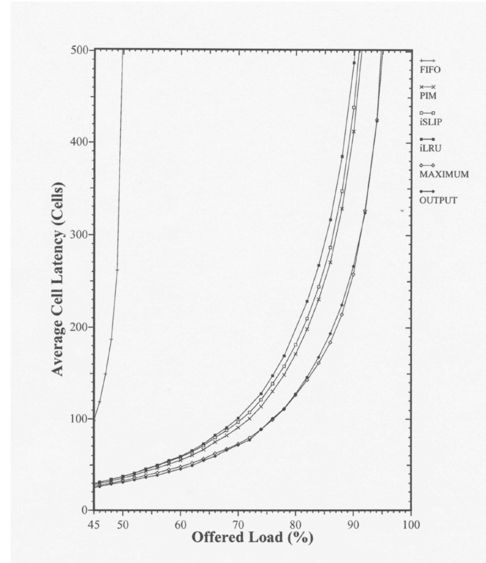

Fig. 5. *i*SLIP with four iterations and bursty traffic. Again, iSLIP and PIM perform equally well.

In Figures 3 and 4 the key thing to notice is that even though *i*SLIP is better than PIM for a single iteration, the performance is identical for four iterations. Furthermore, the performance achieved by four iterations is significantly better than one single iteration for both algorithms, hence justifying the use of four iterations rather than one. Even for bursty traffic, Figure 5 shows that performance is nearly identical. Based on these observations, *we assert that the author's claim of better performance is somewhat misleading*. In a working implementation of *i*SLIP [13] that we will discuss shortly, four iterations of *i*SLIP are used for good performance. This obviates the performance claims achieved by *desynchronization*. Another significant point is that it is not clear what the performance difference is between one and four iterations with bursty traffic, as it was not presented in [10].

The paper also makes the claim that hardware complexity of *i*SLIP is less than PIM owing to the fact that randomization is expensive. It is not exactly clear why randomization should be much more expensive than a programmable priority encoder. We think that with a Pseudo-random pattern generater randomization should not be expensive. *Thus, the only advantage of iSLIP seems to be better performance for a single iteration*.

An *i*SLIP scheduler consists of $2N$ identical arbiters, N each for grant and accept. A schematic of an *i*SLIP scheduler is presented in Figure 6. Each *i*SLIP arbiter is essentially a programmable priority encoder, as shown in Figure 7. A working implementation of *i*SLIP is done in the *Tiny Tera* router [13]. It is a 32×32 crossbar switch with each port operating at 10 Gb/s. Apart from using *i*SLIP, there are certain key features to the hardware implemen-

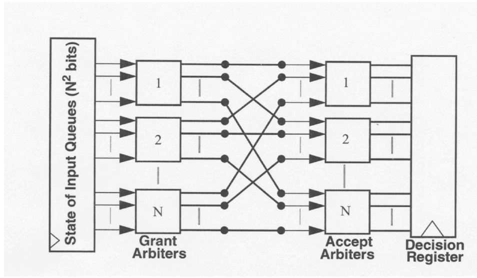

Fig. 6. Schematic of an *i*SLIP scheduler.

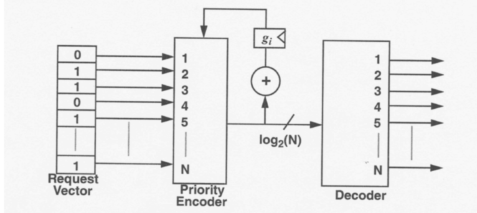

Fig. 7. Implementation details of an *i*SLIP arbiter. It is essentially a programmable priority encoder.

tation which are described in [6]. In this implementation, the grant and accept phases of different iterations have been pipelined. This is based on the observation that an input that receives at least one grant will definitely be matched and so we know which inputs did not receive the grant. Thus, the next grant stage can be started for the unlucky inputs. However, the accept phase must complete eventually to decide on the available outputs and thus allocate them in next grant stage.

Apart from this, the main contribution of [6] is really the design of fast programmable priority encoders (PPE). We omit the hardware details in the interest of brevity. This improved PPE design along with the pipeling mentioned above enables the authors to come up with a fast working implementation.

