Benes Switching Fabrics with $O(N)$-Complexity Internal Backpressure

Georgios Sapountzis and Manolis Katevenis
Institute of Computer Science - FORTH, and University of Crete, Greece
ICS-FORTH, P.O. Box 1385, Heraklion, Crete, GR-711-10 Greece
http://archvlsi.ics.forth.gr/bpbenes/ - {sapunjis,katevenis}@ics.forth.gr

Abstract— **Multistage buffered switching fabrics are the most efficient method for scaling packet switches to very large numbers of ports. The Benes network is the lowest-cost switching fabric known to yield operation free of internal blocking. Backpressure inside a switching fabric can limit the use of expensive off-chip buffer memory to just virtual-output queues (VOQ) in front of the input stage. This paper extends the known backpressure architectures to the Benes network. To achieve this, we had to successfully combine per-flow backpressure, multipath routing (inverse multiplexing), and cell resequencing. We present a flow merging scheme that is needed to bring the cost of backpressure down to $O(N)$ per switching element. We prove freedom from deadlock for a wide class of multipath cell distribution algorithms. Using a cell-time-accurate simulator, we verified operation free of internal blocking, we evaluated various cell distribution and resequencing methods, we found that delay under bursty traffic is only 25 to 50 percent higher than ideal output queueing, and we showed that the delay of well-behaved flows remains unaffected by the presence of congested traffic to oversubscribed output ports.**

Topics Keywords: Switches and switching.
Methods Keywords: System design, Simulations.

1 . INTRODUCTION

Switches, and the routers that use them, are the basic building blocks for constructing high-speed networks that employ point-to-point links. As the demand for network throughput keeps climbing, switches are needed with both faster ports and more ports. This paper concerns *switch scalability* when the *number of ports increases*. For low to modest numbers of ports –up to about 64– the crossbar is the switch topology of choice, owing to its simplicity and non-blocking operation. However, its cost grows with N^2, where N is the number of ports, which makes it very expensive for large N. Additionally, crossbar scheduling is a hard problem, and gets much harder with increasing N.

1.1. Multistage Fabrics and Related Work

For switches with hundreds or thousands of ports, *multistage switching fabric* architectures are needed, whose cost growth rate is less than quadratic. Researchers have been looking at such scalable fabric topologies since the days of electromechanical telephony [1]. The banyan network [2] features a low cost, $N \cdot logN$ and a rich set of paths. Although it can support full egress link utilization under uniformly destined traffic, as well as a number of other specific traffic patterns, it does suffer from *internal blocking:* not all *feasible* rates $\lambda_{i,j}$ (see section 2.1) can be routed through it. The lowest-cost $N \times N$ network that is free of internal blocking is the *Benes* network [3], whose cost is $N \cdot 2logN$. The Benes network is *rearrangeably* non-blocking, that is, when each connection is routed through a single path, setting up new connections may require the re-routing of existing connections; however, using multi-path routing, this disadvantage can be eliminated: see section 2.1. This paper concerns the Benes network.

If a multistage switching fabric contains no buffer storage, there must exist a mechanism to handle the cell routing conflicts that arise *(a)* in internal paths due to the routing algorithm, and *(b)* due to output conflicts. The former conflicts can be handled in a distributed manner ("self-routing fabrics") using Batcher sorting networks [4]. The latter conflicts –cells destined to the same output at the same time– must be avoided at the inputs or tolerated in the fabric. Avoidance at the inputs is equivalent to crossbar scheduling and requires global coordination, hence it is unrealistic for large fabrics. To tolerate output conflicts in the fabric, designers have used recirculation of cells [5] or multiple paths to each output buffer [6]. All of these mechanisms cost a lot in number of stages and paths per stage in the switching fabric: the fabric cost is $O(N \cdot log^2N)$, and the constant in front of the actual cost is significant. In essence, these techniques spend (expensive) communication resources in order to economize on (inexpensive) storage resources, which is the wrong tradeoff in modern VLSI technology.

It is preferable for the switching fabric to contain internal buffer storage, in order to buffer conflicting cells until the conflict goes away. Such internal storage may be "small" enough to fit inside the switching-element chips, or it may be "large" enough to replace the buffer space typically found on the ingress line cards, –usually hundreds of MBytes– hence requiring off-chip DRAM. In the former case, *backpressure* is used to prevent the small buffers from overflowing; effectively, the majority of the buffered cells are pushed back onto the ingress line cards, as in the usual case of virtual-output queues (VOQ) on the input side. Given that the ingress lines are much fewer than the intra-fabric links, this architecture results in significant cost savings when compared to the off-chip DRAM case for intra-fabric buffers, as shown by the ATLAS I switch evaluation [7]. Lucent's ATLANTA chip set uses a 3-stage buffered switching fabric with internal backpressure[1] [8]. Several other commercial chip sets also use backpressure in the ingress-switch-egress connection chain [9] [10]. This paper concerns the application of this advantageous *internal backpressure* architecture to the Benes network –the lowest cost scalable switching fabric.

A different approach to buffer placement in multistage switching fabrics is the *Parallel Packet Switch (PPS)* [11] [12], which is a derivative of central (shared) buffering and uses memory interleaving to provide scalable memory throughput. PPS is defined as a 3-stage fabric, where the bulk of the buffer space resides in the central stage. If one were to scale PPS to arbitrarily large aggregate throughput, one would end up with something along the lines of [13] : an input switching fabric (e.g. crossbar or banyan), which connects the input ports to the memory banks in the central stage and performs inverse multiplexing; the interleaved memory shared buffer in the central stage; and an output switching fabric (e.g. crossbar or banyan), which connects the memory banks to the output ports. This construction is similar to the construction of the Benes network from two banyans, which we review in section 2.1, but differs in buffer placement. The hard part of PPS is how to coordinate the traffic through its two fabrics so that the bulk of the queues end-up in the central stage, without using impractical centralized scheduling; several different proposals [14] and improvements [12] [13] exist. Their drawbacks are that either they need $O(N^2)$ logical queues in *each* of the $O(N)$ memory modules (banks), or they use a static schedule of length $O(N)$ in their switching fabric(s), or both. A quadratic number of queues poses serious cost scalability problems, while inverse multiplexing with a static schedule introduces long delays. By comparison, the architecture of this paper *(a)* features the same $O(N \cdot logN)$ number of switching elements and the same $O(N)$ number of memory modules; while *(b)* it never uses more than $O(N)$ queues per memory module or switching element; and *(c)* it uses dynamic scheduling in the switching fabric, so as to achieve short delays. Backpressure is the mechanism to coordinate the distributed operation of dynamic scheduling.

[1]The middle stage of the ATLANTA chip set consists of multiple $\frac{N}{P} \times \frac{N}{P}$ bufferless crossbars, where P is the number of port interfaces connected to each input module.

