A Pipelined Memory Management Algorithm for Distributed Shared Memory Switches

Xike Li, *Student Member, IEEE,* Itamar Elhanany, *Senior Member, IEEE**

Abstract— **The distributed shared memory (DSM) packet switching architecture has attracted much attention recently, predominantly due to its ability to overcome the inherent memory-bandwidth limitation of output-queued switches. Its performance has been studied from a theoretical point of view by exploring the conditions under which it can emulate an output-queued switch. At the core of the DSM design is a memory management algorithm that determines the memory units to which arriving packets are forwarded. However, the complexity of such algorithms found to date is $O(N)$, where N denotes the number of ports in the system, thereby inherently limiting the scalability of the scheme. In this paper, we propose a novel pipelined memory management algorithm for DSM switches which offers reduced timing complexity at the cost of fixed latency. Moreover, we demonstrate how processing and memory-access speedup factors yield a highly scalable DSM switch architecture design.**

Keywords— **Distributed Shared Memory Switch, Output Queueing Emulation, Packet Switching, Pipelined Algorithms.**

I. Introduction

Output queued (OQ) switches offer several highly desirable performance characteristics, including minimal average packet delay, controllable Quality of Service (QoS) provisioning and work-conservation under any admissible traffic [1]. However, the memory bandwidth requirements of such systems is $O(NR)$, where N denotes the number of ports and R the data rate of each port. This requirement significantly limits scalability of OQ switches with respect to their aggregate switching capacity.

In an aim to mimic the desirable attributes of output-queued switches, many commercial routers employ centralized shared memory (CSM) switch fabric architectures. In a typical CSM implementation, as a packet arrives at a switch port it is placed into a shared buffer that holds other packets destined to the same output port. In order to guarantee work conservation, so long as there is at least one packet in such an output queue, the corresponding output port is to be busy transmitting. If a switch is work conserving, the throughput of the switch is maximized and the average latency experienced by arriving packets is kept to a minimum. It has further been shown that using a well designed output link scheduling algorithm, such as Weighted Fair Queueing (WFQ) [1] or Deficit Round Robin [2][3][4] , the delay of any individual packet and the rate of different flows can be controlled in order to provide strict QoS guarantees.

*The authors are with the Electrical and Computer Engineering Department at the University of Tennessee (e-mails: xli6@utk.edu, itamar@ieee.org). This work has been partially supported by the Department of Energy research contract DE-FG02-04ER25607.

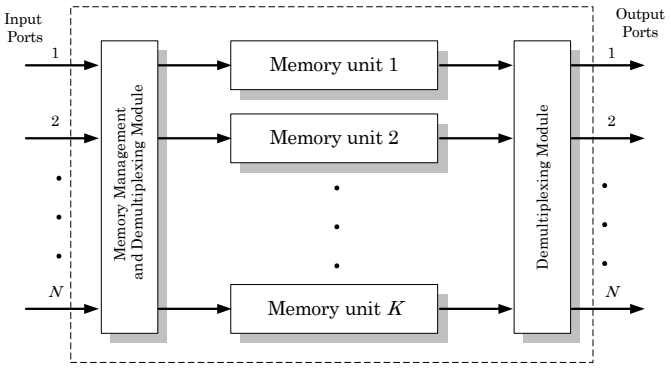

Fig. 1. Generic structure of a DSM switch architecture. Incoming packets are placed in a set of K (>N) shared memory units.

However, in order to operate properly, the shared memory in a CSM switch must have sufficient bandwidth. The latter is derived from the need to be able to accept (write) up to N arriving packets while, simultaneously, transmit (read) up to N departing packets. Clearly, if each port runs at a data rate of R, an aggregate memory bandwidth of $2NR$ is necessary. The latter significantly limits the scalability of the CSM design, and yields a very impractical scheme for switches with high line rates and/or with a large number of ports. Packets of 64 bytes in length traversing a $10Gbps$ link, for example, are approximately $50nsec$ in duration. If 32 ports are to be supported by the CSM switch, the memory must be able to read and write packets at less than $800psec$, which is impractical.

