A Multicast FCFS Output Queued Switch without Speedup

Maurizio A. Bonuccelli and Alessandro Urpi

Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56100 Pisa, Italy. {bonucce,urpi}@di.unipi.it

Abstract. In this paper we propose an architecture for an output queued switch based on the mesh of trees topology. After establishing the equivalence of our proposal with the output queued model, we analyze its features, showing that it merges positive features of the input queued switches (specially their implementability) with all the characteristics typical of output queued ones. Moreover, such an architecture is able to easily and efficiently manage multicast traffic, which is becoming extremely important in networks with traditional communication services integrated in.

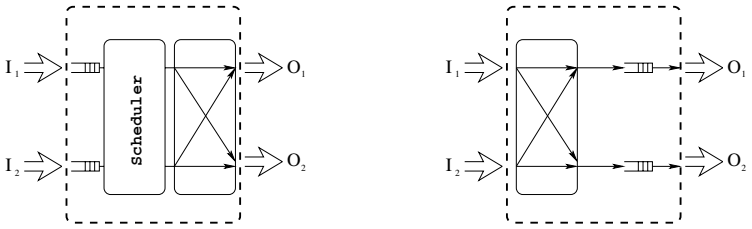
1 Introduction

Internet is evolving to an integrated services network with a large number of users that exchange huge amounts of data, making the efficiency of the switching phase increasingly critical ([1,2,3,4]). This is even more evident since large parts of Internet are circuit switched (SONET[1], just to cite one name), and since link speed is rapidly increasing (for example, 40 Gb/s at OC768c or 160 Gb/s at OC3072), making routers/switches a serious bottleneck. At a suitable level of abstraction, a switch is a box connecting n source inputs that want to exchange messages with m destination outputs. The system is synchronous, and the time is slotted. Without loss of generality, we can think of messages as fixed size cells that arrive at the system at the beginning of each slot, and are processed during the time interval. Since we assume that message destinations are independently chosen by each input without rules, it can happen that more inputs want to communicate with the same output at the same time, causing a potential collision. Such an event should be avoided, because it results in the loss of all the cells involved in it, and in the retransmission of all of them from the originating source. Competing cells need to be stored in a memory, and to be serialized in some way, in order to keep busy outputs with queued cells for and to avoid collisions.

There has been a deep investigation in buffered switches during the last years, that leaded to fundamental results. One of the first proposed solutions ([5,6]) was to put a shared memory between inputs and outputs where to store incoming cells and to forward a suitably chosen subset of them. While such an architecture

[1] http://www.sonet.com

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 1057–1068, 2002.



(a) Input queued (b) Output queued

Fig. 1. Different switch architectures.

is quite simple and practical for systems operating at less than 20 Gb/s, it has many problems, the most penalizing is perhaps the memory access time ($n + m$ accesses should be granted at every cycle).

A natural step to move then was to introduce a queuing system, that led to input queued (Fig. 1(a)) and output queued (Fig. 1(b)) switches. The former is the implementation of the very simple idea that every cell, at the arrival to the switch, should be immediately buffered (with a queue for every input), and then a scheduler will choose in every cycle a set of non conflicting cells (namely, cells bound for different outputs) to forward through a nonblocking interconnection network, for example a crossbar. Easy to implement, the architecture was shown to suffer of limited throughput if a FIFO strategy is used in the queues: conflicting cells in the head of the queues may block other cells that would be free to pass through the switch, causing a performance loss known as *head of line* (HOL) blocking, limiting the throughput of the system to $\sim 58.6\%$ assuming i.i.d. arrivals ([7]).