1.2. Contributions of this Paper

In this paper, we extend the backpressure architecture from single-path fabrics (like banyans) to multi-path topologies, and specifically to the Benes network. This extension is non-trivial. In order for the Benes fabric to operate free of internal blocking, the cells of each flow must be routed over multiple paths, and must afterwards be properly resequenced, as reviewed in section 2.1. In order for backpressure to operate free of head-of-line-blocking effects, it must operate on a per-flow granularity, as reviewed in section 2.2. If these two requirements were combined in a naive way, $O(N^2)$ complexity would result for the switching elements in the middle stages of the Benes fabric. We show how to reduce this complexity down to $O(N)$, using appropriate flow merging techniques which minimally affect performance: see section 3.1. The resulting complexity of $O(N)$ is realistic for modern VLSI technology, because fabrics of size N in the order of a few thousand ports require on-chip buffer storage on the order of several thousand cells (several Mbits), which is feasible.

Multi-path cell distribution interacts with flow merging, and they both interact with the organization and placement of buffers; we show which organization is preferable, and we prove that it is deadlock-free (section 4). Finally, section 5 presents our simulation results, showing that *(a)* non-blocking operation with full output utilization is indeed achieved; *(b)* the delay-versus-load characteristics of this switching fabric under bursty traffic are comparable within a factor of 1.5 to those of ideal output queueing; *(c)* delay to uncongested outputs is minimally affected by the presence of congestion (over-subscribed outputs) elsewhere in the network; and *(d)* delay is not very sensitive to the specific multi-path cell distribution method within the class of methods we consider.

To the best of our knowledge, this is the first time that the application of per-flow backpressure to the Benes switching fabric is studied. Also, we are not aware of other studies of backpressure with multi-path cell routing in general. Multi-path cell routing has been studied before, e.g. [15] [16] [11], but not with backpressure.

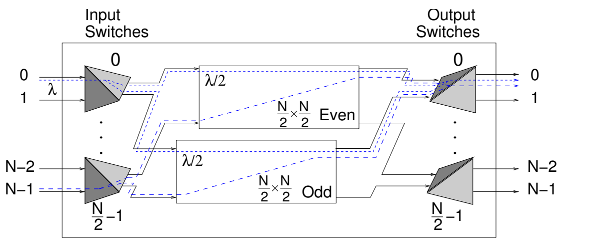

Fig. 1. Recursive construction of an $N \times N$ Benes network.

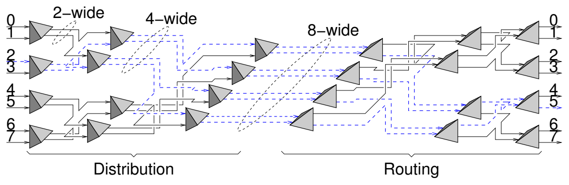

Fig. 2. 8×8 Benes network highlighting distribution and reconstruction of traffic $\lambda_{2,5}$.

2 . The Benes Fabric

This section reviews the two foundations of our design: the Benes fabric, and internal backpressure in switches.

2.1. Non-blocking Operation

The Benes network [3] can be constructed recursively, using *inverse multiplexing* [17] [15], as shown in fig. 1. The $N \times N$ Benes network consists of two $\frac{N}{2} \times \frac{N}{2}$ Benes subnetworks, $\frac{N}{2}$ switches of size 2×2 connected to the inputs of the two subnetworks, called the input switches in this paper, and $\frac{N}{2}$ switches of size 2×2 connected to the outputs of the two subnetworks, called the output switches, here.

Let $\lambda_{i,j}$ denote the traffic entering the network from input i and destined to output j. In order for the $N \times N$ network to be non-blocking, the 2×2 switch connected to input i must equally distribute $\lambda_{i,j}$ among its two outputs. The output switch that feeds output j receives $\frac{1}{2}\lambda_{i,j}$ on each of its inputs, reconstructs $\lambda_{i,j}$ and routes it to the appropriate output. Freedom from internal blocking results as follows. For any set of *feasible* rates $\lambda_{i,j}$ entering the $N \times N$ network (i.e. $\sum_{j=0}^{N-1} \lambda_{i,j} \leq 1,\ \forall i$) and leaving the $N \times N$ network (i.e. $\sum_{i=0}^{N-1} \lambda_{i,j} \leq 1,\ \forall j$), the rates entering and leaving each $\frac{N}{2} \times \frac{N}{2}$ subnetwork will also be feasible. Specifically, input k of either subnetwork will be receiving $\sum_{j=0}^{N-1} \frac{1}{2}\lambda_{2k,j} + \sum_{j=0}^{N-1} \frac{1}{2}\lambda_{2k+1,j}$ which is $\leq \frac{1}{2} + \frac{1}{2} = 1$ because of the above feasibility of the overall traffic. Symmetrically, the load of output m of either subnetwork will be $\sum_{i=0}^{N-1} \frac{1}{2}\lambda_{i,2m} + \sum_{i=0}^{N-1} \frac{1}{2}\lambda_{i,2m+1} \leq \frac{1}{2} + \frac{1}{2} = 1$. Assuming that each subnetwork is internally non-blocking, i.e. can route any such feasible traffic, it follows by recursion that the overall $N \times N$ network will also be internally non-blocking.

Unrolling the recursion in fig. 1, for N = 8, results in the topology shown in fig. 2. Traffic $\lambda_{i,j}$ goes through $\log N$ stages of distribution and $\log N$ corresponding stages of reconstruction. The figure also shows that an $N \times N$ Benes network can be constructed by placing two banyan networks back-to-back. The two banyans are called the *distribution* and the *routing* network, respectively [18], since the first distributes incoming traffic over the N links in the middle of the network – a virtual "wide" link of throughput N – and the second routes cells to the proper output link.

Non-blocking operation as above is based on (repeated) *inverse multiplexing* or *load distribution* in a balanced manner. A "poor man's" method for load distribution is to send all packets of "half" the microflows through one path, and all packets of the other half through the other path, e.g. using a pseudo-random hash function of the source-destination IP address pair to decide the path . This ensures that all packets of a given microflow follow the same route, and hence arrive in-order. The disadvantage of this method is that load distribution may not be balanced in the long run, and even worse on a short term basis, especially where the number of microflows is limited. Imbalanced load distribution will result in internal blocking in the Benes fabric, and thus we do not use this method. At the other end of the spectrum is a method for exact load distribution that resembles the *bit-sliced* processors of the 70's. Each cell is split in two units, of half the original cell (payload) size each, and each unit is sent in one of the two directions. This method is used in several commercial chip sets, but only with splitting degrees up to 8 and with carefully equalized delays through the paths [9]. This method is far from scalable, due to the fixed header and per-unit-processing overheads, and thus we do not use it.

To achieve balanced load distribution in the long run –even if not so on a very short term basis– while still operating at the cell level, a number of methods have been proposed: randomized [19], adaptive [15], per-flow round-robin cell distribution [14]. In all of these methods, cells of a given microflow are routed through either path, hence they may arrive out-of-order. For the switching fabric to preserve cell order within individual microflows, resequencers must exist at the points of path reconvergence [17] [16]. Resequencing is an important issue in our system, dealt with in sections 3.1 and 4.

2.2. Internal Backpressure Protocols

Switches with multistage buffering typically use *backpressure* feedback control between these stages, to avoid overflow of downstream buffers and to control individual flow rates when multiple flows merge into oversubscribed resources, thus enforcing quality-of-service (QoS) guarantees.