In this paper, we propose a novel memory management algorithm for DSM switches which employs a pipelined hardware architecture. The algorithm parallelizes the packet placement process, thereby gaining execution speed at the expense of a fixed latency. In addition, we analyze the conditions on the number of memories required for the given algorithm to guarantee that the DSM switch successfully emulates a First-Come-First-Served (FCFS) output-queued switch.

Typical network platforms, particularly at the Internet backbone where ATM is commonly deployed, partition variable size packets (such as IP) into fixed sized datagrams. Processing fixed-size data units has proven both practical and easier to study. To that end, in our model all packets are assumed to be of fixed size.

The paper is structured as follows. In section 2 a brief overview of prior results pertaining to the DSM switch is presented. Section 3 reviews the sequential process for placing packets into memories and establishes the bounds on any parallel memory management scheme. The novel

pipelined memory management algorithm, with and without speedup, is described in detail in sections 4 and 5, respectively, while the conclusions are drawn in section 6.

II. The Distributed Shared Memory Switch

An intuitive approach to resolving the memory bandwidth constraint for a single shared memory device is to employ a set of shared memories in parallel instead of using only one memory device. Two related architectures are derived from the above approach [5]. The first is the Parallel Shared Memory (PSM) switch, in which all the shared memory resources are located in a central location, and the second is the Distributed Shared Memory (DSM) switch, whereby the memories are distributed among the line cards. Logically, the DSM router is equivalent to a PSM router and, accordingly, the theorems pertaining to the PSM scheme apply directly to DSM switches. We label the duration of each fixed-length packet as a *time slot*.

At the core of the DSM design is a memory management algorithm that determines the memory unit in which each incoming packet is placed, as illustrated in Figure 1. Initial work has indicated that, assuming each memory can perform at most one read or write operation during each time slot, a sufficient number of memories needed to emulate a FCFS output queued switch is $K = 3N - 1$ [5].

The latter can be proven by employing constraint sets analysis (also known as the "pigeon hole" principal), summarized as follows. An arriving packet must always be placed in a memory unit that is currently not being read from by any output port. Since there are N output ports, this first condition dictates at least N memory units are available. In addition, each arriving packet must not 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, that may have already been placed in the memory units. Should this condition not be satisfied, two packets will be required to simultaneously depart from a memory unit that can only produce one packet in each time slot.

A 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 at least $3N - 1$ memory units must exist in order to guarantee FCFS output queueing emulation.Although this bound on the number of memories is sufficient, it has not been shown to be necessary. In fact, a tighter bound was recently found suggesting that at least $2.25N$ memories are necessary [6].

However, regardless of the precise minimal number of memories used, a key challenge pertains to the practical realization of the memory management mechanism, i.e. the process that determines the memories in which arriving packet are placed.

III. Sequential Packet Placement Algorithm

A. Lower Bound on the Number of Memory Units

Let K denote the number of memory units available in a DSM switch. We assume that we have knowledge of the departure time of each arriving packet prior to its placement in the shared memory bank. For FCFS emulation, establishing the departure time of arriving packets is rather straightforward. In addition, we begin by assuming that the shared memories are all single port memory devices, which means that they perform a single read *or* write operation during a single time slot, but not perform both. Let $A_{i,j}(t)$ be an indicator function representing the arrival of a packet at input port i destined to output port j at time t. Correspondingly, we define $\tau(A_{i,j}(t))$ to be the departure time of that packet, if the latter exists.

