Variable Packet Size Buffered Crossbar (CICQ) Switches

*Manolis Katevenis, Georgios Passas, Dimitrios Simos, Ioannis Papaefstathiou, and Nikos Chrysos**

Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH)
ICS-FORTH, P.O. Box 1385, Vassilika Vouton, Heraklion, Crete, GR-711-10 Greece
`http://archvlsi.ics.forth.gr/bufxbar/ – {katevenis,nchrysos}@ics.forth.gr`

***Abstract*— One of the most widely used architectures for packet switches is the crossbar. A special version of it is the buffered crossbar, where small buffers are associated with the crosspoints; this simplifies scheduling and improves its efficiency and QoS capabilities to the point where the switch needs no internal speedup. Furthermore, by supporting variable length packets throughout a buffered crossbar: (a) there is no need for segmentation and reassembly (SAR) circuits; (b) no speedup is necessary to support SAR; and (c) synchronization between the input and output clock domains is simplified. In turn, the lack of SAR and speedup mean that no output queues are needed, either. In this paper we present an architecture, a chip layout and cost analysis, and a performance evaluation of such a 300 Gbps buffered crossbar operating on variable-size packets. The proposed organization is simple yet powerful, can be implemented using modern technology, and, as the performance results demonstrate, it clearly outperforms unbuffered crossbars.**

1. Introduction

The crossbar is the simplest and most popular organization for high performance (internally non-blocking) switches; it is also the building block for switching fabrics. Most of the crossbars considered in the literature, and the most widely known crossbars in commercial products, are *unbuffered*, as shown in figure 1(a). However, *buffered crossbars*, as in figure 1(b), have significant advantages. One advantage that has received little attention up to now is that buffered crossbars can operate directly on *variable-size* packets, i.e. without requiring segmentation and reassembly (SAR). Coupled with the simplicity and effectiveness of scheduling, this eliminates the need for crossbar speedup. The lack of speedup and of packet reassembly, in turn, remove the requirement for output queues and egress buffer memories. Speedup and buffering are major contributors to cost, hence variable-packet-size buffered crossbars have the potential of significantly lowering the cost of packet switches and routers. This paper reports a novel organization together with cost, and performance figures for such a prototype crossbar.

Unbuffered crossbars tolerate no output conflicts: information entering on different inputs must be destined to different outputs *at all times*. To operate efficiently under this constraint, all crosspoint configurations (all control signals in fig. 1(a)) have to change *in synchrony*; input queues have to be organized per-output (VOQ - virtual output queues); and the crossbar scheduler has to solve a bipartite graph matching problem [1] [2] [3]. Synchronous operation introduces at least two overheads: *(i)* variable-size packets have to be segmented into fixed-size cells before entering the crossbar; and *(ii)* cells entering a crossbar chip through links operating in different clock domains have to be synchronized before being switched. Crossbar schedulers, on the other hand, are the source of at least two inefficiencies: *(i)* when the cell time is short, they cannot practically achieve both high throughput and low latency; and *(ii)* it is very hard for them to provide *weighted* fair queueing (WFQ) quality of service (QoS) [4].

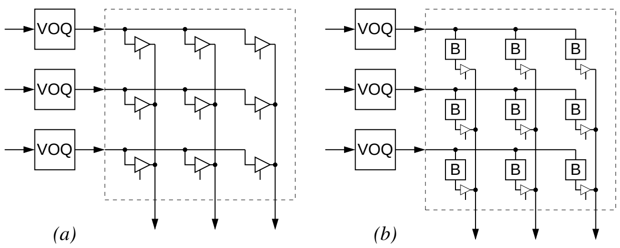

Fig. 1. 3×3 crossbar example: (a) unbuffered, (b) buffered

To cope with the segmentation overhead and the scheduler inefficiencies of unbuffered crossbars, switches and routers use internal *speedup* [5] –often by a factor of *two* to *three* in commercial products[1]. This speedup is very expensive: today, the crossbar chip power consumption is often the limiting factor for the aggregate performance of the system, and power consumption translates directly into (mostly I/O) throughput. Thus, a router that uses e.g. a speedup of two usually ends up providing only half of the aggregate line throughput that it could otherwise offer. Additionally, the use of speedup brings the need for output queues (CIOQ - combined input-output queueing); their size can grow considerably, hence the egress path on the line cards has to provide expensive, off-chip buffer memory.