The simplest backpressure protocol is *stop-and-go*: the upstream stage maintains a single bit of state (in total or per-flow), specifying whether downstream transmission is currently enabled or disabled. A more sophisticated protocol, which economizes on buffer memory space, is the one using *credits*: the upstream stage maintains a credit counter (in total or per-flow), specifying how many cells is is allowed to transmit in the downstream direction before new credit is received via backpressure feedback signals. The buffer space needed is $\lambda \times RTT$ (in total or per-flow), where λ is the peak rate and RTT is the round-trip time. This paper uses credit-based backpressure.

Backpressure signals may refer to individual (micro) flows, or to flow aggregates, or indiscriminately to all traffic passing through a link. Indiscriminate backpressure leads to very poor QoS, because a single oversubscribed flow may stop the service to all other flows with which it shares a link or a buffer (this is analogous to head-of-line (HOL) blocking). Thus, *per-flow* or *virtual-channel* or *multilane* backpressure is needed. The number and definition of "flows" is a crucial parameter and affects cost –amount of state and granularity of feedback information– and QoS –degree of isolation among competing flows. When individual flow granularity is excessive, one can use a "compromise" solution or appropriate flow aggregation. Compromise backpressure protocols yield good performance in the usual cases, but perform badly in some worst cases; they include: wormhole virtual channels [20], a DEC proposal [21], Quantum Flow Control [22], and the ATLAS I multilane backpressure [23].

This paper is concerned with full-fledged per-flow backpressure, which ensures that even if all output ports but one are oversubscribed, traffic going to that one non-congested output will still enjoy delays comparable to those of an ideal output-queued switch. We obtain such strong QoS guarantees at a cost not worse than $O(N)$ per switching element, which is realistic for modern VLSI technology as explained in section 1.2.

The main tools used in this endeavor are the *merging of flows with common destination* and *hierarchical backpressure*. When multiple flows of a same priority level follow a common path to a common destination, they can be treated as a single, merged flow over the common path for purposes of buffer allocation and backpressure granularity. The reason is that cells of one flow will never need to overtake cells of another after the merge point. One (mild) disadvantage of such merging is its transient behavior when one of the flows goes from inactive to active: the "pipeline" ahead of the merge point has already been filled with cells of the other flows. Under weighted round robin (WRR) scheduling schemes, we run the danger that this pipeline empties at the rate corresponding to the weights of the old flows, while the recently activated flow may have much higher weight.

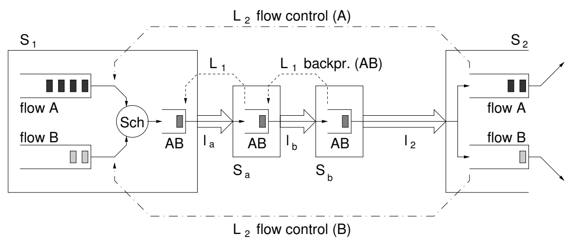

Fig. 3. Hierarchical flow control.

2.3. Hierarchical Backpressure

Hierarchical backpressure [24, section III-D], illustrated in fig. 3, can be used when multiple flows (preferably of the same priority level) share a common portion of their paths. This case differs from the previous case: here the flows are allowed to diverge after their common path. The flows can be treated as a single, merged flow over their common path, provided that a higher, second level of flow control feedback exists from the point of divergence (end of common path) to the point of flow merging. The common path behaves as a single, "virtual" link, and the second level flow control is over this virtual link.

In fig. 3, flows A and B have a common path, extending from switch (or stage) S_1 to S_2, through switches (or stages) S_a and S_b. Over the common path (links l_a, l_b, and l_2) the flows are treated as a merged flow "AB". At the merge point, a scheduler "Sch" decides which flow to get each next AB-cell from; it bases its decisions on *level 2 (L_2)* flow control feedback information that it receives from the divergence point, S_2. Level 1 (L_1) backpressure is at the granularity of the merged flow, AB. L_1 backpressure goes from S_b to S_a over l_b, and from S_a to S_1 over l_a. It is important to notice that no L_1 flow control is needed over link l_2: the flow into buffers A and B of switch S_2 is regulated via L_2 –not L_1– flow control.

An important application of hierarchical flow control, as drawn in fig. 3, is in large switches made using multistage fabrics and "line cards" that contain input buffers with virtual-output queues (VOQ). In figure 3, consider that S_1 is the ingress line card of (large) switch S_1, and S_2 is the ingress line card of the next downstream (large) switch S_2; S_a and S_b are the stages of the switching fabric of S_1. Links l_a and l_b are internal to the switching fabric, while link l_2 is a network (e.g. WAN) link. Under such a scenario, observe that L_1 backpressure is purely internal to the switching fabric, while L_2 is the network level flow control (hop-by-hop or end-to-end, credit- or rate-based). There is no L_1 backpressure on the network link l_2.

This is the model assumed in this paper: we deal exclusively with the flow control "L_1" *inside* the Benes fabric. We assume that this is of the credit-based backpressure type, independent of the type of flow control employed outside the fabric, in the overall network. Note from fig. 3 that L_1 backpressure operates on flow aggregates that consist of all network-wide (micro-)flows that share a common path (and priority level) within the switching fabric. Thus, in the rest of this paper, for an $N \times N$ fabric with pl priority levels, we only consider the $N^2 \times (pl)$ flows defined, each, by one specific fabric input port, i, one specific fabric output port, j, and one specific priority level. The merging of multiple external flows into one of our above internal flows, performed in the scheduler "Sch" in fig. 3, is performed in the ingress line card of the large switch, just before entry into the Benes fabric, and that is not a topic of the present paper.

3 . Switching Element Organization

In this section, we present flow merging schemes that reduce the $O(N^2)$ backpressure cost (per switching element) down to $O(N)$. Next, we describe the queues and the functionality inside the distribution and routing switching elements.

3.1. Flow Groups

As noted in sections 2.2 and 2.3, for an $N \times N$ Benes fabric, backpressure must operate at the granularity of the N^2 flows (per priority level) defined by all input-output pairs. In banyan fabrics, although the total number of flows is N^2, only N flows pass through any individual link in the fabric. In the Benes fabric, however, the traffic of every flow is distributed and sent over both "even" and "odd" subnetworks in fig. 1; consequently, all subnetworks, no matter how small, down to the individual switching elements in the core of the fabric, are traversed by N^2 flows (per priority level). We need to reduce this number, using the flow merging techniques of sections 2.2 and 2.3.