As indicated in section II, we can not place a packet into a memory which is currently being read from or one that contain a packet with the same departure time as that of the arriving packet. By employing a sequential memory management algorithm, i.e. placing one packet at a time, all potential contention scenarios are avoided. We define a binary memory occupancy matrix, $M_{u,f}(t)$, in which u denotes the index of the memory unit and f the offset (address) value in that memory, such that

$$M_{u,f}(t) = \begin{cases} 1 & \text{if a packet exists at offset } f \text{ of} \\ & \text{memory } u \text{ at time } t \\ 0 & \text{else} \end{cases} \quad (1)$$

We note that $u = 1, 2, \ldots k$ and f is assumed to be unbounded. In the sequential scheme, the straightforward packet placement algorithm is: an arriving packet from input port x destined to output port y with departure time τ is placed in any memory unit, u, for which $M_{u,\tau}(t) = 0$. Accordingly, K must satisfy the inequality:

$$K \geq N + N - 1 + N \geq 3N - 1. \quad (2)$$

If we consider dual-port memory devices, which can perform both a read and a write operation during a single time slot, the algorithm need not exclude memories already occupied by packets whose departure time is t, thus reducing the total number of memory units needed to

$$K \geq N - 1 + N \geq 2N - 1. \quad (3)$$

B. Inherent Limitations

Although the above observations have significant theoretical implications, the bottleneck preventing this technology from scaling is clearly the sequential nature of the memory management mechanism. We next address the inherent limitations imposed on any memory management algorithm for a range of DSM architectures.

Let us begin our discussion by investigating an intuitively ideal scenario in which there exist N^2 shared memory units. In this case a sequential packet placement algorithm becomes trivial: an arriving packet from input port x destined to output port y with departure time τ is placed in

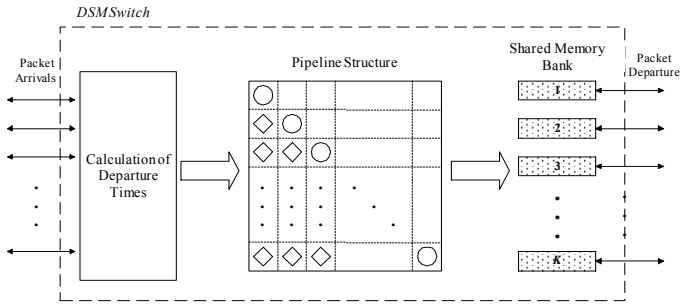

Fig. 2. Pipelined packet placement algorithm. Time-space tradeoff is exploited to gain execution speed at the cost of a fixed latency of N time slots.

one of the N memories (out of the N^2) associated with output port y. Hence, there is little decision making to be done on the part of the memory management algorithm. A closer look at such a design reveals a fundamental drawback: it requires that there would be N^2 physical memory modules implemented since any N memories can be written to concurrently. This means that bundling several logical memory units into a single physical memory device is impossible, thereby imposing a notable constraint on the scalability of such a switch.

The need for N^2 physical memory units can be relaxed if we consider a memory access speedup factor, i.e. assume that the rate at which a packet can be written to a memory is r times faster than its arrival rate. Such an assumption is reasonable when considering, for example, packet duration times of $50nsec$ and memory (SRAM) access times of approximately $6nsec$ (i.e. $r \approx 8$). Given that r write cycles can be completed within a single time slot, using a sequential placement algorithm the number of physical memory units is directly reduced to N^2/r.

Clearly, the inherent constraint on scalability here shifts towards the memory access time. Ideally, we would like to have a DSM memory management algorithm for which the number of physical memory units required is kept to a minimum while memory access time does not become a limiting factor. Relaxing the memory access time offers two alternative implementation advantages: (1) memory devices/ cores with smaller data buses can be used, thereby reducing design complexity and cost, and (2) by speeding up the memory access rate by r a single physical memory unit may be shared among r input ports, thus significantly increasing the feasibility of the design.

IV. Pipelined Memory Management Algorithm Without Speedup

A. Pipelined Architecture

Consider an N-port DSM switch in which arriving packets progress through a pipelined structure prior to being placed into the shared memories, as illustrated in figure 2. Let the pipelined structure be represented as an $N \times N$ matrix, P. Columns of P contain packets that have arrived at the same time, while the rows correspond to the input port numbers. Packets arriving at time t are placed in the leftmost (first) column of the pipeline matrix. Following each time slot, all columns are shifted one position to the right and the newly arriving packets are, once again, positioned in the first column. During each time slot, the algorithm will issue decisions (i.e. select one available memory for each packet) in parallel for all arriving packets.