Moving the queues at the output ports results in efficient switches that don't block cells if their destination is idling, able for this reason to provide quality of service ([1,8]). This is not a solution, since such an architecture is clearly equivalent to the shared memory one, and its problem is again scalability: each queue must be able to serve up to n requests per time slot. This introduces the need for a speedup of the switch of a factor $n + 1$, limiting its implementability to scenarios with few input ports and quite slow links. In order to achieve scalability without performance problems, virtual output queued switches were proposed ([9,2]). Such an architecture avoids HOL blocking by having in each input a different queue for each output. It is clear that the scheduling phase is now critical: a set of cells must be selected for transmission at every time slot to maximize performances. It was shown ([10,11,12,13]) that there exist scheduling algorithms able to exploit a throughput of the 100% and also to avoid starvation of cells ([14]). However, such algorithms have several drawbacks, that can be so classified:

Complexity: an optimal scheduling can be found solving a matching problem on bipartite graphs ([15]), or finding a decomposition of stochastic matrices (see [16] for the switching case). The weak point of these approaches is their complexity; the best known matching algorithm runs in $O(N^2 \log_2(N))$ time in the worst case ([17]), while the second method has been proved useful to implement, at most, 4×4 switches ([18]).

Throughput: with approximate algorithms it is possible to overcome the complexity problem ([2,15,19,13]). Behind this approach there are good simulation results ([15,20]), and a proof that, if traffic reaches a steady state (i.e. there is always a cell that must be sent to every output), the behavior of these algorithms is optimal ([13]). But with *bursty traffic* ([21]) such a stability is never reached ([20]).

Performance guarantees: despite some results on the bounds in queues average sizes and on average delays in input queued switches have been recently found ([22]), it is not yet clear how to offer quality of service in such a class of switches. This justifies the research on output queued like architectures, in order to obtain guarantees on the offered service.

Combined input-output queued switches are another interesting architecture proposed as a trade-off between input and output queuing: there are queues both in the inputs and in the outputs, and a speedup of k is used, in the sense that it is possible to transfer k cells from every queue in the inputs to the desired queue in the outputs at every time slot. In [23,24] it was proved that a speedup of 2 is enough and necessary to emulate an output queued switch with a queuing policy that varies in a well known class. Unfortunately a locally optimal scheduling does not guarantee network optimization: in [25] it is shown that input queued switches with high performance scheduling algorithms, efficient in isolation, cause unbounded delay of cells when put in a network. In order to avoid this problem, and to offer quality of service (QoS), it is then important to have practical solutions resembling output queued switches. Parallel architectures are an encouraging alternative ([26,27,28,29,30]).

In this work we take a completely different approach, and propose an output queued switch obtained by parallelizing the "classical" architecture, having very interesting features like high compositional power (i.e. it is easy to create a greater switch by using smaller ones), no speedup required (in a sense that will be cleared later), real implementability and efficient multicast management.

The paper is organized as follows. Section 2 introduces the notation we will use throughout the paper. Section 3 outlines the idea at the very base of our proposal, relating it with a well known architecture. In Sect. 4 the topology of the mesh of trees is presented, together with some of its most important features. In Sect. 5 we present a new architecture for a switch, proving that it is equivalent to an output queued one. Finally, we conclude in Sect. 6 summarizing our work and proposing future directions.

2 Definitions

The following concepts and terms are very important through the paper:

Number of ports: without loss of generality, the switches are supposed to have n inputs and n outputs,

Names: I_i is the i^{th} input, O_j is the j^{th} output; Q_i is the queue at the i^{th} input (output) in an input (output) queued switch, and L_i is its size,

Acronyms: IQ means Input Queued, OQ is Output Queued, while VOQ is Virtual Output Queued and $CIOQ$ is Combined Input-Output Queued,

Mimicking: as defined in [24], a switch S mimics another switch S' if, for any arrival pattern and independently of the switch size, the outputs are exactly the same. In [30] the definition is extended by considering a possible queuing delay for the cells, i.e. the outputs of the two switches are the same but with a temporal shift caused by queuing. So, an architecture X mimics an architecture Y with a delay of $f(n)$ if, under the same arrival process, the outputs of X at time $t + f(n)$ are the same of Y at time t.