We first consider *per-output* merging of the flows destined to the same output port of the fabric. Fig. 4 shows the *flow groups* that internal backpressure must operate on; "01 $\rightarrow$ 0" denotes the merging of flows 0 $\rightarrow$ 0 and 1 $\rightarrow$ 0, and "0123 $\rightarrow$ 0" is the merging of flow groups 01 $\rightarrow$ 0 and 23 $\rightarrow$ 0. This example uses 2×2 switching elements. Each switching element of the distribution network (left half of the Benes fabric) merges, one-by-one,

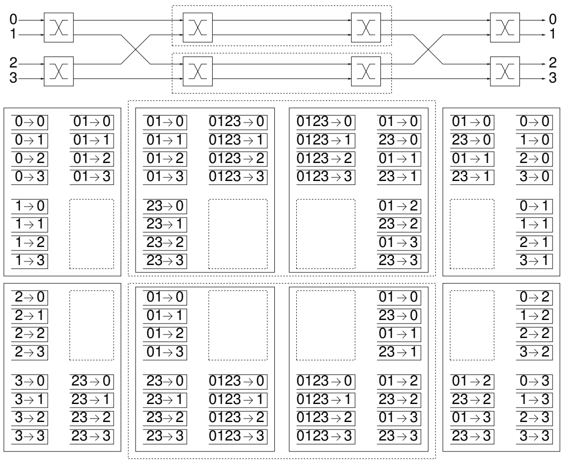

Fig. 4. A 4×4 fabric and the flow groups at the inputs and outputs of the switching elements for the per-output flow merging case.

the N flow groups entering through one of its inputs with the N flow groups entering through the other, and produces N merged flow groups; the merging factor is two-to-one. These switching elements also distribute the cells to both of their outputs, so the N merged flow groups appear on each of these outputs; fig. 4 shows one of these copies in full detail, and uses an empty box for the other. Hence, all links carry precisely N flow groups. (The two central stages of the fabric are shown separate for conceptual reasons, only; in reality, they are implemented as a single stage.)

In the routing network (right half of the Benes fabric), cells that had been distributed to the even and odd subnetworks must be resequenced. Resequencing, in output switches, must be performed separately for each flow in a merged flow group. The reason is that merged flow groups carry cells that were distributed at different input switches, independently of each other, before the merge points. Hence, merged flow groups from different inputs to a same output, must be split again in order for resequencing to work correctly.

Splitting of flow groups and cell resequencing can be performed progressively, per-stage, or cumulatively, in the very last stage of the fabric. In the latter case, we need not split flows within the routing banyan, thus, there would be $\frac{N}{2}, \ldots, 2, 1$ flows passing though the switching elements in the $\log_2 N$ stages of the routing banyan, respectively. However, each resequencer at the output ports of the fabric would then require N resequence buffers, one for each of the N (per-input) flows leading to that output, each of size $O(N)$. There is no reason to accumulate so much complexity in the last stage of the fabric, so we prefer the former solution –progressive flow group splitting and cell resequencing.

An alternative method to reduce the number of flows in the center of the fabric is *hierarchical flow merging*, which follows the recursive construction of the Benes network. Specifically, the switching elements at the edges of a $K \times K$ Benes subnetwork (refer to fig. 1) maintain state for the K flows destined to each output or originating from each input of that subnetwork. This method reduces the number of flows even more than per-output merging, down to $N/2^k$ in stage k. However, flows destined to different *final* destinations are merged together, so *hierarchical backpressure* (section 2.3) is needed, which is more complicated than plain backpressure to implement. This scheme needs a total of $\log N$ levels of backpressure. Although hop-by-hop backpressure is no longer needed in the routing network, the flow control feedback delay is up to $2 \log N$ for the upper flow control level, hence buffers of size up to $2 \log N$ are needed for full link utilization, which is undesirable.

In conclusion, per-output flow merging with per-stage resequencing is much simpler to implement and has a uniform implementation cost of $O(N)$ per switching element, across all stages of the switching fabric, so we use this architecture in the rest of the paper. Lucent's ATLANTA chip set [8] also uses per-output flow merging and cell distribution, but avoids resequencing because the middle stage consists of $\frac{N}{P} \times \frac{N}{P}$ *bufferless* crossbars, thus, it does not reorder cells. However, it is not clear how to scale to larger port numbers and larger port rates and still use only three stages.

3.2. Logical Buffer Organization

Figure 5 shows the preferred logical buffer organization of the distribution and routing switching elements, along with the active components needed. We follow the flow merging and cell resequencing architecture that was chosen above. The flows from inputs 0 and 1 to four different fabric outputs are shown in the left (distribution) switching element, along with the flows to outputs 0 and 1 from four different fabric inputs in the right (routing) switching element. The FIFO's shown are *logical* queues, containing *references* to cells; the actual cells do not move inside the switching element.

Distribution switching elements must perform flow merging and cell distribution; they can perform these tasks in either order. Routing switching elements must perform cell resequencing and flow splitting in the proper, corresponding order. Cell distribution can be performed in a number of ways; as discussed in section 4, it relies on per-flow state and aims to optimize per-flow criteria. Flow merging before cell distribution reduces the number of flows seen by cell distribution. The smaller the number of flows, the easier it becomes to coordinate the per-flow (local) decisions so as to optimize global criteria; also, buffer space gets reduced, as explained later in this section. Thus, we choose this arrangement, as shown in fig. 5.

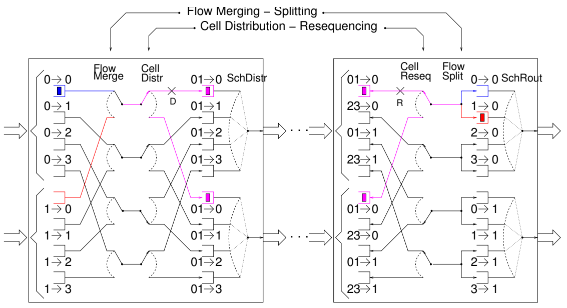

Fig. 5. Logical buffer organization of a distribution and the corresponding routing switching element.

At the inputs of the switching elements, buffers are needed per input port and per flow group, because credits for that buffer space and at that granularity must be sent to each upstream neighbor. Besides these input buffers, it is advantageous or necessary to also have output buffers, as shown in fig. 5. The chosen arrangement requires $2 \times P \times N$ FIFO queues per distribution switching element. If the distribution switching elements performed cell distribution before flow merging, then, each of them would need $P^2 \times N$ FIFO queues.

In the distribution switching elements (left half of the network), it is advantageous to have output buffers *(a)* in order for output schedulers to operate independently, and *(b)* for efficiency in some distribution circumstances, as explained below. Suppose that output buffers did not exist. First, assume that input buffer $0 \rightarrow 0$ contains a cell while input buffer $1 \rightarrow 0$ is empty (as in fig. 5), and that the cell distribution algorithm allows the cell to depart in either direction. Then, up to one but not both output schedulers of this switching element would be allowed to choose flow group $01 \rightarrow 0$ for service; hence, the two schedulers would not be able to operate in parallel. Next, assume that both input buffers $0 \rightarrow 0$ and $1 \rightarrow 0$ contain cells, and assume that the cell distribution algorithm dictates that the next-in-order cell of flow group $01 \rightarrow 0$ must depart through the top output of the switching element. Until the top-output scheduler is able to serve this next-in-order cell, it would be very hard for the bottom-output scheduler to serve flow group $01 \rightarrow 0$, although two cells exist in this flow group, because we don't quite know which cell is the second-next in order.

