Published in IET Communications
Received on 17th September 2009
Revised on 3rd February 2010
doi: 10.1049/iet-com.2009.0609

ISSN 1751-8628

Multicast and quality of service provisioning in parallel shared memory switches

B. Matthews I. Arel D. Rose B. Bollinger

Department of Electrical Engineering and Computer Science, The University of Tennessee, 1508 Middle Drive, Knoxville, TN 37996-2100, USA
E-mail: itamar@ieee.org

Abstract: Growing demand for differentiated services and the proliferation of Internet multimedia applications requires not only faster switches/routers, but also the inclusion of guaranteed qualities of service (QoSs) and support for multicast traffic. Here, the authors introduce a parallel shared memory (PSM) architecture that addresses these demands by offering both QoS guarantees and support for multicast traffic. It is well known that PSM architectures represent an effective approach for distributing the high-memory bandwidth requirement found in output-queued (OQ) switches, while maintaining their desirable performance attributes. At the core of the PSM architecture is a memory management algorithm that determines, for each arriving packet, the memory unit in which it will be placed. The PSM architecture discussed should be considered with the context of fabric on a chip in mind, where an implementation is conceivable on a single chip, providing a plug-in emulated OQ switching solution. A description and detailed analysis of an efficient memory management algorithm that supports QoS and multicast traffic is given with a discussion of hardware implementation considerations that highlight the PSM architecture's scalability and performance attributes.

1 Introduction

Multicast provides an effective mechanism for conserving network bandwidth by enabling the efficient transmission of a single stream of data to multiple destinations. This has become increasingly important with the growing demand for multimedia applications and has led to considerable attention in the area of packet scheduling algorithms, as indicated by [1–4]. In concert with this increased demand for multicast traffic, there exists a concurrent need for service guarantees to support multimedia applications such as VoIP, IPTV and video on demand. The basic framework for these multimedia applications require that content be sent to multiple recipients simultaneously while meeting stringent performance requirements (low latency, minimal jitter, fairness and so on). For continued proliferation of these applications, the underlying infrastructure must provide efficient mechanisms for storage and forwarding of packets while offering quality of service (QoS) guarantees. Indubitably, this presents a significant challenge as we attempt to optimise performance while increasing the complexity in supporting multicast services.

To this end, OQ switches are appealing as they offer many desirable performance attributes, such as minimal packet delay, controllable QoS provisioning and work conservation under any admissible traffic. Their use is of highest priority in high-performance packet switching applications, driven greatly by the demands of data centres and internet backbones which require guaranteed throughput and room for growth on a budget. However, OQ switches are limited, in terms of scalability, due to high-memory bandwidth requirements. This bandwidth requirement can be observed when we consider an $N \times N$ switch, where N is the number of switch ports, which can have N packets with matching arrivals times destined for the same output port. In this scenario, both the output buffer and the switching fabric must operate N times faster than the line speed, R, to achieve full throughput. The resulting memory bandwidth requirement of $O(NR)$ is clearly impractical as line rates and port densities increase.

Alternate queueing approaches such as input-queued (IQ) [5] and combined input and output queued (CIOQ) architectures [6] have been extensively studied to address the high-memory bandwidth requirement found in OQ switches. A common goal of both IQ and CIOQ architectures is the desire to mimic the desirable attributes of OQ switches. However, the existing solutions often require significant complexity in the form of internal speedup, scheduling algorithms and additional queueing to approximate OQ switch performance. This complexity continues to grow as we consider the inclusion of additional features, such as multicast and QoS, and the persistent increase in line rates and port densities. Recent work in IQ Birkhoff–von Neumann switches [7–9] has paved way for guaranteed rates as well as multicast, although hardware complexity remains a problem.

In this context, we introduce the fabric on a chip (FoC) [10] approach which seeks to exploit recent improvements in the fabrication of VLSI circuitry in order to consolidate several key switching functions on a single silicon die. Observing recent advances in packaging technology, it is now possible for large amounts of information to simultaneously be forwarded to a single chip, which was not possible several years ago. Additionally, there are several key advantages that are attributed to the concept of FoC. First, it eliminates the need for virtual output queueing (VOQ) [11] as well as some output buffering associated with standard switch architectures. Second, by exploiting the ability to access the multiple on-chip Mbits of dual-port SRAM, packets can be internally stored and switched through a simple crossbar without the need for external memory devices. The crosspoint switches and scheduler, pivotal components in IQ switches, are also avoided thereby substantially reducing chip count and power consumption. Third, much of the signalling and control information that typically spans multiple chips can be carried out on a single chip. Finally, the switch management and monitoring functions can be centralised and provisioned far more easily given all the information is available at a single location.

To enable this approach, we employ a parallel shared memory (PSM) architecture that resolves the high-memory bandwidth requirements dictated by OQ switches, while maintaining their desirable performance attributes. PSM architectures employ a pool of small memory units operating at line rate, in parallel, to distribute the bandwidth requirement. At the core of the PSM architecture resides a memory management algorithm which, for an arriving packet, determines the memory unit in which it will be placed. We describe a novel placement algorithm that ensures placement without conflict and extend this framework to support multicast and QoS. Additionally, we incorporate memory and computation speedup, while also introducing load balancing, to yield an efficient high-speed memory management algorithm with attainable scalability properties.

The rest of the paper is organised as follows. In Section 2, prior work on PSM architectures is presented. Section 3 describes the proposed memory management algorithm while also introducing placement and computation speedup. Section 4 discusses hardware implementation considerations. Section 5 introduces architectural extensions to the FoC in the form of multicast, QoS provisioning and load balancing, while the conclusions are drawn in Section 6.

2 Prior work on scaling PSM switches

There are two key challenges to utilising PSM switches to emulate OQ switches. The first of which is determining how many parallel memories are necessary to guarantee packet placement while providing sufficient bandwidth. Initial work proposed by Iyer *et al.* [12] has indicated that a sufficient number of memories needed for a PSM switch to emulate a first-come first-served (FCFS) OQ switch is $K = 3N - 1$. This assumes that each of the shared memory units can perform at most one packet-read or -write operation during each time slot. This sufficiency condition can be proved using constraint sets analysis (also known as the ‘pigeon hole’ principle), summarised as follows. An arriving packet must always be placed in a memory unit that is not currently being read from by any output port. Since there are N output ports, this first condition dictates at least N memory units are required. In addition, no arriving packet may be placed in a memory unit that contains a packet with the same departure time. This results in additional $N - 1$ memory units representing the $N - 1$ packets having the same departure time as the arriving packet, which may have already been placed in the memory units. Should this condition not be satisfied, two packets would then be required to simultaneously depart from a single memory unit, which can only support a single read operation in each time slot. The third and last condition states that all N arriving packets must be placed in different memory units (since each memory can only perform one write operation). By aggregating these three conditions, it is shown that $3N - 1$ memory units are sufficient in order to guarantee FCFS OQ emulation. It is important to note that while this limit on the number of memories has been shown to be sufficient [12], it has not been shown to be necessary. We also not that, as shown in [13], the emulation of OQ switching as a deterministic scheduling process can be attained using various switching fabrics. Employing analysis based on the pigeon hole principle, it is enough to show that any given switching architecture performs identically to a pure OQ switch.