3 A First (Impractical) Step

We begin by presenting a new point of view of an OQ switch. Later in this section the same intuition will be presented from a different perspective.
Let us assume we have a $n \times n^2$ crossbar[2] and that each cell is associated with an integer representing its arrival time (a time stamp). Then, we can think of splitting the queues in the output ports in n different queues, one for each input, like in Fig. 2. Such an architecture can be thought of as the complement of a VOQ switch, and it should not be hard noting that it can perfectly emulate an OQ switch. In fact, assuming a FIFO strategy, the division in n queues is equivalent to the distribution of the cells in queues, sorted by sender. Q_i in an OQ switch with a speedup of n, would contain all the cells sent to output i, sorted by the time of arrival to the system, with simultaneous arrivals serialized with a specific rule (for example smaller index of sender first, or randomly). In this way, in the n queues at the i^{th} output, there is a double sorting: by arrival time and by sender. Then, if the S_i element chooses the oldest cell from all the queues, breaking ties with the same rule that would have been used in the target OQ switch, we perfectly emulate it.
The proposed architecture apparently does not require any speedup to achieve an OQ switch behavior emulation. Actually there is a logarithmic factor to be accounted for. In fact, assuming to be able to compare n time stamps in only one cycle is unrealistic (specially for very large n, that is our final target). The best thing we can do is to use a comparing tree, with $\log_2(n)$ stages of parallel comparisons. Such a logarithmic factor must be paid off in terms of scheduling iterations, in the worst case, also in almost every proposed VOQ switch ([2] for

[2] Actually, a full crossbar is not necessary. A structure containing n selectors or a sorting network would be enough, but it is easier to imagine a crossbar

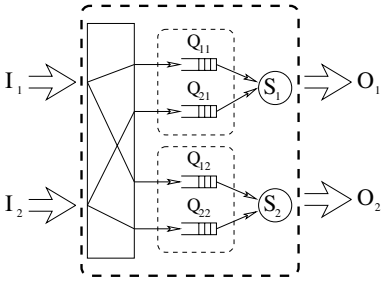


Fig. 2. Output queued switch?

PIM, [15] for iSLIP, [31] for iLPF, just to cite some very popular proposals). In *IQ* and *VOQ* switches, it is usually assumed that these computational steps can be done during a time slot (thus limiting the power of scheduling algorithms). In our proposal, we can avoid this delay with a very simple pipelining technique. In fact, it is not necessary to wait for the entire comparison to be over before starting a new one but, because of the tree structure of the comparison part, each element (leaf or internal node) can compare two cells, and forward the oldest to its parent (the root must send the oldest outside the switch). Thus, as soon as an element finishes its work, it can start again for another round, waiting only if its successor in the tree (its parent) is not ready to receive. Thus it is possible to perform $\log_2(t)$ comparisons (one for level) in parallel. The latency of the cell in the switch is proportional to the logarithm of the switch size but the throughput of the system will not suffer from this.

The introduction of a pipelined part in a switch is not a new concept: in [32] the scheduler for a *VOQ* switch is improved with this technique, but the whole system is very different (and more complicated) from the one presented here. We make another step in our description, in order to have a system easier to implement. It is possible to come to the same idea also by starting from well known results. In [33], the architecture of an *OQ* switch called *knockout* was presented. Each input is connected to one bus, and each output is connected to every bus. We can think of the output modules as single queues, and still have the mentioned speedup problem. We can also think of increasing the number of queues, in order to avoid speedups by increasing the cell loss probability (namely, the probability of dropping conflicting cells). Of course, by putting one queue for every input in each output module , there is no cell loss (at this stage), and the architecture is very similar to the one we sketched. It is also possible to put less queues (say L), preceded by a statistical multiplexer that just chooses L cells, if there are more, and discarding the others. A buffering scheme that uses several (L) FIFO queues as just one queue with L inputs and one output in a knockout switch makes it acting like an *OQ* switch without requiring any speedup. It was shown that $L = 8$ queues are sufficient to reduce the loss probability to 10^{-6} for an arbitrarily large switch size n ([33]). However we are interested in avoiding cell loss (and then in using n queues without multiplexer). The architecture,

conceptually interesting, has many problems, like number of busses too high when the number of inputs grows and a number of crossing points not feasible when there are many outputs. Moreover, there is a spatial speedup to pay: implementing N adjacent memories is not so different from implementing one with a (temporal) speedup of T. In Sect. 5 we will see why the architecture proposed in this paper can be more practical, while potentially having the same problems.