Buffered crossbars suffer none of the above overheads or inefficiencies. Their properties stem from their capability to tolerate output conflicts: information entering on different inputs can be destined to any output, because it does not have to be delivered to that output right away –it can be buffered at the relevant crosspoints. This greatly simplifies

* The authors are also with the Dept. of Computer Science, University of Crete, Heraklion, Crete, Greece.

[1] For example, to handle a sequence of 65-byte back-to-back packets, a 64-byte-cell system would need a speedup of at least $128/65 \approx 2$.

scheduling: input transmissions can be decided *independent* of each other and independent of output transmissions. The N^2 buffers of an $N \times N$ crossbar would be too expensive if they had to be large enough for all packets to be queued in them. Instead, as in figure 1(b), it is better to provide small crosspoint buffers "backed up" by VOQ's in large input buffers –hence the name *Combined Input-Crosspoint Queueing (CICQ)*; backpressure control (not shown in the figure) ensures that the small buffers do not overflow. Small buffers and feedback control provide *coupling* among the N –otherwise independent– input schedulers and the N output schedulers: although *short-term* output conflicts are tolerated, the traffic pattern has to be *feasible* (admissible) in the long run. Hence, essentially, crosspoint buffering allows scheduling to solve the bipartite graph matching problem in an approximate and long-term way, rather than the exact and short-term solution required by unbuffered crossbars.

Scheduler independence removes the requirement for synchronized decisions, thus also removing the need for fixed-size cells and synchronization to a common clock. Additionally, the loosely-coupled input and output schedulers are able to find very efficient long-term solutions to the crossbar scheduling problem, with capability for advanced QoS, *without* requiring speedup [6] [7] [8] [9]. These facts allow significant cost reductions, since they eliminate the need for speedup and egress buffering.[2] Although advantageous, the buffered crossbar architecture was not very popular in products, due to the difficulty, in the past, to integrate large amounts of memory on the crossbar chip. With the progress of semiconductor technology, however, we are today at the point where buffer space up to 2 to 4 MegaBytes, can easily be placed on an ordinary chip. Thus, we consider buffered crossbar as the architecture of choice for the switching components of the coming years, for port counts from less than 32 to 128.

This paper studies buffered crossbars that operate directly on *variable* size packets. Although a number of previous studies explored fixed-size-cell buffered crossbars, very little work has been done up to now on variable packet size operation. We review this previous work and point out the novelty of our results in section 2. Section 3 discusses the organization and operation of variable-packet-size buffered crossbars, and gives hardware cost metrics for them. In particular, we discuss crosspoint queue organization, inter-clock domain communication, cut-through operation, scheduler placement, and backpressure format; then, we give gate count, silicon area, and power consumption figures, that have been taken out from a designed prototype switch. Section 4 evaluates the performance of the crossbars under consideration, using simulation; input loads include realistic network traffic, as well as some worst-case scenarios. We show that a reasonable crosspoint buffer size is approximately one maximum-size packet plus one round-trip-time (RTT) window, and we demonstrate the superior performance of buffered crossbars without speedup relative to unbuffered ones with considerable speedup.

[2]Egress buffering will still be needed for the ports that drive multiple output sub-ports each.

2. Previous Work

Buffered crossbar proposals date at least as far back as 1987: Nojima e.a. [10] described a "bus matrix" switch with buffers only at the crosspoints (no input buffers), operating on variable-size packets; we [11] proposed a switch with small crosspoint buffers, large input buffers, and backpressure between them. Recently, with the availability of technology for single-chip buffered crossbars, a number of groups studied *fixed-size-cell* buffered crossbars –see e.g. [7] [8] [12] and our previous work [9]; from industry, a representative example is [13]. This paper differs from the above in that we consider buffered crossbars directly operating on *variable-size* packets. To the best of our knowledge, there have been only two previous studies on this topic: *(i)* Stephens and Zhang [6]; and *(ii)* Yoshigoe and Christensen [14][15] [16].

Our present work differs from these studies in the following ways. Firstly, we consider the *hardware* implementation of such crossbars: section 3 discusses a number of issues and gives cost numbers (gates, area, power) from a designed such switch; the only other hardware study, [16], concerns a relatively low-end FPGA-based system, and does not discuss the internal logic of the crossbar chips at all. Secondly, our *performance evaluation* is more comprehensive than previous studies, as explained below.