The second challenge involves selecting a practical scheduling algorithm to determine in which memory a packet should be placed. The source of this difficulty might not be immediately obvious but consider the sources of contention previously discussed. It is clear that the subset of available memories for any given packet is rather specific and improper selection could result in packet loss.

Recent work by Prakash *et al.* [14, 15] proposed a variation of the PSM switch, termed the switch–memory–switch (SMS) architecture. This SMS architecture represents an abstraction of the M-series Internet core routers from Juniper. The approach attempts to address the second issue by statistically matching input ports to memories, based on an iterative algorithm that statistically converges in $O(\log N)$ time. However, in this scheme, each iteration comprises multiple operations of selecting a single element from a binary vector. Although the nodes operate concurrently from an implementation perspective, these algorithms are $O(\log^2 N)$ at best [assuming $O(\log N)$ operations are needed for each binary iteration as stated above] [16]. Given the critical nature of timing within the targeted device, it is imperative the computational complexity directly reflects the intricate nature of the digital circuitry involved, as opposed to the high-level algorithmic perspective.

To address the placement complexity issue, we propose a pipelined memory management algorithm that reduces the gate level complexity of placing a packet in a buffer to $O(\log N)$. The subsequent cost associated with reducing the placement complexity is an increase in the number of required parallel memories to $O(N)$ and a fixed processing latency. The justification resides in the new found ability to store and switch packets on chip, in particular large FPGA devices, as multiple megabits of dual-port SRAM are now available. There are several ancillary benefits of this approach. First, it is plausible to transmit all data packets to the FoC directly, eliminating the need for VOQ [11] as well as a portion of the output buffering commonly found in the existing router designs. In addition, the scheduler and requisite crosspoint switches, found in IQ and CIOQ-based architectures, provide a reduction in both chip count and power consumption. In achieving a greater degree of integration from this continued consolidation, a substantial reduction in overall resource consumption is expected.

3 Packet placement algorithm

The primary focus of this paper is the memory management algorithm which determines the exact memory unit in which to place each arriving packet. Accordingly, this work extends on the design of large-scale PSM switches from a single-chip realisation perspective. In this paper, we introduce the column-associative packet placement scheme which provides a high-speed memory management algorithm with a memory requirement of $O(N)$. In subsequent sections, we will extend this architecture to provide multicast support, QoS guarantees and load balancing capabilities.

3.1 Column-associative packet placement algorithm

In a conventional shared memory architecture, packets arrive at each ingress port where they are segmented into fixed size cells for efficient placement in memory. Prior to the presentation of a cell to the switch fabric, a departure time must be determined in order to establish the appropriate departure sequence for a given set of arriving cells. We initially consider a first-come-first-serve (FCFS) scheduler whereby packets are assigned departure times in accordance with their arrival order. In later sections, we will provide support for more sophisticated schedulers whose departure time assignments are intended to provide delay and rate (QoS) guarantees. Regardless of the exact scheduler applied, the focus of this section is the memory management algorithm that attempts to distribute the memory bandwidth requirement across multiple PSMs.

The proposed memory management algorithm, illustrated in Fig. 1, consists of $M \times N$ cell buffering units arranged in a rectangular structure, whereby cells enter at the left-most column and are guaranteed placement upon reaching the final column. Each column in this structure is associated with a single memory unit resulting in a total of M parallel memories. Packets arriving at an ingress port i are segmented into k byte cells prior to insertion into row i, where k is determined by memory unit capacity. Consequently, cells located in the same column will contain a matching arrival time when this scheme is utilised. At the end of each cell time, all cells located in column j will be transferred to column $j+1$, providing every cell with the opportunity to be placed in each of the M available parallel memories.

The cell placement unit, located in each column, determines whether a cell, located in a column, can be placed in the corresponding column memory. A cell will be placed in the associated column memory if the following conditions are met: (1) the memory associated with the column does not already contain another cell with an identical departure time; (2) the cell is selected for placement over the potential $N-1$ other packets with the same arrival time. As a cell progresses through each subsequent stage, memory placement contentions are

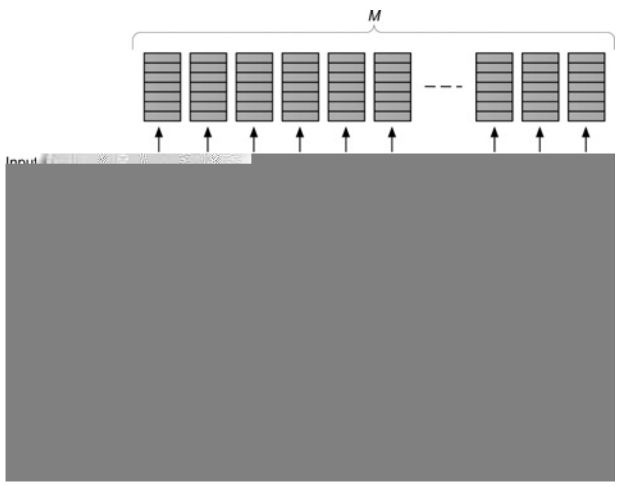

Figure 1 *Cells are selected for placement into each column memory based on their departure time*

reduced. In this scheme, once a cell is shifted to the last column of the pipeline, it is guaranteed to be placed in a memory that does not contain a cell with a matching departure time.

In the proposed architecture, rows can be viewed as simple shift registers, whereby cells are shifted one stage to the right at each time step. At each stage of the pipeline, a single cell assignment is attempted per column. The motivation for this approach is to reduce the complexity of the placement algorithm by isolating memory assignments, thus minimising memory contention. Routing of cells from the PSMs to the egress ports naturally concludes the placement process and can be achieved without the use of a final crossbar, for example, through an N-way multiplexer at the egress ports.

3.2 Basic analysis of resource requirements

Theorem 1: A total of $2N-1$ dual-port column memories is sufficient for an N-port switch utilising the proposed packet placement algorithm.

Consider an arriving cell, C_1. It will find at most $N-1$ other cells with an identical arrival time competing for the same memory. Furthermore, at most $N-1$ other cells with the same departure time could have been placed in unique memories prior to the arrival of C_1. Assuming one cell can be read and another written to a memory, then at least $(N-1)+(N-1)+1=2N-1$ memories are required to guarantee the placement of cell C_1.