4 Mesh of Trees

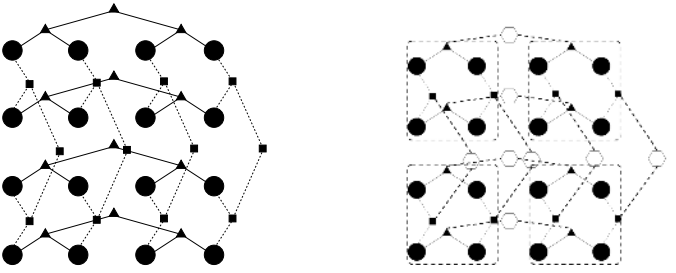
We present here a well known topology called mesh of trees, recalling only what is helpful to our aim. For more details the reader can refer to [34].
An $N \times N$ two-dimensional mesh of trees is a structure obtained from a $N \times N$ mesh (or two-dimensional array) by adding nodes in order to form a complete binary tree for every row and every column, with the nodes of the mesh as shared leaves (see Fig. 3). There is also an interesting recursive definition of the topology: given four $\frac{N}{2} \times \frac{N}{2}$ meshes of trees it is possible to combine them in a $N \times N$ one just by using the four smaller meshes as elements of a 2×2 mesh, and combining the $4N$ roots pairwise adding $2N$ new roots (for a practical example see Fig. 3(b), where the nodes to be added are represented by hexagons).
The total number of nodes in a $N \times N$ mesh of trees is $4N^2 - 2N$. Communications between root nodes of column trees and root nodes of row trees are interesting in several ways. First of all they have a fixed length of $2\log_2(N)$ hops. Moreover, if we label each destination node with the binary representation of a number between 0 and $N - 1$ (of course a different label for each different node), the routing of a message through the mesh of trees is very simple (i.e. the topology has a *self-routing* property). For example, nodes at i^{th} level will forward the message to their right son if the i^{th} digit of the label is 0, to their left son otherwise. The leaves work as interchange points, and they just have to forward the message from the column tree to the row tree they belong to. The communication is then logically divided in two steps:

1. a *selection* phase, in which the message is directed to the right row,
2. a *gathering* phase, in which the message is conveyed to the desired root.

In terms of hardware complexity it is clear that each node, if only communication is needed, is very simple. Implementability of meshes of trees in single chips was widely studied (e.g. see [35]). In the next section, we will see how to combine meshes of trees and the switch architecture we proposed in Sect. 3.

5 The New Architecture

It is possible to produce a $n \times n$ OQ switch equivalent to the one we presented in Sect. 3, by means of a $n \times n$ mesh of trees. Assume to associate each input to a column tree root, and each output to a row tree root. In this way, a cell from



(a) 4×4 mesh of trees

(b) Composition of smaller meshes

Fig. 3. Two views of a 4×4 mesh of trees

input i to output j can be seen as a communication between roots, exactly like those presented in Sect. 4. Thus, the selection stage is exactly equivalent to the crossbar-like element in Fig. 2, while the gathering stage, choosing the oldest cell in case of contention, is just a comparing tree. We can think to put the queues in the leaves: in this way they would become very simple elements encapsulating a queue. It is easy to see that the two architectures are equivalent. Later in this section, a formal proof of the above mentioned mimicking will be presented.
The logarithmic factor in the selection stage can be amortized in the same way we did in the comparing phase: up to $\log_2(t)$ communications can be present in parallel in the column trees, for a total of $2\log_2(t)$ communications at most that can be done in parallel. In the remainder of this section, we shall assume infinite size queues, and we use the following additional notation:

Memories: in the mesh of trees, for each output, the memory is divided into *queuing memory* (in the leaves) and *tree memory* (up to one cell can be stored in each internal node in the row tree while waiting to exit from the switch).

Symbols: Extending (in a natural way) the names given in Section 2, Q_{ij} is the queue from input i to output j in the mesh of tree, and L_{ij} is its length.