Copying of packets from the pipeline structure to the shared memories occurs only for packets that are on the diagonal of P (marked by circles in figure 2). Before we elaborate on the rationale behind the placement of packets in the memories, we note that the preliminary processing stage in which packets are assigned departure times can be arbitrarily long, so long as it is constant. The respective delay simply adds a fixed latency to the overall switching process.

Packets that arrive at the same time are copied to memories during different times (due to the diagonal placement approach). In order to make sure they depart at the correct relative time, a constant N is added to the calculated departure time of each packet. The latter guarantees similar identical departure times to packets arriving to the switch simultaneously at the cost of a fixed latency of N time slots. What remains to be described is the manner by which the diagonal packets in the pipeline structure are assigned to memory units. We define the term *availabilitymap* of an arriving packet to denote the set of all memories to which this packet may be written. The underlying concept governing the memory assignment process is that an identical availability map is created for all of incoming packets so as to guarantee that each packet is assigned a valid and non-conflicting memory.

B. Packet Placement Algorithm

In order to illustrate the main constrains imposed on the memory management algorithm and derive the minimal number of memories needed, we consider the extreme case in which all N arriving packets have identical departure times. Clearly, each of the packets is destined to a distinct output. Let's look at the conditions that need to be met in order for the placements to be valid and non-conflicting:

1. Even though the packets are later copied to the memories at different time steps, each must select a distinct memory since they will be required to depart (be read from memory) at the same time in the future.
2. Each packet must not be assigned a memory unit which has been previously selected by other packets residing on the diagonal line of the arriving packet. This is to guarantee that N packets on the diagonal are each written to a different memory unit.

If a single and identical availability map is to be generated for all arriving packets, the number of memories needed is at least $N(N-1)/2$, corresponding to selections already made by packets residing in the lower triangle of the pipeline structure (excluding the first column). Since all packets may have the same departure time, N additional memories should be available. Last, there should

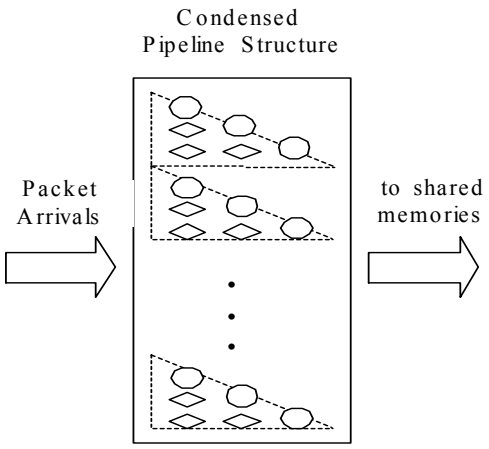

Fig. 3. The pipelined packet placement algorithm with processing speedup ($s > 1$). The structure comprises of s triangular structures each pertaining to N/s time steps.

be a strict mechanism under which two or more packets with identical departure time select different memories (when referencing the similar availability map). The most straightforward way of achieving this goal is for packets arriving at input port i to select the i^{th} available memory. This can be easily implemented using a priority encoder design. An extreme case is that in which the N^{th} (last) packet has a unique departure time, in which case N additional memories are needed to guarantee no conflicts. In conclusion, the total minimal number of memories required is given by

$$\begin{aligned} K &\geq \underbrace{\frac{N(N-1)}{2}}_{pipeline\ structure} + \underbrace{N}_{\substack{concurrent \\ arrivals}} + \underbrace{N}_{\substack{concurrent \\ departures}} \\ &\geq \frac{N(N+3)}{2} \end{aligned} \tag{4}$$

V. Pipelined Memory Management Algorithm With Speedup

The assumption thus far has been that the concurrent assignment of arriving packets to memories required an entire time slot (packet time) to complete. However, since the algorithm comprises simple, albeit large, binary operations over the availability map, it is reasonable to assume that the propagation delay of such operations is a fraction of a packet duration. We let the processing speedup factor, s, denote the ratio of packet duration to a single packet assignment time. The reader will note that since N packets may be arriving at any given time slot, N decisions regarding packet-to-memory assignments must be made concurrently. If the latter does not hold, packets will inevitably be lost. Practical values of s may be as high as 8 if, for example, the assignment operation consumes ~6 $nsec$ while a packet time is ~50 $nsec$.