The cell placement unit, located in each column, maintains a mapping of memory locations to corresponding to departure times, which have been reserved by cells successfully placed in the memory. This mapping, which is effectively a binary mask, shall be referred to as the column's occupation vector. In addition, each column maintains a mapping that specifies pre-allocated (or reserved) departure times of cells that have yet to be placed in a column memory. This mapping is referred to as the column's request vector. The placement element in the pipeline performs a bitwise AND operation over the occupation and request vectors to determine if there is a cell with a departure time available for placement in the memory. If a cell is selected for placement, it will be extracted from the pipeline and written the corresponding column memory. Cells not selected for placement are subsequently forwarded to the next stage of the pipeline.

Consider the simple scenario depicted in Fig. 2 illustrating the state of the pipeline for three consecutive time slots (i.e. t, $t+1$, $t+2$). At time t, three cells are inserted into the first column of the pipeline with departure times {1, 2, 1}, respectively. Assuming that all memories are initially empty, the placement engine will simply select the first cell in the column for placement. Thus, the cell in row 1 with departure time 1 is placed in the memory associated with column 1. The remaining cells {2, 1} will shift right to column 2 at time $t+1$, while new cells, with departure times {1, 3, 2}, are inserted into the first column. The placement element at column 2 will now select the cell with departure time 2, leaving the only remaining cell, {1}, to shift to column 3. In column 1, a cell with departure time 1 has already been placed in the memory associated with column 1. Thus, the placement element in column one must select the cell at row 2, with departure time 3, for placement. At time $t+2$, additional cells, with departure times {2, 4, 3} are introduced into the pipeline at column 1. As the cell with departure time 2 has yet to be selected, it is next selected for placement. Additionally, columns 2 and 3 both select cells with departure time of 1.

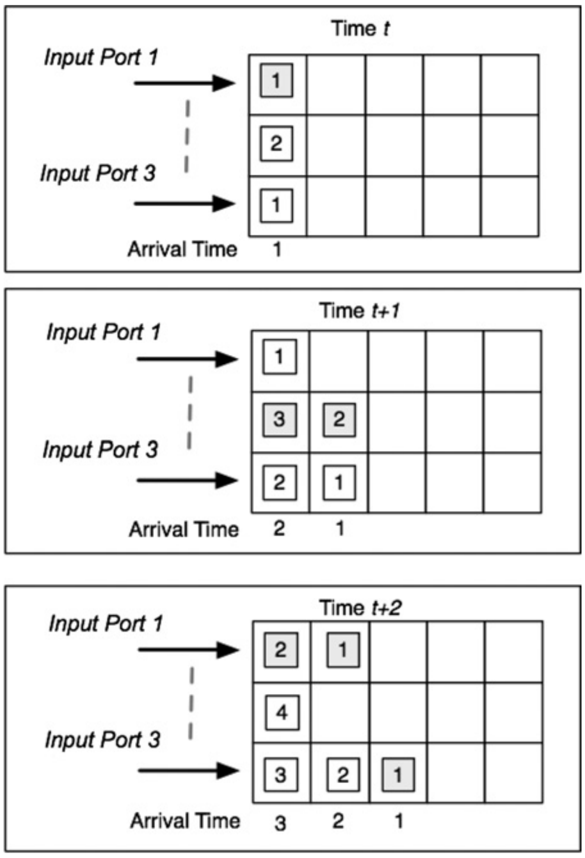

Figure 2 *Example illustrating the proposed memory management algorithm for a three-port switch*
The state of the pipeline structure is depicted for three consecutive time slots

It can be observed from this example that the column-based memory management algorithm clearly provides an effective mechanism for the resolution of memory contention.

Scalability is achieved as decisions are disassociated from the number of ports in the system. Increased port densities do not directly imply an increase in the time required to make a placement decision. Instead, placement decisions are determined only by the number of departure times offered by the switch fabric in the form of the occupation and request vectors. In the context of FoC, these mapping vectors are bound only by the depth of an individual column memory.

The coupling of placement decisions with memory depth is not unbounded. In all practical switching systems, once a buffer approaches (or is close to approaching) its limit, flow-control signalling should be provided to the traffic sources, indicating the need to either slow down or temporarily stop the flow of packets to a particular destination. Such a mechanism is always required since instantaneous data congestion may occur at any router or switch. In fact, even if the traffic is said to be statistically admissible, implying that no input or output is oversubscribed, it may still be the case that for short periods of time a given output port is oversubscribed. To address such scenarios, and in an effort to reduce the probability of packet loss, the line cards typically host large memory spaces. As such, each parallel memory unit contained in the FoC is relatively shallow with its depth governed by the system response to back pressure and the number of memory units employed.

3.3 Incorporating speedup

To achieve a further reduction in the number of required memories, it is clear that we must limit memory contention in the packet placement process. One source of contention is a blocked placement that occurs as a result of offering only one memory location per departure time for each memory unit. This constraint can be alleviated by provisioning multiple cell placements for a single column memory, allowing a single memory to host $1 < m \leq N$ cells with identical departure times. As a result, a cell C, with departure time d, will consider a memory unavailable for placement only if it contains m cells, having arrived prior to C, that have a matching departure time.

Theorem 2: $((m+1)/m)N-1$ memories are guaranteed to be sufficient for an N-port switch, where each memory can hold m cells with the same departure time.

Consider the arrival of cell C_1 to the first column of the switch. It will find at most $N-1$ other packets with the same arrival time competing for the same memory. Moreover, there are at most $N-1$ other packets with the same departure time that may have been placed in a memory prior to the arrival of C_1. As cells are presented to memories sequentially, each offering multiple departure times, the $N-1$ cells with matching departure times must be contained in one of $N/m-1$ memories. Therefore a total of $((N/m)-1)+(N-1)+1=(m+1/m))N-1$ memories are required to ensure placement of C_1.

It should be noted that provisioning multiple cell placements for each column memory does not impact the time required to establish a memory's availability. The primary trade-off in this instance resides in the increased density of each memory unit and the requirement that packets must be read from memory at a rate m prior to being forwarded to the appropriate egress ports. Having made this observation, we introduce a placement (or computation) speedup of s, which assumes the pipeline operates at a rate s times faster than the line rate. One advantage of operating the pipeline at an increased rate is reduced latency. Moreover, cell arrivals from a set of N inputs can be presented to the switch fabric in groups consisting of N/s cells. This further provides a reduction in the number of conflicts associated with packets possessing the same arrival time from N to N/s.

Theorem 3: A total of $(s+1/s)N-1$ memories is sufficient for an N-port switch with a placement speedup of s.

For a placement speedup of s, an arriving cell, C_1, will find at most $(N/s-1)$ other packets with the same arrival time competing for the same memory. Furthermore, at most $N-1$ other packets with the same departure time may have been placed in unique memories prior to the arrival of C_1. Therefore at total of $((s+1)/s)N-1$ memories are required to ensure placement of cell C_1 [16].