3.3. Weight Based Algorithms

So far we have looked at non-weighted matching algorithms. There's another class of algorithms called weighted-matching algorithms, that have better performance but are more difficult to implement. Weighted algorithms consider request weights when making scheduling decisions. The switching community has experimented with many different weights such as the queue size of a virtual output queue (VOQ) or the waiting time of the head-of-line packet in each VOQ [14]

Non-weight algorithms such as Round-Robin Arbitration work just as well as weight based ones when the input traffic pattern is uniform as all inputs are equally prefered. But if the traffic is non-uniform, there are some inputs which might be more loaded than others. If the algorithm continues to treat all queues equally, some input buffers will eventually overflow while others will be relatively empty. If the algorithm takes into consideration the weight of a queue in terms of the queue size, and gives preference to the heavier queues, all queues get served in the right proportion with the net result being that each queue has approximately the same average queue weight. This is the more desired situation.

Consider the bar graph in Figure 8 that shows the queue occupancies of the switch in Figure 8. If the scheduler chooses a max-size non-weight based match of size four, the VOQs (1,1), (2,2), (3,3), (4,4) will become empty making the graph more sparse, and reducing the size of subsequent matches. If, instead, the scheduler maximizes the weight of the match, the "tall" VOQ s are pushed downwards, while the "shorter" VOQs are allowed to grow, Over time this tends to balance the occupancies of the (active) VOQs, making large matches possible.

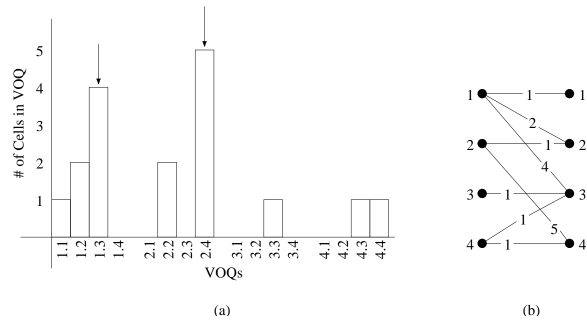

Fig. 8. (a) A bar graph showing VOQ occupancies of a 4×4 switch; (b) The corresponding request graph. The arrows indicate that the corressponding VOQs are pushed downwards (services) if a maxweight match is chosen.

The above explanation can also be extended to the case where one has priorities assigned to different kinds of traffic. In such a scenario we would again need to use weights to differentiate between the service given to different kinds of traffic in a dynamic manner [7]. Another issue that crops up because of non-weighted algorithms is that of starvation. In [11], the authors discuss how a non-weight based algorithm can result in permanent starvation of a switch. This problem is overcome by weight based algorithms because as a packet is retained in a queue (starved of bandwidth), its weight increases proportionally and hence eventually the weight will be large enough to ensure that the packet is flushed out of the queue.

Finally there is also the issue of stability of a switch. A switch is said to be stable if its queue size can be bounded. Again in [11], the authors have proven that a non-weighted algorithm would render the switch unstable for certain non-uniform traffic patterns [2]. On the other hand, the authors of [2] found that for any kind of admissible traffic pattern, a

weight based algorithm such as a maximum weight matching ensures that the switch is stable and that no input queue increases in size infinitely.

There are a few weight based matching algorithms explained in literature [11], [5], [14] . We will discuss one of them in a little depth, namely iLQF. The rest of the algorithms have a similar design and would be explained briefly.

iLQF is an implementation of a weighted algorithm called Longest Queue First Algorithm [9]. LQF uses a max-weight algorithm to give preference to more heavily occupied virtual output queues. Each request weight is set to the corresponding queue occupancy. Unfortunately, LQF is impractical [9]. Using a max-weight algorithm makes LQF too complex to implement in hardware with the running time complexity of $O(N^3 log(N))$. In [9], an iterative version of LQF, called iLQF, was introduced to solve this problem. Each iteration consists of three steps:

1. **Request**. Every unmatched input makes a request to every output for which it has a cell destined. Every request carries a weight equaling the associated queue length for iLQF.

2. **Grant**. Each output grants the largest request. Ties are broken randomly.

3. **Accept**. Each input accepts a grant to the largest request. Ties are broken randomly.

It is similar to the implementation of iSLIP and PIM, however steps 2 and 3 consist of weight comparisons rather than simple round robin arbitration.