In order to establish that the mesh of trees switch mimics an OQ switch (with a delay, as we will see), it is useful to introduce an intermediate architecture that will be used as a paragon. In Fig. 4, it is shown a queued architecture for a single output (referred in the remainder of this section as DFIFO[3]) composed by n queues, from which the K element chooses the oldest to forward (breaking ties in the usual way), and $\log_2(n)-1$ elements that just forward from one end to the other (actually the architecture is just the one shown in Fig. 2, with $\log_2(n)-1$ more stages).

[3] DFIFO is just a short name for Delayed FIFO.

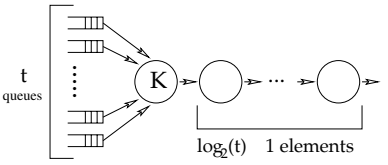


Fig. 4. An intermediate architecture.

It is now useful to introduce some straightforward lemmas:

Lemma 1. *A switch with DFIFO queuing architecture mimics a FCFS* OQ *switch with a delay of* $\log_2(n)$ *steps.*

Proof. As noted in Sect. 3, the architecture that, for every output, selects the oldest cell from t queues, exactly behaves like a FIFO *OQ* switch. Adding $\log_2(n)$ forwarding units, we just introduce a delay in the output.□

Lemma 2. *A mesh of trees switch mimics a switch with DFIFO queuing architecture with a delay of* $\log_2(n)$ *steps.*

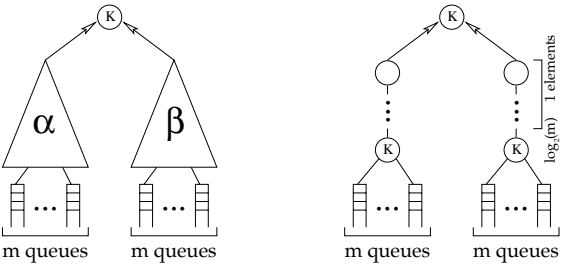
Proof. The delay is caused by the selection phase done at the column trees: as we assume infinite size queues, it takes exactly $\log_2(n)$ time slots to a cell for arriving to the queues, while in a DFIFO based switch they would arrive in one step. So it is enough to show that, without considering the column trees in the mesh of trees, the two architectures are totally equivalent.
We focus on an output j, in order to prove that cells bound for that output are handled in the same way (once they arrive at the queues) by the two architectures. Since we don't make any assumption on j, this will hold for all the outputs, establishing the lemma. We shall prove the lemma by induction on the number of queues[4]:

basic step: for $n = 2$ (2 inputs/outputs, the minimum case), the two architectures are exactly the same (the K element that chooses between two queues),
induction step: for $n = 2m$ and $m > 1$ the row tree can be seen as the composition of two trees with m leaves (queues) (see Fig. 5(a)). By induction, such an architecture is equivalent to the one shown in Fig. 5(b).

It is not hard to show the equivalence of such an architecture and a DFIFO of height $\log_2(m)$ (or equivalently $\log_2(2m) - 1$) forwarding elements. To avoid tedious details, it can be sufficient noting that

- the number of steps that cells must undergo, is the same ($\log_2(2m)$ after the first selection),
- during any time slot, if at level i of Fig. 5(b) architecture there is one cell, then there is one cell also at the same level of the DFIFO architecture,

[4] Given the mesh of trees features, we only deal with powers of 2, with 2 as bottom of the induction chain.



(a) Trees composition

(b) DFIFO composition

Fig. 5. Inductive step.

- in any time slot, if at level i of Fig. 5(b) architecture there are two cells, then in the DFIFO architecture there is one cell at level i and one cell at level $i+1$,
- inversely, at any time slot, if at level i of DFIFO architecture there is one cell, then either there is at least one cell at the same level of Fig. 5(b) architecture, or there are two cells at level $i-1$ (note that this holds for $i > 0$ since the root is unique in both systems).

So, at every time slot there is a cell in output in one architecture if and only if there is a cell in output in the other. Since outputs are time ordered, they must be exactly the same.□

Lemma 3. *Consider three switch architectures A, B and C. If A mimics B with a delay of $f(n)$ and B mimics C with a delay of $g(n)$, then A mimics C with a delay of $f(n) + g(n)$.*

Proof. By definition of mimicking with a delay (see Sect. 2), under the same arrivals, the output of C at time t is the same of B at time $t + g(n)$, which in turn is the same of A at time $t + g(n) + f(n)$.□

We have thus established the following

Theorem 1. *The mesh of trees switch mimics a FCFS* OQ *switch with a delay of* $2\log_2(n)$.