Given that columns are interlocked with memories, the incorporation of placement speedup implies a memory write speedup equal to s. Thus, we can utilise both multiple cell placements and placement speedup concurrently to gain an even greater reduction in memory requirements. In view of the above assertions, it should be evident that the sufficient number of memories in this case is $k=((s+m)/sm)N-1$. Moreover, we achieve a reduction in the size of the cell buffering structure such that $k \times (N/s)$ cell buffering units are now required.

4 Implementation considerations

4.1 Critical path analysis

If we now consider the implementation aspects of the pipelined memory management architecture, we quickly realise the critical path in the design resides is the placement decision process that must be made by the placement primitives located in each column. Each primitive must determine whether a cell residing in column $C_i (i \in [1, 2, \ldots, k])$ can be placed in the associated column memory. If the cell is determined to be a viable candidate for placement, it is extracted from the pipeline and written into the column memory. Otherwise, the cell will advance to the next column where its placement viability will be re-evaluated for the next column memory.

To determine the feasibility of placing a cell in a column memory, each memory maintains a binary occupation vector, of length k denoting the depth of the memory, to indicate departure times that are currently reserved. To determine which cells in a given column are available for placement, a requests vector is created as cells are inserted into the pipeline. This vector, also of length k, provides a bitmap corresponding to the departure time for each of the cells located in a column. For unicast traffic, a single bit is set to indicate the requested location. In our discussion of multicast traffic, we utilised the notion of a bound, b, which limited the number of replications, or placement requests, that could be made by a single cell. Consequently, a requests vector for a multicast cell will have at most b bits set to indicate the requested placement location.

The requests and occupation vectors are constructed and maintained in order to obtain the set of viable cells (referred to as the candidates vector). Each cell is represented by a single bit in the candidate vector which allows a priority encoder to be used such that a cell can quickly be selected for placement into the corresponding column memory. The cumulative result of this decision process is that a given cell is either written to the memory associated with the column in which it resides or shifted to the next stage of the pipeline.

Given that the AND operation can be achieved at high speed, it becomes apparent that the priority encoder is the predominately time-consuming function. The complexity of a priority encoder is generally acknowledged to be $O(\log(N/s))$, where N denotes the number of elements at its input and s represents the placement speedup. It is noted that this complexity pertains to a single bit-level operation (rather than more complex arithmetic abstraction), clearly suggesting that the method is very efficient from a computational standpoint.

4.2 Alternative pipeline structure

The column-associative packet placement algorithm discussed thus far consists of $M \times N$ cell buffering units arranged in a rectangular structure, where N is the number of switch ports and M represents the number of parallel memories. In its most basic implementation, cells are introduced at the left-most column and are guaranteed placement upon reaching the final column. Once a placement decision is made, the cell is written directly to memory. As a result, there can be up to M write operations that occur in parallel. While the architecture will support M concurrent write operations, the system only needs to support an aggregate of N write operations per time slot.

The use of $M \times N$ cell buffering units implies a $c \times M \times N$ register requirement, where c is the cell size. If we consider a 32-port switch employing 63 memories, this would require approximately 1 Mbit of registers to implement a pipeline switching 64 byte cells. To reduce this requirement, we observe that at most N cells are presented to the fabric during a single time slot. In addition, each of these N cells are guaranteed placement upon reaching the final column. Therefore it is feasible to simply indicate which of the M memories a cell should be written to as it passes through the pipeline. Once the N potential cells arrive at the final column, they can all be written to memory simultaneously. Memory contention is avoided as the pipelined placement algorithm dictates that all cells, with the same arrival time, contain placement assignments to unique memories.

Given that cell placements can be delayed until the final column, it is no longer necessary to propagate each cell through the pipeline. Rather, we can extract the departure time from each arriving cell, then en queue the cell in a delay buffer of depth M. Noting that only the departure time is required by a cell placement unit, we can simply forward the extracted departure time to the pipelined memory management algorithm. As the departure time for each cell propagates through the pipeline, each column placement unit will attach the index of the associated column memory to a selected departure time. Once all departure times have reached the final column, the corresponding cell will be read from the delay buffer and placed in the memory indicated by the attached index.

Propagating only the departure time and a memory index through the pipeline provides considerable reduction in the register requirement. The placement pipeline no longer requires $c \times M \times N$ registers to store cells as they are being placed. Instead, each cell buffering unit is only required to store the departure time, DT, and the index of the memory for which the cell has been marked for placement. Therefore the pipeline needs to be of size $(\lceil \log_2(\mathrm{DT}) \rceil + \lceil \log_2(M) \rceil) \times M \times N$, which is considerably smaller than a full 64-byte cell. Considering, for example, a 64-port switch employing 127 memories and 256 departure times, we can now reduce to the register requirement to be approximately 28 kbits, considerably less than the 1 Mbit required when cells are propagated. This reduction does have an associated cost in that we now require N multiplexers to place each cell in one of the potential M memory destinations.

5 QoS provisioning and multicast support

5.1 Memory provisioning for multicast traffic

The memory requirement for the proposed architecture has, to this point, been defined considering unicast traffic only. Given that each unicast cell can be destined for at most one egress port, the analysis is considerably easier since there can be at most one departure time assignment associated with each cell. With multicast traffic, each cell can be destined to multiple egress ports. The exact number of copies is expressed in terms of the maximum fan-out q, $q \in \{1, \dots, N\}$, which denotes the number of egress ports for which a cell, C, can be destined. Consequently, a total of q departure time assignments must be made for C.

One straightforward approach for processing multicast cells, known as copy multicast [17], is to simply replicate the cell at the ingress, thus creating q unicast copies which are each delivered to the fabric and switched individually. Copy multicast has obvious drawbacks in that it requires a fabric speedup of q at the ingress, while also potentially requiring q additional memory locations to store the copied cells in a column memory. This is derived from the fact that each ingress port can receive a multicast packet destined for up to q egress ports.

To more efficiently store cells, an alternate approach, termed fan-out multicasting, can be employed. In fan-out multicasting, a multicast cell is transmitted to the switch fabric only once and avoids replication. Instead, only one copy of the cells is stored in memory and is read by the fabric multiple times for delivery to each of the q destinations. From an effective memory utilization perspective, this is an optimal solution. However, achieving this goal is a significant challenge given the difficulty of the placement process.

$(q+1)(N-1)+1$ memories are sufficient for a PSM switch, utilising the column-based packed placement algorithm, with fan-out multicasting.

Consider fan-out multicasting in a PSM switch, we observe an arriving multicast cell, C_q, will contain α, $\alpha \le q$, unique departure times, which means there are α potential departure time conflicts. For the column-based placement process using fan-out multicasting, we recognise that a departure time conflict with any of the α unique departure times results in a placement failure. For each of the α unique departure times associated with the arriving cell, there are at most $(N-1)$ other cells with the same departure time that may have been placed in unique memories prior to the arrival of C_q. Ignoring arrival time constraints for the moment, clearly the placement of the multicast cell C_q requires $q(N-1)+1$ memories to resolve departure time contention when fan-out multicasting is employed.