We now discuss the hardware and computation complexity of iLPF. For all iterative algorithms, the running time depends on two factors: the number of iterations and the time per iteration. For iLQF, in the worst case, it can take up to N iterations to find an optimal match for an $N \times N$ switch [9]. But in practice, it has been found that only $log(N)$ iterations are needed to achieve close to optimal performance.

The time per iteration, however, depends on the implementation. For the implementation in Figure 9, the iteration time can be calculated simply by adding up the functional blocks along the loop shown by dotted lines. Among all components in the loop, the grant arbiters and the accept arbiters dominate the iteration time. Both arbiters, each a modified N-input magnitude comparator, are slow due to the large number of input values that they need to compare and the relatively high complexity of the basic building block, a two-input comparator. Thus if b bits are used to describe the weight of each input queue request, the best running time of each arbiter is $O(log(b) * log(N))$ as compared to $O(log(N))$ as in the case of iSLIP. These multi-bit integer comparators that are the primary source of the complexity problem.

Other examples of weight based schemes are the iOCF [14], MUCS [5], and iLPF [15].

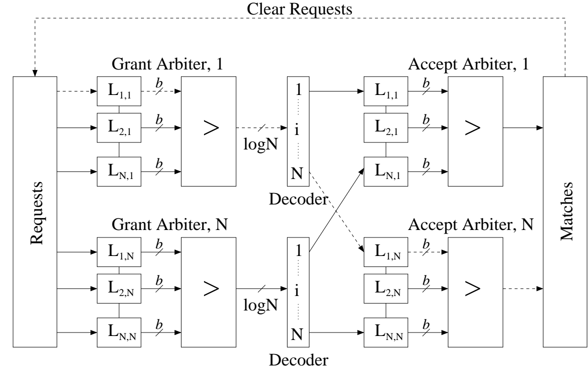

Fig. 9. An iteration starts from the request block containing registers that hold request indicators and is completed by the feedback from the matches block. Between the requests block and the matches block, the iteration progresses through the set of registers holding request weights (in this case, queue occupancies $L_{i,j}$), the grant arbiters, the decoder and the accept arbiters. Upon completion, the match result is fed back to clear the requests of matched inputs and all requests to matched outputs. The critical path in the iteration is marked by the dotted lines.

iOCF is similar to iLQF except that it uses the waiting time of a packet in the input queue as the weight rather than the queue size. This has the added advantage that no queue gets starved of bandwidth which could potentially be the case in iLQF [15].

In MUCS (Matrix Unit Cell Scheduler), the authors have come up with a novel analog circuit that implements a weight based algorithm. Here the weights used are:

$$w_{ij} = \begin{cases} \frac{a_{ij}}{wr_i} + \frac{a_{ij}}{wc_j}, & \text{if } a_{ij} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

where a_{ij} is the queue occupancy of VOQ (i,j).

The heaviest element in the matrix corresponds to a buffered packet that has the least contending candidates in the same row and the same column.

iLPF is another interesting idea where the authors have defined their weights in such a way that the maximum weight match also gives a maximum size match. However this scheme is prone to all the problems that a maximum size based matching algorithm would have, specifically the problem of starvation of a queue.

As compared to non-weighted algorithms discussed earlier, weighted ones perform better for a broad range of traffic patterns. Simulation results are shown that compare the average paket delay of two weighted algorithms with two non-weighted ones for non-uniform traffic.

3.4. Limitations of Discussed Schemes

In this subsection, we discuss the shortcomings of the solutions described thus far. As we saw, size based matchings

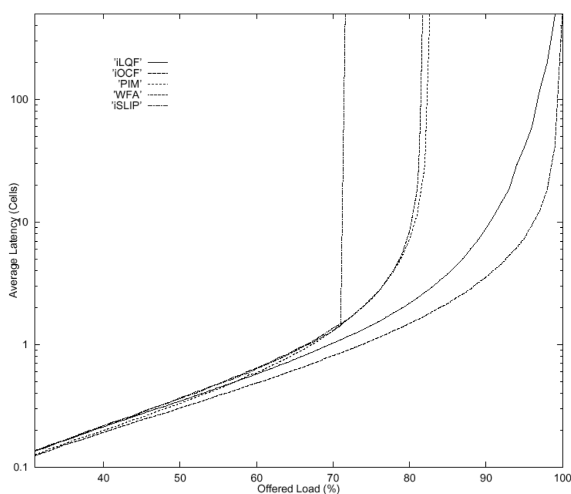

Fig. 10. Performance comparison of iLQF and iOCF to PIM, WFA, and *i*SLIP. The graph shows simulation results of the average cell latencies as a function of the offered load of 3×3 switches under non-uniform traffic described in previous sections. Arrivals at each input are Bernoulli i.i.d. The number of iterations is three for iLQF, iOCF, PIM and *i*SLIP.