□

The mesh of trees architecture is particularly suitable to efficiently provide multicast. An addressing technique already known suffices: the destination of every cell is coded by a t bits string with the i^{th} bit set to 1 if and only if the output i is in the set of receivers. So, during the selection stage, the node at level

i in the column tree must only perform two "or" operations when a cell to route arrives: one of the bits in the left half of the word, and one of the rightmost ones. If the first "or" operation is equal to 1, then the cell is forwarded to the left child (with the left half word as destination information), and the same happens with the second operation, but the cell is forwarded to the right child (note that at least one operation must be positive, but both can produce a 1). The so implemented multicast is a copy multicast, and it is the most efficient way to implement it: the cells arrive at the queues during the same time slot (because of the synchronism of the selection phase), and will depart during the first empty time slot.
We believe this feature makes particularly interesting the proposed switch: the IQ architecture in fact has several problems managing multicast traffic, both from a theoretical point of view ([36]) and from a practical one (e.g. the simulations results in [37]), while in the mesh of trees switch the scheduling of multicast traffic practically comes for free. As previously established, the mesh of trees switch can mimic a FCFS OQ switch. Besides, for very large n's the OQ switch can be considered purely theoretical because of the needed speedup, while the mesh of trees scales very well. Moreover, the time slot length limit is given just by the memory speed: in fact, the whole architecture behaves like a pipeline, and the time of the system is given by the time of the slowest element. If a comparing step is faster than a memory cycle, we can think to group several comparing steps into a single system cycle, in order to reduce the delay of the mesh of trees and to improve performances.

The mesh of trees architecture seems to suffer of the same spatial speedup problem of the knockout switch: the queues in the leaves, for graphical presentation reasons, are drawn as adjacent, and at a first sight they can be imagined as a single big memory with a speedup problem. In the real physical implementation, memories not necessarily are positioned as in Fig. 3(a). Moreover, we think that, at least theoretically, the study of such a kind of architectures can be interesting, because of the positive performances offered that can overcome technical problems.

6 Conclusions

In this paper, we considered a parallel architecture for the implementation of the well known output queued switch. The widely studied mesh of trees topology has been used to propose a switch that can mimic (even if with a logarithmic delay) a FCFS output queued switch without the speedup problem. A future work will be to extend the class of queuing policies that is possible to emulate, in order to achieve quality of service, and to give some bounds on queues sizes and dimension of time stamps needed.

References

[1] M. G. Hluchyj and M. J. Karol. Queueing in high-performance packet switching. *IEEE Journal on Selected Areas in Communications*, 6(9):1587–1597, Dec. 1988.

[2] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, 11(4):319–352, Nov. 1993.

[3] N. Mckeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The tiny tera: a packet core switch. *Hot Interconnects IV, (Sstanford University)*, pages 161–173, Aug. 1996.

[4] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, G. D. Troxel, D. Waitzman, and S. Winterble. A 50 gb/s ip router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, Jun. 1998.

[5] J. P. Coudreuse and M. Servel. PRELUDE: an asynchronous time-division switched network. In *Proceedings of IEEE International Conference on Communications '87*, pages 769–773, 1987.

[6] N. Endo, T. Kozaki, T. Ohuchi, H. Kuwahara, and S. Gohara. Shared buffer memory switch for an ATM exchange. *IEEE Transactions on Communications*, 41(1):237–245, Jan. 1993.

[7] M. J. Karol, M. G. Hluchyj, and S. Morgan. Input versus output queueing on a space division switch. *IEEE Transactions on Communications*, 35:1347–1356, 1987.

[8] H. Zhang. Service disciplines for guaranteed performance service in packet switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, Oct 1995.

[9] M. Karol, K. Eng, and H. Obara. Improving the performance of input-queued atm packet-switching. In *Proceedings of IEEE INFOCOM '92*, pages 110–115, 1992.