A linear reduction in this memory requirement can be achieved with the inclusion of memory speedup, m. If we allow multiple cells having the same departure time to be written to the same memory, we reduce the departure time constraint such that

$$k = \left(\left\lceil \frac{q}{m} \right\rceil + 1\right)(N-1) + 1 \qquad (1)$$

memories are now sufficient to ensure placement of a multicast cell into memory. However, this approach has limited scalability given the speedup needed to minimise the number of required memories cannot be easily attained.

The formulation of an optimal policy for placing cells into shared memories should incorporate the beneficial aspects of both copy and fan-out multicasting. For copy multicasting, we seek to minimise the number of replications required to place a packet without increasing the number of column memories required. To accomplish this goal, we utilise the positive aspects of fan-out multicasting by bounding the number of replications that can occur. This can be achieved by allowing the association of a finite number of departure times, b, to occur for some number of replicated cells. The approach restricts the maximum number of duplicate packets, or replications, to $\lceil q/b \rceil$. From Lemma 5.1, we recognise the memory sufficiency condition can now be represented by

$$k = (b+1)(N-1) + 1 \qquad (2)$$

Further optimisation of the multicast placement algorithm can be attained with the introduction of both memory and computation speedup.

Offering a memory speedup, m, provides a mechanism for reducing the departure time constraint associated with allowing multiple departure times to be contained in one cell. Additionally, the incorporation of computation speedup, s, enables cells to be introduced in groups of N/s which reduces the arrival time constraint. As a result, the introduction of both memory and placement speedup yields a simultaneous reduction in both the arrival and departure time constraints.

Theorem 4: $\lceil (sb+m)/sm \rceil N - b$ memories are sufficient for an N-port multicast PSM switch utilising the proposed packet placement algorithm.

For the N-port multicast switch with computation speedup, there are at most b departure times associated with a given multicast cell, C_m. Additionally, there can be at most m cells located in a given column memory with departure times that match those contained in C_m. As placement opportunities occur sequentially, C_m can be denied placement into at most $b((N/m)-1)$ memories due to a departure time constraint violation. Furthermore, an arriving multicast cell, C_m, will find at most $(N/s)-1$ other cells with the same arrival time. Therefore, a total of $b((N/m)-1)+((N/s)-1)+1 = \lceil (sb+m)/sm \rceil N - b$ memories are required to ensure placement of cell C_m.

Bounding the number of replications that can occur for each multicast cell facilitates an optimal trade-off between placement complexity, required memories and memory depth. For fewer cell replications, the replication bound, b, can be increased which will result in reduction of the required memory depth. While this yields fewer duplicate cells, the number of unique memory instances will increase given the added departure time conflicts. This can be minimized if we tightly couple the replication bound with computation and memory speedup. For $s = m$, the memory sufficiency condition becomes $\lceil (b+1)/s \rceil N - b$. In doing so, we are able to limit the number of unique memory instances by only increasing the replication bound, b, in parallel with the speedup, s. Table 1 outlines the number of memories needed for various port densities and m and s values.

5.2 Providing QoS guarantees

In a network of OQ or shared memory switches, it has been demonstrated that a scheduler capable of providing weighted fairness among flows, or delay guarantees, at all queueing points in the network can yield end-to-end guarantees for well-behaved flows [18]. Obviously, the FCFS scheduler considered thus far is incapable of providing such fairness, or delay guarantees, as the scheme is biased towards flows that transmit at high rates.

Implicit in our goal of controlling delay is an underlying need to modify the relative departure times of packets residing in memory. This can be accomplished if we allow the departure time assignment policy to follow any push-in first-out (PIFO) [13] queueing discipline (WFQ, GPS, strict priority, etc.). In a PIFO queueing discipline, the relative transmission order of cells remains fixed once placed in the queue. Arriving cells may be inserted into the queue at any point, but always depart from the head of line. The departure time assigned to an arriving cell is performed prior to insertion. The scheduling policy can dictate cells within a packet be given sequential departure times, thus enabling performance guarantees for both packets and cells.

These changes in scheduling for QoS introduce both new challenges for application to the PSM design and, as a result of the measures taken to resolve these challenges, a penalty that must be paid for a final solution. In the following described method, multiple PIFO queues for each PSM must be employed. This change from FIFO to PIFO creates two significant obstacles. First, multiple PIFOs introduce the potential for modification of relative departure order of packets. This reordering implies that a true PIFO discipline is not strictly enforced and therefore the positive attributes gained when using PIFOs are not guaranteed. To regain PIFO status, packets need to be reordered at the output with the conflict-free permutation modification described shortly below. This technique provides maximum throughput at the cost of a fixed delay.

A second and perhaps less obvious issue arises from the fact that unrestricted placement of priority packets means that memory requirements can grow without bound. However, constraint set analysis can be employed to determine a set of rules to govern the minimum memory requirements for such a system to guarantee placement while still maintaining throughput. Speedup's effect on the memory requirements provide further direction towards reducing memory units and lowering the total number of pipeline stages. With performance in mind, the total fixed delay observed by an arriving cell is finally considered. The sources of a packet's wait are the inherent delay of placement, the pipeline and the delay associated with the reordering buffer, which we examine in more detail after setting out the necessary new placement process.

Consider the single egress queue employing a PIFO queueing discipline, containing cells destined for output a only. In this example, we denote the cell destined for output a, with departure time 2, as a_2, departure time 3 as a_3 and so forth. Initially, three cells are destined to leave output a in order, as depicted in Fig. 3a. Now consider the insertion cell of a_2' into the PIFO queue. In this instance, a_2' is scheduled to depart prior to cell a_2. Thus, it will be 'pushed in' just after cell a_1. The insertion of a_2' will obviously delay the departure of cells a_2 and a_3, as depicted in Fig. 3b, by one cell time. This is the desired outcome, as all cells will depart in the same relative order.