In the routing switching elements (right half of the network), input buffers are needed for the same reason as for the distribution switching elements, unless we know where to expect the next cell from in which case we only need

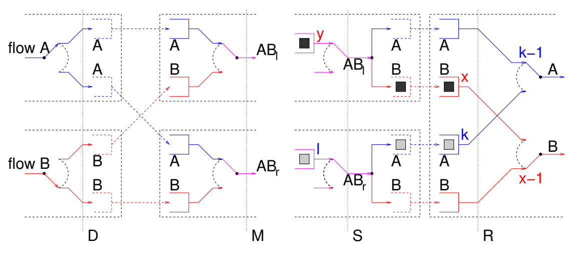

Fig. 6. Deadlock situation when flow merging precedes cell distribution. The participating switching elements are shown with dashed lines and the participating flows are indicated with A, B and AB. The numbers by the FIFO buffers denote the sequence number of the cell at the head of the buffer.

one buffer slot and the credit for that buffer is sent to the upstream node from which the next cell will arrive. Each output buffer, together with its input counterpart in the downstream neighbor switch, forms a double-depth buffer pipe, which is needed for deadlock-free operation of cell resequencing under the preferred distribution methods, as will be seen in section 4. However, output buffers are not necessary, they could be dropped by making, at the same time, the input buffers of greater depth.

4 . Freedom from Deadlock

The Benes fabric with finite buffers, internal backpressure, flow merging, and resequencing is a distributed system with finite resources and resource sharing. In such a system, we have to make sure that deadlock situations either do not occur, or if they do occur, the system detects and resolves them. In this section, we show that for a wide and interesting class of cells distribution methods, a deadlock situation cannot arise.

We consider cell distribution methods with a maximum per-flow *imbalance* of 1: at any time, the total number of cells belonging to some flow that have been forwarded through any two paths available to that flow differs by at most 1. At the other end of the two paths, resequencing "consumes" cells in order; it follows that, for such distribution methods, the number of cells buffered along the two paths can differ by at most 2. We see that these distribution methods equalize the loads on the two paths. Per-flow round-robin cell distribution is such a method.

Figure 6 shows how a deadlock could arise in our switching elements. Figure 7 shows a simplified but *equivalent* view of the deadlock situation: consecutive FIFO buffer dedicated to the same flow have been merged into a single FIFO buffer with depth equal to the sum of the depths of the individual FIFO buffers. Let c_f^s denote the cell of flow "f" with sequence number "s", B_f^s denote the buffer slot occupied by cell c_f^s, and R_f denote the resequencer of flow "f". The deadlock situation is the following (see fig. 8 for the resource allocation graph): *(a)* the resequencer of flow

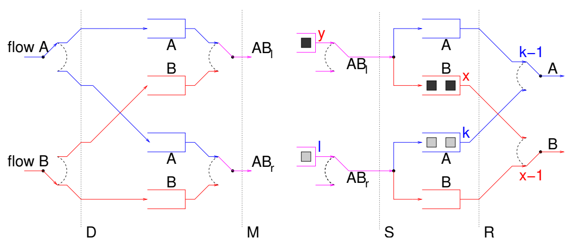

Fig. 7. Simplified but equivalent view of the deadlock situation shown in fig. 6.

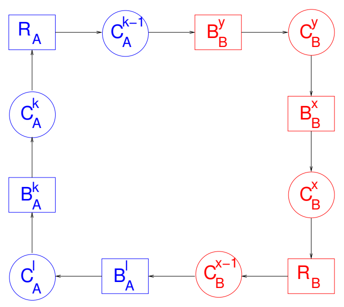

Fig. 8. The resource allocation graph for the deadlock situation. Circles represent cells, while rectangles represent resources which can be either buffer slots or resequencers.

A is waiting for cell c_A^{k-1}, *(b)* cell c_A^{k-1} is somewhere in the fabric behind cell c_B^y and it needs a buffer (B_B^y) or resequencer (R_{AB_l}) held by cell c_B^y in order to move forward – note that cell c_A^{k-1} could not be on the same path with cell c_A^k since they belong to the same flow, thus, the Benes fabric would not reorder the two cells, *(c)* cell c_B^y needs a buffer held by cell c_B^x in order to move forward, *(d)* cell c_B^x is waiting to be resequenced by R_B, and so on, so forth until the cycle closes to R_A.

4.1. Basic Case

Let b_D denote the size of the distribution FIFO buffers, and b_R denote the size of the routing FIFO buffers shown in fig. 7.

Theorem 1 *If no cells are lost,* $b_D = 1$*,* $b_R = 2$ *and cell forwarding is subject to hop-by-hop credit-based flow control, then any cell distribution method with maximum per-flow imbalance of 1 is deadlock-free.*

Proof. Let L_f^s denote the time slot at which cell c_f^s crosses line "L" in fig. 7. The cell distribution method and the fabric operation impose limitations on the set of cells that can be active on the paths available for flows A and B, and the ordering between various values of L_f^s for lines D,M,S,R, respectively. With regard to the set of active cells, there are four cases for flow A shown in fig. 9, the cases for flow B are analogous. The ordering relations are of four types and are listed below:

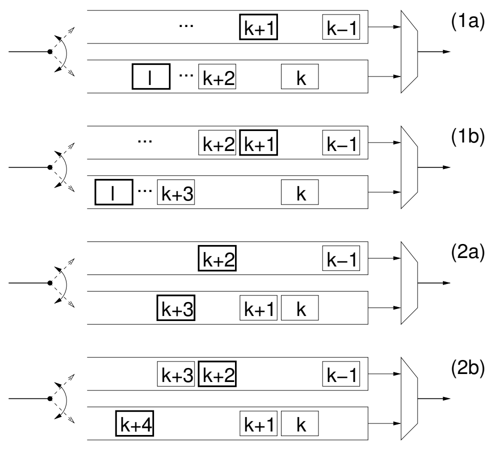

Fig. 9. Active cells allowed by cell distribution methods with maximum per-flow imbalance of 1.

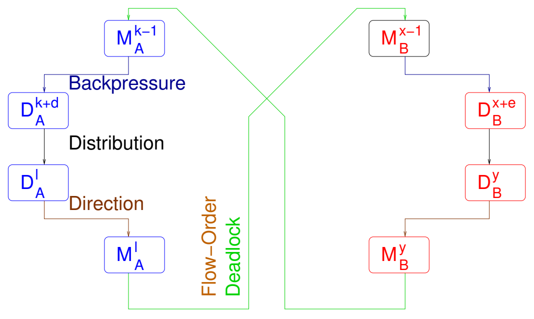

Fig. 10. The time-order graph. The nodes of the graph represent events of the form L^s_f and the arcs represent ordering relations between the events. The direction of an arc is from the older to the newer event.