We exploit this processing speedup by condensing the computation process and complexity of the pipeline architecture, as shown in figure 3. Here, each triangle shown in the figure corresponds to a step in the assignment process in which N/s packets are assigned to memories.

TABLE I
NUMBER OF MEMORY UNITS IN THE PROPOSED DSM SWITCH

	$N=32$		$N=64$		$N=128$	
	$s=4$	$s=8$	$s=4$	$s=8$	$s=4$	$s=8$
$r=4$	44	28	152	88	560	304
$r=8$	22	14	76	44	280	152

* *r denotes the memory access speedup and s the processing speedup*

In order to complete N assignments, s consecutive iterations are needed during each time slot. By partitioning the process in this manner, we achieve several goals. First, the latency introduced by the pipeline is reduced to N/s time slots, as can be observed by the reduction in the number of columns. Second, and more importantly, the number of overall memory units needed is reduced to

$$\begin{aligned} K &\geq \underbrace{\frac{N}{s}\left(\frac{N}{s}-1\right)\frac{s}{2}}_{pipeline\ structure} + \underbrace{N}_{\substack{concurrent \\ arrivals}} + \underbrace{N}_{\substack{concurrent \\ departures}} \\ &\geq \frac{N(N+3s)}{2s} \end{aligned} \tag{5}$$

Note that for $s = N$ we have $K = 2N$, which corresponds to the theory pertaining to the sequential memory management algorithm with dual-port memories. In addition, since the requirement from the memory access time is still one time slot, we may further exploit the fact that a packet may be written to a memory in a shorter period of time. Consequently, we may reduce the number of physical memories required by a factor of r to yield

$$K \geq \frac{N(N+3s)}{2sr} \tag{6}$$

Consideration of both contributing speedup factors yields a very low requirement for physical memory devices and areas on a chip. Table 1 provides a breakdown of the minimal number of memory units needed in a DSM switch employing the proposed pipelined memory management algorithm, for different values of s, r and N. If we define x as the ratio between the number of memory units to N, we observe that in most cases $x < 2$, which is an extremely attractive attribute from an implementation perspective. The implications of the latter are that, utilizing existing VLSI technology, a $Terabit/sec$ switching fabrics is within the realm of feasibility using the proposed approach.

VI. Conclusion

In this paper, we have described a pipelined memory management algorithm for DSM switches that emulate FCFS output-queued switches. The proposed scheme exploits a time-space tradeoff to offer reduces computational complexity of the memory management process, thereby gaining scalability at the cost of fixed latency. Moreover, by considering memory access speedup, the number of physical

shared memory units required is further reduced, yielding a highly scalable and pragmatic switch architecture. Future work will focus on applying the pipelining concept to non-FCFS scheduling disciplines in order to support QoS provisioning.

References

[1] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344–357, June 1993.

[2] J. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," *Proc. of IEEE INFORCOM '96*, pp. 120–128, March 1996.

[3] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *Proc. of ACM SIGCOMM '95*, pp. 231–242, September 1995.

[4] S. K. A. Demers and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM Computer Communication Review (SIGCOMM '89)*, pp. 3–12, 1989.

[5] R. Z. S. Iyer and N. Mckeown, "Routers with a single of buffering," *ACM Computer Communication Review (SIGCOMM'02)*, pp. 251–264, 2002.

[6] H. Liu and D. Mosk-Aoyama, "Memory management algorithms for dsm switches," *Stanford Technical Paper*, July 2004.