Table 1 Number of memories required for the proposed PSM multicast switch

Switch ports (N)	Speedup (s)	Replication bound (b)	Maximum replications	Memory units
16	1	1	16	31
16	2	2	8	30
16	4	4	4	28
32	1	1	32	63
32	2	2	16	62
32	4	4	8	60
64	1	1	64	127
64	2	2	32	126
64	4	4	16	124

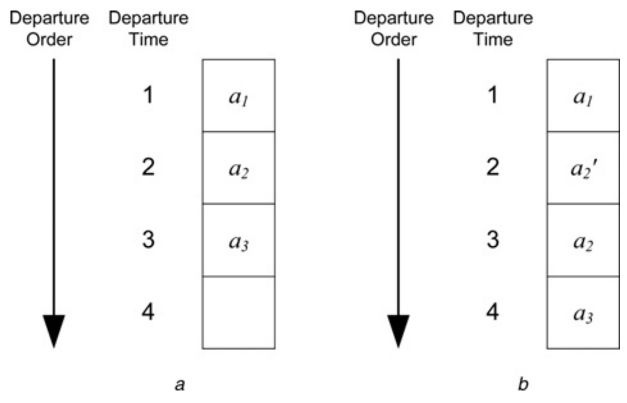

Figure 3 *Initial ordering of a PIFO queue prior to insertion of cell a_2' and ordering of that PIFO queue after cell a_2' has been 'pushed in'*

a Prior to insertion of cell a_2'
b After cell a_2' has been pushed in

The introduction of PIFO queues does, however, introduce contention not evident in the previous example. In that instance, all cells were destined for a single output a and stored in the same PIFO queue. This is convenient for the purpose of highlighting the operation of a PIFO queue. However, this neglects the potential conflict associated with cells destined for an output other than a in a system consisting of k PIFO queues. Allowing cells to be placed into memories regardless of their departure times introduces a conflict, whereby a cell destined for output a could be pushed into a PIFO queue such that it modifies the relative order of cells destined for the other $N - 1$ egress ports.

To highlight the difficulty in managing k PIFO queues, we consider the insertion of multiple cells into a subset of k memories, to demonstrate conflicts that can can occur. In Fig. 4*a*, we have three PIFO queues containing packets destined to ports a, b and c. The cells $\{a_1, b_1, c_1\}$ will depart together in the first time slot, then followed by cells $\{a_2, b_2, c_2\}$ in the next time slot and so forth. First, we consider the insertion of cell a_2', depicted in Fig. 4*b*, into the first PIFO queue, just prior to cell a_2. As observed in the previous example, the relative order of departures of cells destined to port a remains unchanged. However, the relative order of cells destined for port c does change, as c_3 will now depart one time slot later than its scheduled departure time. Now consider the insertion of b_3' into the second PIFO queue. We again face a similar problem given cell a_3 will also depart one time slot after its scheduled departure time. Further, cells destined for the same output, b_3' and b_3, will now depart at the same time from separate memories. As multiple cells can be delivered simultaneously to the same egress port, an output buffer, operating at $O(NR)$, is now required to eliminate output contention at each egress port. This approach is impractical as it reintroduces the high bandwidth requirement found in OQ switches, while also eliminating their desirable flow isolation attributes.

To resolve the memory contention present when multiple PIFO queues are employed, we first seek to avoid conflicts for

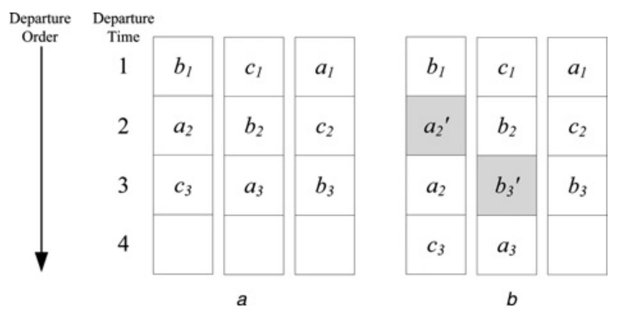

Figure 4 *Initial ordering of a PIFO queue prior to insertion of cell a_2' and b_3' and ordering of PIFO queue after cells a_2' and b_3' have been 'pushed in'*

a Prior to insertion of a_2' and b_3'
b After cells a_2' and b_3' have been pushed in

cells destined to different output ports. To do so, we utilise the conflict-free permutation approach introduced in [12], whereby the relative departure order is modified to avoid read conflicts. With this approach, we adjust the read process such that each output, in a router consisting of N outputs and k shared memories, is allowed to read at most one cell from each of the k memories. If we consider a set of k memories, whereby each memory is denoted by m_i, $i \in k$, then cells destined for output a would be read from memories using the read sequence $\{m_1, m_2, m_3, \dots, m_k\}$. As N cells will need to depart every time slot to obtain the maximum throughput, we define a read window, w, that consists of N cells (one cell destined for each egress port), such that N memories will be simultaneously read. With this approach, the read sequence for additional outputs will need to be shifted as depicted in Fig. 5 in order to perform simultaneous reads for each output. Consider cells destined for output b, the read cycle must start from m_2 such that its read sequence does not conflict with sequence used by output a. Thus, for output b, the read sequence would be $\{m_2, m_3, \dots, m_k, m_1\}$. Assuming there is a cell in every memory destined for each output, then the application of this approach will achieve the maximum throughput given each output will receive k cells in an interval consisting of k cell times.

As we have elected to use an alternate read process, we have introduced an additional source of contention. This contention is derived from the fact that at most one cell can be read, during a single read cycle consisting of k cell times, from each of the k memories for the individual outputs. For instance, if β cells, $2 < \beta \leq k$, are destined for the same output and happen to be placed in the same memory, then only one cell will depart in the scheduled interval of k departure times. This results in the delay of $\beta - 1$ cells, such that these cells will no longer depart during their assigned interval. This implies additional placement constraints are needed to ensure each of the k cells scheduled to depart in a given read cycle are placed in unique memories.

If we reconsider the scenario described above employing the alternate read process, we must first define a read cycle, ς, to consist of three cell times. Prior to the insertion of a_2', ς consists of the cells $\{a_1, a_2, a_3\}$. As such, each cell must be located in separate memories to depart in the correct order. After the insertion of a_2', ς now contains the cells $\{a_1, a_2', a_2\}$ which all must be placed in unique memories. Further, if we considered the insertion of a cell a_1', clearly ς would then contain the cells $\{a_1', a_1, a_2'\}$, where the three cells are yet again required to be located in unique memories. At this point, we have inserted two cells and now require five unique memories to store each of the cells. If we allow incoming cells to be 'pushed in' to memories without restriction, the number of unique memory instances required will clearly increase without bounds.