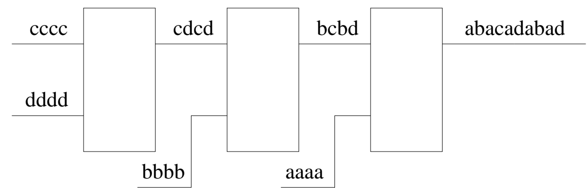

Fig. 11. Unfairness with Point-to-Point Networks. Packets from 'c' and 'd' only get 1/8th of the bandwidth each.

do not result in fair allocations and they are not well suited to implement QoS guarantees. A number of weight based algorithms are proposed to address this issue. A disadvantage of even weight based matchings is that we end up with a maximal weight match rather than a maximum weight match. It is not yet known if maximal weight matching algorithms are stable for all possible traffic patterns. This is an interesting problem being considered by the switching community.

Another problem that none of the schemes discussed so far resolve is that of maintaining global fairness. Consider Figure 11 consisting of a ring of routers in an interconnection network. Here at each router node, a new stream of packets gets added. If it is a weight based algorithm that looks at allocating equal resources to all queues entering it locally, the packets coming from the left-most end will suffer the most as each router along the path will keep cutting down the bandwidth available to these packets by half. A possible solution to this problem [4] is to have a time stamp on each packet, where each router along the path gives highest priority to the oldest packets.

A limitation of weight based matching algorithms is that, although they are far superior than plain non-weight based algorithms, they are extremely difficult to implement. A lot of research is going on in trying to simplify the implementation of such algorithms. One interesting approach could be that since we know that the bits used to represent weights add complexity to the matching precess, we could use weights that use a much smaller number of bits. For example, instead of using the queue size as the weight, we could use $log(queuesize)$ as the weight. This would immediately reduce the running time of the arbitration block from $log(b)log(N)$ to $log(log(b))log(N)$.

Also, it should be noted that all the papers we have discussed assume that the data are transmitted in fixed length cells. They assume this because the performance guarantees are easier to provide and the non-FIFO buffer management is simpler in this situation. The authors leave the reponsibility of dividing up packets into cells to the host controller when dealing with variable-length packets. This assumption actually leads to bandwidth loss which is not discussed in these papers. It is addressed in the paper by Kar, Lakshman, Stiliadas, and Tassiulas [8]. Incoming packet streams are made up of variable sized packets in IP networks. Crossbar switches fragment packets into envelopes which get transfered over the fabric. If the packet size is not an integral multiple of the envelope size, then there may be a significant loss in the fabric bandwidth. A scheme to avoid loss of fabric bandwidth as proposed by [8] is illustrated in Figure 12. In this scheme, partially filled envelopes wait at the shaper for more packets to arrive. They are not transfered across the fabric until the envelope has been completely filled.

4. AVOIDING THE SCHEDULING PROBLEM

To do justice to the topic, it is essential to describe certain solutions that have taken a totally different approach. Here we proceed to describe three such solutions. One of these solutions deminishes the need of a good crossbar scheduling algorithm. The other two solutions do not use crossbars, thus completely eliminating the problem.

4.1. Overprovisioning the crossbar

Overprovisioning the crossbar is a frequently used solution in multi-computer interconnection networks [4]. In such networks, latency is of utmost importance. Most of the algorithms that we have desribed are iterative solutions that take multiple iterations to give satisfactory results. In a multi-computer network, one wants to come up with a good match as quickly as possible, preferably in one iteration. Overprovisioning the crossbar helps alleviate the inefficiencies of a matching algorithm and comes up with a good match.