[10] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, Dec. 1992.

[11] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proceedings of IEEE INFOCOM '98*, pages 533–539, 1998.

[12] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM '96*, pages 296–302, 1996.

[13] Y. Li, S. Panwar, and H. J. Chao. On the performance of a dual round-robin switch. In *Proc. of IEEE Infocom 2001*, 2001.

[14] A. Mekkittikul and N. McKeown. A starvation-free algorithm for achieving 100% throughput in an input- queued switch. In *Proceedings of the ICCCN*, pages 226–231, 1996.

[15] N. McKeown. *Scheduling algorithms for input queued cell switches*. PhD thesis, University of California at Berkeley, 1995.

[16] C.S. Chang, W.J. Chen, and H.Y. Huang. On service guarantees for input buffered crossbar switches: a capacity decomposition approach by birkoff and von neumann. In *IEEE IWQoS'99*, pages 79–86, 1999.

[17] R. E. Tarjan. *Data structures and network algorithms*. Society for industrial and apllied mathematics, 1983.

[18] C.S. Chang, W.J. Chen, and H.Y. Huang. Birkhoff-von neumann input buffered crossbar switches. In *Proc. of IEEE Infocom 2000*, 2000.

[19] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ ACM Transactions on Networking*, 7(2):188–201, Apr. 1999.

[20] M. W. Goudreau, S. G. Kolliopoulos, and S. B. Rao. Scheduling algorithms for input-queued switches: randomized techniques and experimental evaluation. In *Proc. of IEEE Infocom 2000*, 2000.

[21] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic (extended version, 1994.

[22] E. Leonardi, M. Mellia, F. Neri, and M. Ajmone Marsan. Bounds on average delays and queue size averages and variances in input-queued cell based switches. In *Proc. of IEEE Infocom 2001*, 2001.

[23] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999. (A preliminary version appears in Proceedings of INFOCOM '99).

[24] B. Prabhakar and N. McKeown. On the speedup required for conbined input and output queued switching. *Automatica*, 35(12):1909–1920, Dec. 1999.

[25] M. Andrews and L. Zhang. Achieving stability in networks of input-queued switches. In *Proc. of IEEE Infocom 2001*, 2001.

[26] F. M. Chiussi, D. A. Khotimsky, and S. Krihsnan. Generalized inverse multiplexing of switched atm connections. In *Proc. of IEEE Globecom '98*, 1998.

[27] F. M. Chiussi, D. A. Khotimsky, and S. Krihsnan. Advanced frame recovery in switched connection inverse multiplexing for atm. In *Proc. of IEEE International Conference on ATM '99*, 1999.

[28] D. A. Khotimsky and S. Krihsnan. Stability analysis of a parallel packet switch with bufferless input demultiplexor. In *Proc. of IEEE ICC 2001*, 2001.

[29] S. Iyer, A. Awadallah, and N. McKeown. Analysis of a packet switch with memories running slower than the line-rate. In *Proceedings of IEEE INFOCOM 2000*, 2000.

[30] S. Iyer and N. McKeown. Making parallel packet switches practical. In *Proceedings of IEEE INFOCOM 2001*, 2001.

[31] A. Mekkitikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings of IEEE INFOCOM '98*, pages 792–799, 1998.

[32] A. Mekkittikul. *Scheduling non-uniform traffic in high speed packet switches and routers.* PhD thesis, Stanford University, 1998.

[33] Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora. The knockout switch: A simple modular architecture for high performance switching. *IEEE Journal on Selected Areas in Communications*, SAC-5:1274–1283, Oct. 1987.

[34] F. T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, h ypercubes.* Morgan Kaufmann, 1992.

[35] F. P. Preparata and J. E. Vuillemin. Area-time optimal vlsi networks for matrix multiplication. 11(2):77–80, 1980.

[36] Z. Liu and R. Righter. Scheduling multicast input-queued switches. *Journal of scheduling*, 2(3):99–114, May 1999.

[37] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. On the throughput of input-queued cell-based switches with multicast traffic. In *Proc. of IEEE Infocom 2001*, 2001.