- *Direction:* $(D^i_f < M^i_f < S^i_f)$
- *Distribution:* $(D^i_f < D^{i+1}_f)$
- *Backpressure:* $(M^i_f < D^j_f)$ *for every cells* c^i_f *and* c^j_f *with* $i < j$ *which were both forwarded through the same path*
- *Flow-Order:* $(M^i_f < M^j_f \iff S^i_f < S^j_f)$ *for every cells* c^i_f *and* c^j_f *which were both forwarded through the same path*

Using the above relations, we can partially construct the time-order graph shown in fig. 10, specifically relations $(M^{k-1}_A < M^l_A)$ and $(M^{x-1}_B < M^y_B)$. Cell c^{k+d}_A refers to the *first* cell of flow A after cell c^{k-1}_A on the same path with cell c^{k-1}_A. Sequence number $k+d$ corresponds to either $k+1$, cases *(1a)* and *(1b)*, or $k+2$, cases *(2a)* and *(2b)*. Cell c^l_A refers to the *second* cell of flow A after cell c^k_A on the same path with cell c^k_A. Sequence number l also differs in each of the four cases of fig. 9. The important property is that l is greater than $k+d$, and this property can only be guaranteed in all of the four cases only if the size of the routing FIFO buffers is 2 slots. Similarly for flow B.

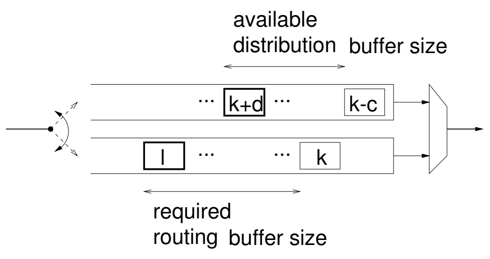

Fig. 11. Required size of routing FIFO buffers in order to ensure deadlock free operation.

If we assume that a deadlock arises, then the following ordering relations also hold: $(M^y_B < M^{k-1}_A)$ and $(M^l_A < M^{x-1}_B)$. The deadlock ordering relations cause a cycle in the time-order graph, which is a contradiction. Thus, a deadlock situation cannot arise.

◁

Note that the proof does not assume any properties for the scheduling discipline at the flow merging point. However, it assumes that no cells are lost due to electrical noise within the fabric. This is definitely an unrealistic assumption and a real system would have to employ robust resequencing protocols as the ones described in [16]. The proof of the basic case extends easily for the case of more than two participating flows. With regard to per-flow round-robin cell distribution, note that it remains deadlock free even with routing buffers of size 1, since it only allows case (1a) of fig. 9 to arise.

4.2. *Extensions*

The proof continues to hold in the general cases of *(a)* distribution FIFO buffers of size b_D slots each, where $b_D > 1$ and *(b)* switching elements of size $P \times P$, where $P > 2$. The important observation is that the ordering relations $M^{k-1}_A < M^l_A$, and $M^{x-1}_B < M^y_B$, shown in fig. 10, can be guaranteed *independently* for each flow, and depend only the properties of the cell distribution method and the available distribution buffer size b_D.

Consider fig. 11 and assume that cell c^{k-c}_A is the next cell to be resequenced, cell c^{k+d}_A is the *first* cell of flow A after cell c^{k-c}_A on the same path with cell c^{k-c}_A, and cell c^l_A is the cell of flow A with the *smallest* sequence number *greater* than $k+d$ on the same path with cell c^k_A. The relation $M^{k-c}_A < M^l_A$ holds in this case provided that we choose b_R large enough so that sequence number l is *greater* than sequence number $k+d$ in all cases allowed by the cell distribution method for the given buffer size b_D.

To sum up, for a given cell distribution method and distribution buffer size b_D, we can choose b_R so that the

Benes fabric is deadlock free. In the special case of maximum per-flow imbalance of 1 and $b_D = 2$, the required b_R is 3. With regard to per-flow round-robin cell distribution, the required b_R is equal to b_D for any value of b_D.

5 . Simulation Results

A simulation model operating at the granularity of cell times was developed in order to verify the design and evaluate its performance under various traffic patterns and for various switch sizes, and in order to evaluate cell distribution and resequencing methods. In the simulation model, the cell-credit round-trip time is 1 cell time, and each buffer shown in fig. 5 has a size of 1 cell, except for the input buffers of the routing switching elements which have a size of 2 for a specific cell distribution method.

We simulated the switch under smooth, bursty, and hotspot traffic. Smooth traffic consisted of Bernoulli arrivals with uniformly distributed destinations. For bursty traffic, each source alternatingly produces a burst of cells (all with the same destination) followed by an idle period of empty cells; the bursts and idle periods contain a *geometrically distributed* number of cells. The reported results use bursty/12 traffic, where the mean burst size is 12 cells; this is close to one of the modes of IP traffic size distribution (assuming 48-byte cell payload). Under hotspot traffic, each destination belonging to a designated set of "hot spots" receives (smooth or bursty) traffic at 100% collective load, uniformly from all sources; the rest of the destinations receive smooth or bursty traffic as above. The reported results use hotspot/4 traffic, where the four hotspots are ports 0, 1, 2, and 3.

The delay reported is the average over all cells of the cell's exit time, minus the cell's birth time, *minus the fabric length* (number of stages); for example, most of the reported results are for a 64×64 fabric made of 4×4 switching elements, hence the fabric has $2 \cdot \log_4 64 = 2 \cdot 3 = 6$ stages, and the number subtracted is 6 cell times. In our simulation model, it takes 1 cell time for a cell to traverse 1 stage of the fabric, in an otherwise idle system, with the cell moving at top speed. Thus, by subtracting the fabric length from the actual delay, we report the sum of all queueing delays for the cells. In all of the reported results, the duration of the simulation is 200,000 cell times and collection of statistics starts after the first 40,000 cell times. We ran each simulation 10 times and then computed the sample mean and the corresponding confidence interval for the measured delay. With regard to average delay, the 95% confidence intervals were well below 5% in all cases but one case for load equal to 99% where the confidence interval was 7.1%.

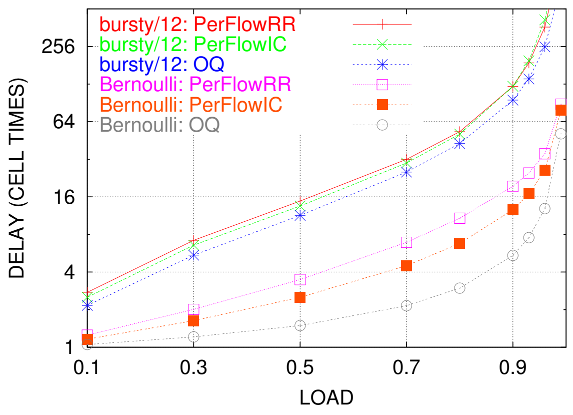

Fig. 12. Delay versus load for uniform destinations; 64×64 fabric made of 4×4 elements; upper curves: bursty/12 traffic; lower curves: Bernoulli traffic; ideal output queueing (OQ) also shown for comparison.

As a means to get an indication regarding the lack of internal blocking, we also simulated the 64×64 fabric under the following artificial load. In each and every cell time, a randomly-selected full permutation was presented to the input of the switch; that is, all inputs were continuously loaded at precisely 100%, while the overall load presented to the fabric was *feasible*, in the sense of section 2.1, during each and every cell time. After one million simulation cell times, there were virtually no cells queued at the inputs: most of the VOQ's were empty, while a few others contained 1 or 2 cells each.