To explain how overprovisioning works, consider a very

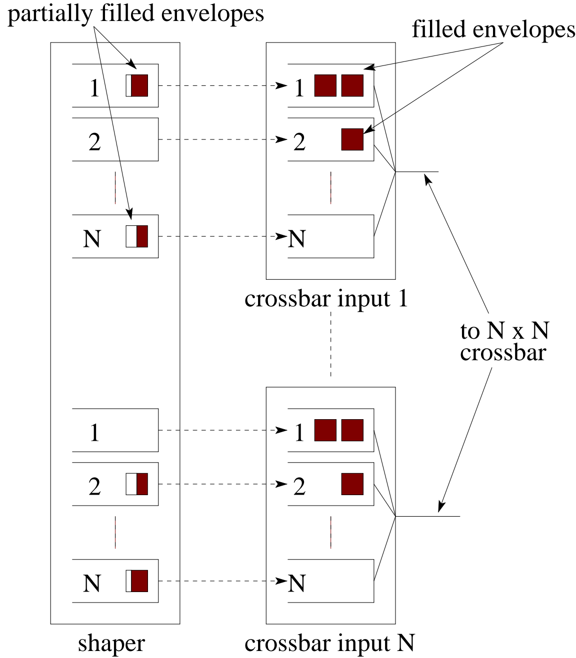

Fig. 12. Scheme for avoiding fabric bandwidth in input queued switches for variable packet sizes.

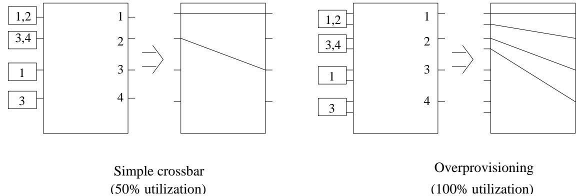

Fig. 13. Achieving full throughput in a single iteration with an overprovisioned crossbar.

naive example of a 4×4 crossbar, where contention is as shown in 13. Suppose in the first iteration all four outputs are granted between two input channels. After one iteration only half the capacity is utilized. However, with an overprovisioned 8×4 crossbar, full capacity would be utilized after just one iteration. In the limit, if we make a 16×4 crossbar, we achieve perfect output queuing. In general for a big crossbar, some happy median exists.

4.2. Torus routing

This is the solution implemented in the Avici Terabit Switch [16]. Instead of using a crossbar, it uses a torus interconnection network as the basic switching fabric. With a torus network, the problem of scheduling is completely eliminated and the problem becomes distributed. Sharing of physical channels also makes these networks more economical, and thus the solution is more scalable. The key concern in a torus network is, however, the fact that it is not a non-blocking network. The proposed solution is to use virtual channel flow control [3], where node is given its own virtual network.

Such a network would probably have higher latency as compared to a crossbar switch. In a packet switching fabric for internet applications, this is not a major concern. However, due to congestion intrinsic to the router, there will also be variation of delays. This would be unacceptable in a packet switch fabric. We could not find any studies that come up with a bounds on delay variation in such a scheme.

Another important point is that a packet switch fabric is not expected to mess up the order of packets as per the RFC specifications for routers. In the presence of congestion in a torus, this could potentially happen. We are not aware of how this is addressed in Avici, but we think this is a key issue to be resolved.

4.3. Shared memory architecture

Instead of having a dedicated crossbar network, one could have a switch fabric implemented as a shared memory system. This is a solution that's used in the Juniper M40 router [17]. Shared memory architecture is totally non-blocking as compared to a crossbar switch architecture, which requires multiple input queues to prevent head-of-line blocking. In the Juniper implementation, there is a distributed buffer manager that manages the shared memory and does the task of allocation.

However, the primary disadvantage of a shared memory system is its complexity as well as scalability. Thus, we do not think that it is going to be a solution that could possibly replace a crossbar switch. We primarily felt the need to describe the solution here since it is implemented as part of a contemporary high-end router.

5. SUMMARY

In this paper, we reviewed some of the key issues involved in crossbar scheduling. To summarize, the scheduling algorithm should maximize throughput while maintaining fairness of arbitration. We saw that although maximum matching is the choice achieving highest throughput, its computational complexity is very high. It also has the problem of starvation. Maximal matching is a better solution in terms of hardware complexity.

We discussed two maximal size matching algorithms, PIM and *i*SLIP. Both thse approaches are iterative attempts to achieve a maximal size match. Though these solutions are relatively simple to implement in hardware, they suffer from the problems inherent to maximal size matching. It is difficult to ensure fairness in these algorithms. The problem is aggravated if one desires to put in QoS guarantees.