In outlining the placement constraints, we recall that the read process is structured such that k cells are delivered to

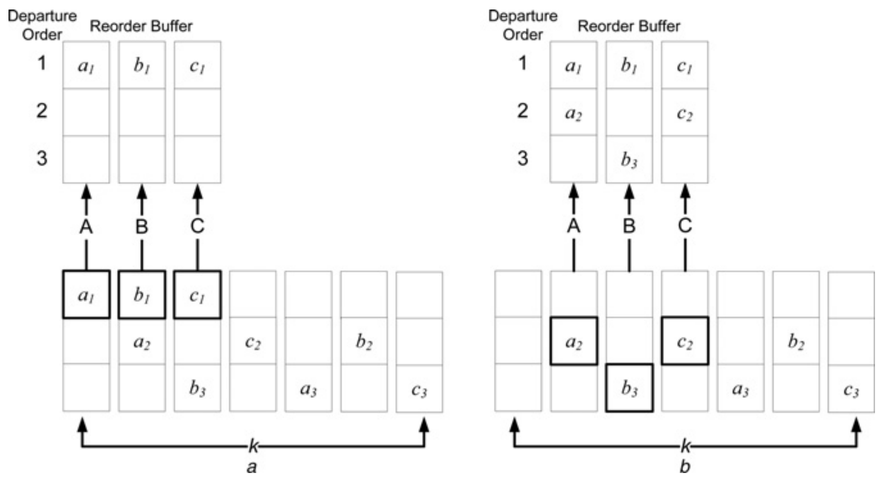

Figure 5 *Each egress port reads from a given shared memory into a reorder buffer and on the next cycle, the egress ports will shift to the next memory until all k memories have been read*

a Egress port reads from a shared memory
b Egress ports shift to next memory

each of the N outputs in a read cycle consisting of k cell times. The exact number of shared memories, k, necessary to support this read operation must be $k \geq N$, since each output must read at least one packet from each memory without contention. For $k = N$, an arriving cell C_1 must not be 'pushed in' to one of the $N - 1$ memories containing cells destined to depart in the $N - 1$ time slots prior to C_1. This is apparent given that each of the N cells scheduled to depart in the k cell times, all destined for the same output, must be located in separate memories. While this ensures C_1 will be depart in the correct interval of k cell times, its insertion can still produce conflicts with cells scheduled to depart after C_1. This is obvious as the relative departure order of cells following C_1 is altered resulting in the scenario presented above. To resolve this issue, we restrict C_1 from being placed in the $N - 1$ memories containing cells scheduled to depart just after it, which are also destined for the same output port. These two constraints can then be combined to state an arriving cell, C_1, should not be placed into the $N - 1$ memories containing cells, destined for the same output port, that are scheduled to depart either before or after C_1. From these constraints, we observe that $k = 2(N - 1) + 1 = 2N - 1$ memories is sufficient to ensure no placement conflicts will occur.

The result we have just presented, however, neglects the arrival time conflicts that are a result of the column-associative packet placement algorithm. If we consider an arriving cell can be blocked by at most $N - 1$ cells having the same arrival time, we recognise the memory sufficiency condition increases such that, $k = 2(N - 1) + (N - 1) + 1 = 3N - 2$ memories are now sufficient to ensure placement occurs without conflict. Further reduction in the memory requirement can be achieved by incorporating speedup as discussed in Section 3.3. Since PIFO queues are employed, we do not need to offer multiple departure times for each memory, instead we utilise both memory write- and read-speedup, s, to provide a reduction in resource requirements. Cells will again be introduced to the pipeline in groups of N/s and written to memory with speedup s. Additionally, we will now read cells from memory in groups of N/s to achieve a simultaneous reduction in pipeline depth and memory requirements. Incorporating the speedup gains, we observe the memory requirement decreases such that, $k = 2((N/s) - 1) + ((N/s) - 1) + 1 = 3(N/s) - 2$ is now sufficient.

In reading a complete window of cells, the order in which cells arrive at a given output is not guaranteed. The read process only ensures that all k cells scheduled to depart, in a period of k cells times, will be delivered to each output in a given interval. Since the order is not guaranteed, we recognise a small reordering buffer containing at least $k - 1$ cells is required to hold cells associated with a given read cycle. To determine the total delay observed by an arriving cell, we must include the delay of the pipeline, k cell times, and the delay associated with the reordering buffer, $k - 1$ cell times. Aggregating the delay associated with both the placement process and the reorder buffer, we note that a PSM router, utilising the column-associative packet placement algorithm, can emulate an output queued router within $2k - 1$ time slots.

5.3 Load balancing

Our discussion thus far has assumed that cells are inserted into the memory management pipeline at the left-most column and are shifted right at each time slot. As cells propagate through the pipeline, column placement units select a single cell for storage in the associated column memory. The progression of cells through each subsequent stage of the pipeline results in placements that tend to be concentrated near columns closest to their insertion point. This is intuitive considering that for every cell in a given pipeline column there must be a corresponding cell in that column's memory with a matching departure time. As a group of arriving cells, always located in the same column, advances in the pipeline, the likelihood that a conflict exists for all cells in the column decreases significantly. In addition, we note that after $N-1$ successive occurrences where no packets are placed, at least one packet is guaranteed to be placed in the successive N columns.

To illustrate the clustering of cell placements around the insertion point, we present simulation results, shown in Fig. 6, for a 32-port switch with 63 memories. In highlighting the clustering effect, we employ a FIFO queueing scheme. Since the clustering of cells is the result of a horizontal progression of cell placements (not the queueing discipline), the results are equally relevant for PIFO queueing schemes. In the simulation results presented, cell arrivals are uniformly distributed across the destinations and obey a Bernoulli i.i.d. process with a normalised traffic load of one (where a packet is arriving at every time slot to emulate the worst-case scenario). As evident from the queue distribution presented, cell placements are densely clustered around the insertion point and droped off considerably after N placement opportunities.

From an implementation perspective, we would prefer the distribution of cells across the shared pool of memories be uniformly distributed. This enables dynamic queue management algorithms and thresholds to be more efficiently implemented. In order to more effectively distribute cell placements, the elements at which cells are inserted into the pipeline must vary for each port to avoid clustering. To accomplish this goal, a rotation technique is employed, whereby cells arriving at port i are inserted into the column that precedes the insertion point used in the previous time slot. For example, if at time t a cell was inserted into column φ, then all cells inserted at time $t+1$ must be placed in column $\varphi - 1$. If a cell is inserted into