5.1. Cell Distribution Methods and OQ Comparison

We experimented with two cell distribution methods, called *PerFlowRR* and *PerFlowIC*, on a 64×64 Benes fabric made of 4×4 switching elements. PerFlowRR is per-flow round-robin cell distribution, where the per-flow distribution pointers are randomly initialized. PerFlowIC (standing for per-flow imbalance count) chooses the port for forwarding the next cell as follows: among the set of ports that have received the least number of cells of this flow up to now, choose the port that currently has the least number of *ready cells*; ready cells are the cells (of any flow group) that are queued at this port and that have an available downstream credit. Both methods have a maximum per-flow imbalance of 1, and, in the long run, send the same number of cells in each path; PerFlowIC, though, is more flexible every time the imbalance count returns to 0. We also performed simulations with larger buffer sizes, up to 4, which allow more "slack" in the two paths, and found that performance is *insensitive* to this parameter. The results are shown in fig. 12, for uniformly destined traffic, and in fig. 13, for traffic in the presence of hot spots.

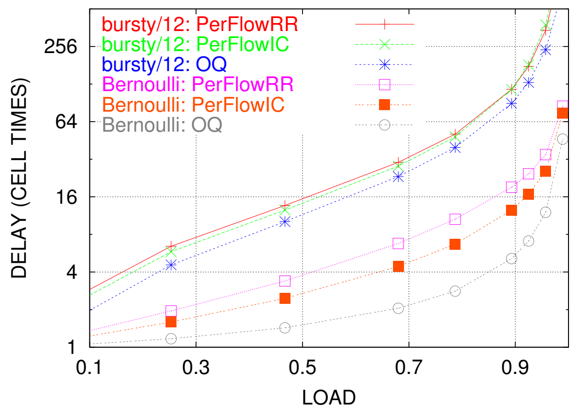

Fig. 13. Delay of non-hotspot destinations in the presence of hotspot/4 traffic; horizontal axis is the load to non-hotspot outputs; other parameters as in fig. 12.

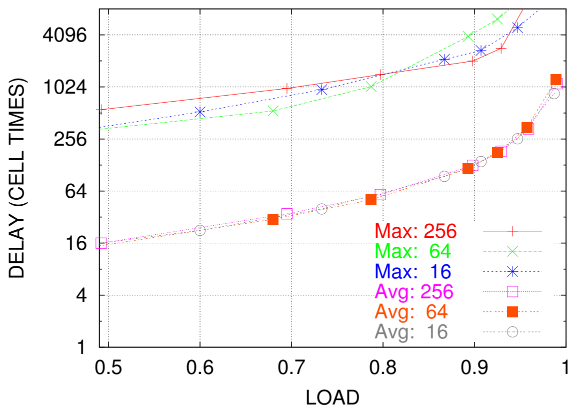

Fig. 14. Performance for various fabric sizes, 16×16 to 256×256: average delay and maximum delay versus load, under bursty traffic in the presence of hot spots. The results are for the PerFlowRR cell distribution method.

Under smooth (Bernoulli) traffic, the cell distribution method does make some difference: imbalance count (PerFlowIC) yields 30% to 60% lower delay when compared to round-robin distribution (PerFlowRR). The difference is more pronounced for medium loads, and less pronounced for light or heavy loads. The presence or absence of hotspot traffic does not affect this aspect of the results. Under *bursty* traffic, though, the cell distribution method makes virtually *no difference*. This must be due to the large number of back-to-back cells in the same flow: in this case, PerFlowIC becomes similar to PerFlowRR not only in the long but also in the short term.

By comparing the delays in fig. 13 to those in fig. 12, we notice that they are almost identical, which shows that non-hotspot traffic stays virtually *unaffected* by the presence of hot spots in the network, thus proving the *excellent QoS properties* of this switch. Not shown in the plots is the throughput (utilization) of the hotspot destinations (remember that the load offered to them is 100%). Under smooth traffic this output utilization was consistently over 99%; under bursty traffic, it ranged from 92% to 98%.

Figures 13 and 12 also show, for comparison, the delay of the ideal output-queued (OQ) switch under each traffic load; in every triplet of curves, output queueing is the lower of the three curves. We see that, under bursty traffic, the Benes fabric has only 25% to 50% worse delay when compared to ideal output queueing. Under smooth traffic, the switching fabric's delay is longer by a factor of 1.6 to 4, the difference being less pronounced for light load and more pronounced around 80% load.

5.2. Fabric Size Dependence of Performance

One of the advantages of the proposed architecture is than it can scale to very large sizes. It is important for the performance of the fabric not to degrade with increasing size. We experimented with fabrics of up to 256 ports. We used the more "interesting" of the previous traffic patterns, bursty/12 arrivals with hotspot/4 destinations.

The results are plotted in fig. 14, and they show that maximum cell delay generally increases with increasing fabric size, roughly by about 25% to 75% when the fabric size quadruples. However, *average* cell delay remains virtually *unaffected* by fabric size.

5.3. Alternative Cell Resequencing Methods

As discussed in section 3.1, cell resequencing can be performed progressively, "PerStage", or cumulatively, in the very last stage of the fabric ("FinalOut"). From the point of view of implementation, per-stage resequencing is simpler and less expensive than FinalOut, but the question remained regarding performance: it appears that FinalOut lets cells go faster through the routing network, and thus may lead to lower delays. In reality, things are the other way around!

Figure 15 shows the average delay under the two resequencing methods; input traffic is bursty/12 and hotspot/4, as in section 5.2. For the "FinalOut" method, we show separately the delay for the cells to get through the fabric, without yet being resequenced ("FinalOut Fabric"), and separately their total delay, including the resequencing process in the very last stage of the fabric ("FinalOut Total"). Interestingly, although cells do indeed get a bit faster through the fabric, as compared to the case where per-stage resequencing delays them in the routing network, when the delay of FinalOut resequencing is added, the overall delay of FinalOut is worse.

We see that letting some cells get quickly through the fabric, ahead of their order, without per-stage resequencing, appears to consume such fabric resources that, overall, it harms other cells more than it benefits the early-out cells.

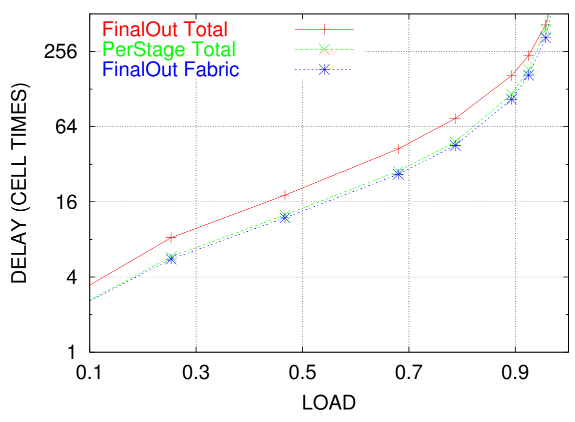

Fig. 15. Average delay under different resequencing methods; bursty traffic in the presence of hot spots. The results are for the PerFlowIC cell distribution method.

We conclude that *per-stage resequencing is strictly better* than cumulative resequencing in the very last stage of the fabric, both from the point of view of implementation cost and complexity as well as from the point of view of performance.