To achieve fairness and to incorporate QoS guarantees, several schemes have been proposed that attempt a maximal weight matching rather than maximal size matching. We have described some of these approaches. One of the popular algorithms, *i*LQF, is discussed in detail. We find that

though providing better performance, these algorithms are more complex. Satisfactory hardware solutions that could operate at current data rates have not been proposed. Another issue that all these algorithms fail to address is the ability to provide global fairness rather than local fairness.

We have also discussed some other schemes that obviate the need of a good switching algorithm. Overprovisioning a crossbar switch helps to achieve better match with less iterations, but it is not a scalable solution. A shared memory architecture also suffers from the scalability problem. Torus routing, as proposed by Avici, is scalable but it is not clear how delay guarantees and packet ordering is maintained inside the router.

In summary, the problem of coming up with a good, fast arbitration scheme is still a very open problem. In our opinion, possible solutions could be clever hardware implementations that speed up the weight based matching algorithms. One could also try pipelining the algorithms. This would introduce dependencies that would undermine the quality of the solution. But it might still be better than maximal size matching algorithms. Another possibility is to use a completely different approach that does away with crossbars. We have discussed some of these solutions but a lot needs to be done if we want to get the same performance as offered by a crossbar switch.

6. REFERENCES

[1] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker, High Speed Switch Scheduling for Local Area Networks, In: *ACM Transactions on Computer Systems*, November 1993.

[2] J.G. Dai and B. Prabhakar, The Throughput of Data Switches With and Without Speedup, In: *Proceedings of the IEEE INFOCOM*, Tel Aviv, Israel, 2000.

[3] W. J. Dally, Virtual-channel flow control *IEEE Trans. on Parallel and Distributed Systems*, Vol 3, No 2, pp. 194-205, March 1992.

[4] W. J. Dally, *EE482B Class Notes*, Stanford University, 2001.

[5] H. Duran, J. Lockwood, and S. M. Kang, Matrix Unit Cell Scheduler (MUCS) for Input-Buffered ATM Switches, *IEEE Communication Letters*, Vol 2, No 1, January 1998.

[6] P. Gupta and N. McKeown, Design and Implementation of a Fast Crossbar Scheduler, In:*Hot Interconnects, 1998*, August 1998.

[7] A. Kam and K. Y. Siu, Linear-Complexity Algorithms for QoS SUpport in Input-Queued Switches with No Speedup, *IEEE Journal on Selected Areas in Communication*, Vol 17, No 6, June 1999.

[8] K. Kar, T. V. Lakshman, D. Stiliadis, and L. Tassiulas, Scheduling of Variable Size Packets in Input Queued Switches with Bandwidth and Delay Guarantees, *Masters' Thesis*, University of Maryland at College Park, December 1999.

[9] N. McKeown, Scheduling Algorithms for Input-Queued Cell Switches, *PhD Thesis*, University of California at Berkeley, May 1995.

[10] N. McKeown, The *i*SLIP Scheduling Algorithm for Input-Queued Switches, *IEEE Transactions on Networking*, Vol 7, pp. 188-201, April 1999.

[11] N. McKeown, V. Anantharam, and J. Walrand, Achieving 100% Throughput in an Input-Queued Switch In: *Proceedings of IEEE Infocom '96*, San Francisco, Vol 1, pp. 296-302, March 1996.

[12] N. McKeown and T. E. Anderson, A Quantitative Comparison of Iterative Scheduling Algorithms for Input-Queued Switches, *Computer Networks & ISDN Systems*, Vol 30, No 24, pp. 2309-2326, 1998.

[13] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, Tiny Tera: a packet switch core, *IEEE Micro*, pp. 26-33, January 1997.

[14] A. Mekkittikul and N. McKeown, A Starvation-free Algorithm for Achieving 100% Throughput in an Input-Queued Switch, In: *ICCCN '96*, pp. 226-231, October 1996.

[15] A. Mekkittikul and N. McKeown, A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queued Switches, In: *IEEE Infocom 98*, San Francisco, Vol 2, pp. 792-799, April 1998.

[16] courtesy of L. S. Peh, The Avici Terabit Switch/Router *Presentation slides*, August 1998.

[17] Juniper Corporation: Internet Backbone Routers and Evolving Internet Design http://www.juniper.net/techcenter/techpapers/200002.html