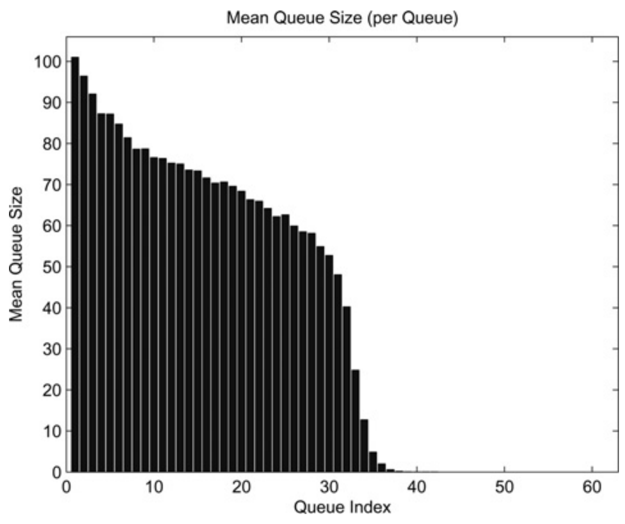

Figure 6 *Queue size distribution for a PSM system without load balancing*

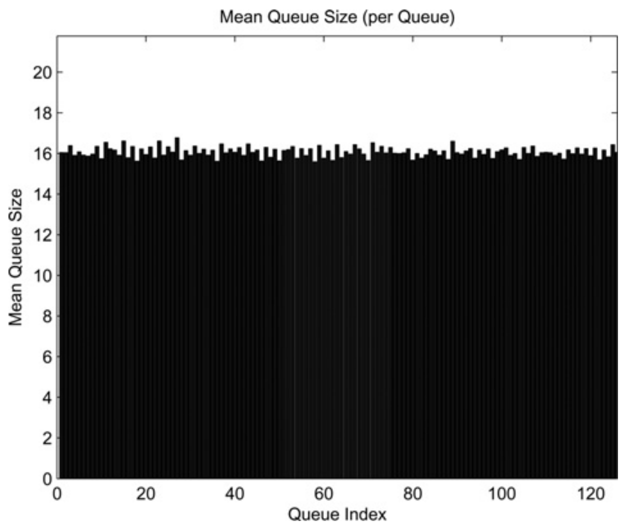

Figure 7 *Queue size distribution with load balancing*

the rightmost column and has yet to be placed in memory, it is shifted to the left-most column, representing a circular shift, to avoid cell loss. As cells advance in a clockwise manner and insertion points occur counter-clockwise, then we must ensure that a cell will not meet an insertion point within $2N-1$ time slots to avoid cell loss.

Theorem 5: Uniform distribution of cells across the memories, employing a FIFO queuing discipline, can be achieved with $4N-2$ memory units.

For a cell C_1 inserted at column φ_1, there must be $2N-1$ columns prior to the insertion of another cell. Thus, an additional $2N-1$ columns are required to rotate the insertion points to ensure uniform distribution across the memories. Aggregating the two terms, we conclude that $4N-2$ memories are sufficient to achieve a balanced cell distribution.

In Fig. 7, a 32-port switch with 126 memories is simulated assuming maximal load. It is observed that a rather uniform memory occupancy distribution results, with fewer cells stored in each memory.

6 Conclusion

In this paper, we introduced a novel architecture for PSM switches that provides QoS guarantees, as well as support multicast traffic. In the context of emulating an OQ switch, it has been argued that a fundamental challenge pertains to the memory-management algorithm employed. A proposed memory management algorithm that supports multicast traffic, while also providing QoS guarantees, was described in detail. We have established the memory requirements needed to emulate the ideal OQ switching performance through the use of a pigeon-hole analysis. The proposed architecture was shown to facilitate speedup and load balancing, both of which are important for scalability and efficient use of resources. These features are well suited for the realisation of a packet routing architecture that requires both service guarantees and proficient switching capabilities. The switch model and framework presented here can be broadened to further investigate the concept of consolidating multiple switch fabric functions on silicon.

7 Acknowledgment

This work has been partially supported by the Department of Energy (DOE) under research grant DE-FG02-04ER25607.

8 References

[1] MINKENBERG C.: 'Integrating unicast and multicast traffic scheduling in a combined input- and output-queued packet-switching system'. Proc. Ninth Int. Conf on. Computer Communications and Networks, Las Vegas, NV, USA, 2000, pp. 127–134

[2] NONG G., HAMDI M.: 'Providing qos guarantees for unicast/multicast traffic withfixed/variable-length packets in multiple input-queued switches'. Proc. Sixth IEEE Symp. on Computers and Communications, Hammamet, Tunisia, 2001, pp. 166–171

[3] IYER S., MCKEOWN N.: 'On the speedup required for a multicast parallel packet switch', *IEEE Commun. Lett.*, 2001, **5**, pp. 269–271

[4] MARSAN M.A., BIANCO A., GIACCONE P., LEONARDI E., NERI F.: 'Multicast traffic in input-queued switches: optimal scheduling and maximum throughput', *IEEE/ACM Trans. Netw.*, 2003, **11**, pp. 465–477

[5] MCKEOWN N.: 'The iSLIP scheduling algorithm for input-queued switches', *IEEE/ACM Trans. Netw.*, 1999, **7**, pp. 188–201

[6] PRABHAKAR B., MCKEOWN N.: 'On the speedup required for combined input and output queued switching'. Technical report CSL-TR-97-738, 1997

[7] CHANG C.-S., LEE D.-S., JOU Y.-S.: 'Load balanced Birkhoff–von Neumann switches, Part I: one-stage buffering', *Comput. Commun.*, 2002, **25**, (6), pp. 611–622

[8] CHANG C.-S., LEE D.-S., LIEN C.-M.: 'Load balanced Birkhoff–von Neumann switches, part II: multi-stage buffering', *Comput. Commun.*, 2002, **25**, (6), pp. 623–634

[9] CHANG C.-S., LEE D.-S., YUE C.-Y.: 'Providing guaranteed rate services in the load balanced Birkhoff–von Neumann switches', *IEEE/ACM Trans. Netw.*, 2006, **14**, pp. 644–656

[10] MATTHEWS B., ELHANANY I., TABATABAEE V.: 'Fabric on a Chip: Towards consolidating packet switching functions on silicon'. Proc. IEEE Int. Conf. on Communications (ICC), June 2006

[11] TAMIR Y., FRAZIER G.: 'Higher performance multiqueue buffers for vlsi communication switches'. 15th Annual Symp. on Computer Architecture, 1988, pp. 343–354

[12] IYER S., ZHANG R., MCKEOWN N.: 'Routers with a single stage of buffering'. Proc. 2002 SIGCOMM Conf., 2002, vol. 32, pp. 251–264

[13] CHUANG S.-T., GOEL A., MCKEOWN N., PRABHAKAR B.: 'Matching output queueing with a combined input/output-queued switch', *IEEE J. Sel. Areas Commun.*, 1999, **17**, pp. 1030–1039

[14] AZIZ A., PRAKASH A., RAMACHANDRA V.: 'A near optimal scheduler for switch–memory–switch routers'. Proc. Fifteenth Annual ACM Symp. on Parallel Algorithms and Architectures, July 2003, pp. 343–352

[15] PRAKASH A., AZIZ A., RAMACHANDRA V.: 'Randomized parallel schedulers for switch–memory–switch routers: analysis and numerical studies'. IEEE INFOCOM 2004, 2004

[16] PRAKASH A., SHARIF S., AZIZ A.: 'An o(log2 *n*) parallel algorithm for output queuing'. IEEE INFOCOM 2002, 2002

[17] LEE J., SOHN J., LEE M.: 'Design of a shared multi-buffer ATM switch with enhanced throughputin multicast environments'. IEEE Proceedings ATM Workshop, 1999, (Kochi, Japan), 1999, pp. 455–461

[18] PAREKH A.K., GALLAGER R.G.: 'A generalized processor sharing approach to flow control inintegrated services networks: the single-node case', *IEEE/ACM Trans. Netw.*, 1993, **1**, pp. 344–357