6 . Conclusions

We showed how to efficiently scale packet switches to very large numbers of ports, while maintaining non-blocking operation and high quality of service. This can be done using the Benes network, the lowest-cost switching fabric that is free of internal blocking. Large buffer memories are only needed at the inputs of the system, to implement virtual output queues (VOQ); their number scales linearly with system size, the number of queues in each memory also scales linearly, while their throughput stays fixed. Internal backpressure is used in the Benes fabric, in order to provide: *(a)* low cost switching elements, since they only need on-chip buffer memory; *(b)* zero cell loss in the switching fabric, although buffer memories are small; *(c)* low system cost, since the fabric needs no internal speedup; *(d)* low system cost, since the fabric does not need redundant paths to handle cell conflicts using deflection routing; *(e)* low system cost, since no global scheduler is needed, and all scheduling and coordination is distributed; and *(f)* high system performance and high quality of service, even though system cost is kept low as detailed above.

To achieve all these, we had to extend the known per-flow backpressure architecture so as to make it applicable to multipath routing (inverse multiplexing) and cell resequencing, while keeping its cost manageable. We achieved this using an appropriate flow merging scheme that keeps the cost of backpressure down to $O(N)$ per switching element. We proved freedom from deadlock for a wide class of multipath cell distribution algorithms. Finally, using a cell-time-accurate simulator, *(a)* we verified operation free of internal blocking; *(b)* we showed that per-stage resequencing is preferable; *(c)* we found that cell distribution based on imbalance counts leads to lower delays than round-robin distribution, but under bursty traffic this difference becomes negligible; *(d)* we noticed that delay under bursty traffic is only 25 to 50 % higher than ideal output queueing; and *(e)* we showed that the delay of well-behaved flows remains unaffected by the presence of congested traffic to oversubscribed output ports, thus proving the excellent quality of service properties of the system.

References

[1] M. Marcus, "The Theory of Connecting Networks and their Complexity: a Review," *IEEE Proceedings*, vol. 65, no. 9, pp. 1263–1271, Sept. 1977.

[2] C.-L. Wu and T.-Y. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. on Computers*, vol. 29, no. 8, pp. 694–702, Aug. 1980.

[3] V. Benes, "Optimal Rearrangeable Multistage Connecting Networks," *Bell Systems Technical Journal*, vol. 43, no. 7, pp. 1641–1656, July 1964.

[4] K. Batcher, "Sorting Networks and their Applications," in *AFIPS Proc. 1968 Spring Joint Computer Conf.*, 1968, vol. 32, pp. 307–314.

[5] A. Huang and S. Knauer, "Starlite: A Wideband Digital Switch," in *Proc. IEEE GLOBECOM '84 Conf., Atlanta GA USA*, Dec. 1984, pp. 121–125.

[6] J. Giacopelli, J. Hickey, W. Marcus, W. Sincoskie, and M. Littlewood, "Sunshine: a High Performance Self-Routing Broadband Packet Switch Architecture," *IEEE-JSAC*, vol. 9, no. 8, pp. 1289–1298, Oct. 1991.

[7] G. Kornaros, D. Pnevmatikatos, P. Vatsolaki, G. Kalokerinos, C. Xanthaki, D. Mavroidis, D. Serpanos, and M. Katevenis, "ATLAS I: Implementing a Single-Chip ATM Switch with Backpressure," *IEEE Micro*, vol. 19, no. 1, pp. 30–41, Jan. 1999, http://archvlsi.ics.forth.gr/atlasI/hoti98/.

[8] F. Chiussi, J. Kneuer, and V. Kumar, "Low-Cost Scalable Switching Solutions for Broadband Networking: The ATLANTA Architecture and Chip Set," *IEEE Communications Magazine*, vol. 35, no. 12, pp. 44–53, Dec. 1997.

[9] "IBM PowerPRS Q-64G Packet Routing Switch Datasheet," Dec. 2001, http://www.ibm.com/chips/techlib/techlib.nsf/products/-PowerPRS_Q-64G_Packet_Routing_Switch.

[10] "ETT1 Chip Set Datasheet," Mar. 2002, http://www.pmc-sierra.com/products/details/pm9312/index.html.

[11] Sundar Iyer, Amr A. Awadallah, and Nick McKeown, "Analysis of a Packet Switch with Memories Running Slower than the Line-Rate," in *IEEE INFOCOM*, Mar. 2000, http://klamath.stanford.edu/~sundaes/Papers/infocom2000.pdf.

[12] D. A. Khotimsky and S. Krishnan, "Stability Analysis of a Parallel Packet Switch with Bufferless Input Demultiplexors," in *ICC 2001*, June 2001.

[13] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load Balanced Birkhoff-von Neumann Switches, Part I: One-stage Buffering," *IEEE HPSR Conf.*, May 2001, http://www.ee.nthu.edu.tw/cschang/-PartI.ps.

[14] Sundar Iyer and Nick McKeown, "Making Parallel Packet Switches Practical," in *IEEE INFOCOM*, Mar. 2001, http://klamath.stanford.edu/~sundaes/Papers/infocom2001.pdf.

[15] F. Chiussi, D. Khotimsky, and S. Krishnan, "Generalized Inverse Multiplexing for Switched ATM Connections," in *Proc. IEEE GLOBECOM Conf., Australia*, Nov. 1998, pp. 3134–3140, http://www.bell-labs.com/org/113480/Papers/fabio-globecom98B.ps.

[16] D. Khotimsky, "A Packet Resequencing Protocol for Fault-tolerant Multipath Transmission with Non-Uniform Traffic Splitting," in *Proc. IEEE GLOBECOM Conf., Brasil*, Dec. 1999, pp. 1283–1289, http://www.bell-labs.com/org/113480/Papers/-dkh-globecom99.ps.

[17] J. Duncanson, "Inverse Multiplexing," *IEEE Communications Magazine*, vol. 32, no. 4, pp. 34–41, Apr. 1994.

[18] J. Turner, "Design of a Broadcast Packet Switching Network," *IEEE Transactions on Communications*, vol. 36, no. 6, pp. 734–743, June 1988.

[19] L. G. Valiant and G. J. Brebner, "Universal Schemes for Parallel Communication," in *ACM STOC*, 1981, pp. 263–277.

[20] William J. Dally, "Virtual Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, Mar. 1992.

[21] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 4, pp. 642–650, May 1995.

[22] Quantum Flow Control Alliance, "Quantum Flow Control: A cell-relay protocol supporting an Available Bit Rate Service," July 1995, version 2.0.

[23] M. Katevenis, D. Serpanos, and E. Spyridakis, "Credit-Flow-Controlled ATM for MP Interconnection: the ATLAS I Single-Chip ATM Switch," in *HPCA*, Feb. 1998, pp. 47–56, http://archvlsi.ics.forth.gr/atlasI/atlasI_hpca98.ps.gz.

[24] M. Katevenis, "Fast Switching and Fair Control of Congested Flow in Broad-Band Networks," *IEEE Journal on Selected Areas in Communication*, vol. 5, no. 8, pp. 1315–1326, Oct. 1987.