Stephens and Zhang [6] consider variable-size *internal* packets, but limit their length up to twice the minimum packet size, (i.e. up to 80 bytes); larger external packets are still segmented. Also, [6] only simulates a 4×4 switch, with one specific crosspoint buffer size, under one specific traffic scenario (which contains traffic of different QoS classes, with three specific packet sizes, and an aggregate load in excess of 100%). While this simulation demonstrates the excellent properties of buffered crossbars with appropriate line schedulers, we simulate a 32×32 switch under a much wider spectrum of traffic scenarios, and we compare our results to unbuffered crossbars with various speedup factors.

Yoshigoe and Christensen [14] evaluate the performance of the buffered crossbar only for crosspoint buffer size of 1500 bytes, without specifying the backpressure RTT, while we explicitly study the dependence of performance on the relative sizes of these two parameters. Next, [14] simulates variable-size packets only in one experiment (#3), using Poisson arrivals with uniformly selected outputs, while our traffic mix closely represents internet backbone traffic, and we examine non-uniform destinations and hot-spot scenarios, as well. In addition, we show that it is trivial for the crossbar to provide cut-through operation, and we use cut-through in our simulations. Note that we do *not* consider multi-priority traffic in this paper (as [14] does), due to lack of space; we are considering multiple priorities, in great depth, in [17] [18].

3. Internal Organization and Cost

In this section we discuss implementation issues for variable-packet-size buffered crossbars, and we present the silicon cost and the power consumption of such a switch that we have designed, synthesized, and placed and routed in a state-of-the-art CMOS technology.

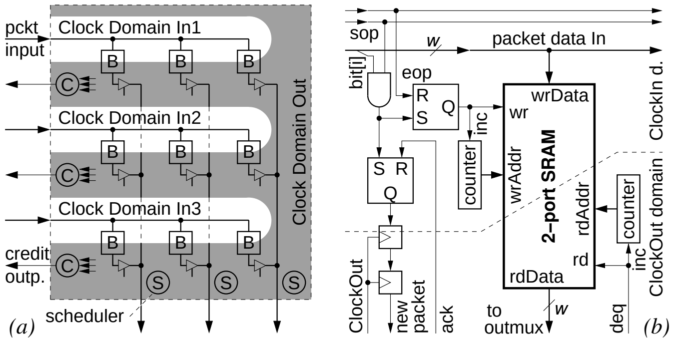

Fig. 2. (a) Clock domains, (b) crosspoint logic

3.1. *Crosspoint Logic and Inter-Clock Communication*

Buffered crossbars allow a simple separation among the clock domains in the switch. By placing their boundaries in the crosspoint switches, as shown in figure 2(a), we eliminate elastic buffers at the chip inputs; this reduces latency and power consumption, because each word of the packet payload is only written once into and read once out of a memory during its transition through the chip.This requires 2-port crosspoint buffers as described in [17].

Figure 2(b) shows the entire logic of each crosspoint – the reader will appreciate the simplicity of the architecture. Packets arrive through the w-bit bus; transceiver logic at the input of the chip asserts sop (start-of-packet) in the proper clock cycle. We assume that the first word of each packet contains a multicast bitmap, specifying the crosspoints where the packet should be enqueued. Enqueueing is activated when both sop and the appropriate bit of the bitmap are ON, and it is terminated when the input transceiver asserts eop (end-of-packet). We do not have to check for buffer overflow, because the flow control protocol ensure it will never happen. In the baseline architecture, shown in fig. 2(b), which was used for our prototype described in the next sections, each crosspoint contains a single FIFO queue hence, enqueue addresses are generated by a single counter.

We assume that the length of the packet appears in one of the first words of the packet (3rd and 4th bytes, for IP) and it is written into the buffer. Then, the only inter-clock domain communication needed is for the output to be notified every time a new packet arrives –the packet length can be found in the FIFO. This notification is accomplished by raising a flag, newPacket, which gets synchronized to the output clock; when the output sees newPacket, it increments a counter (not shown) and resets the flag (before the minimum-size packet duration elapses). The counter mentioned stores the number of packets that are currently enqueued in this crosspoint. When the output decides to dequeue a packet from this crosspoint, it decrements that counter, and it raises deq for the proper number of cycles (determined by the packet length number which is read out of the FIFO). Buffer underflow cannot occur, since the input always writes entire packets into the buffer.

3.2. *Cut-through, Output Scheduling*

We just saw that the output is notified of a packet arrival one synchronization delay after packet enqueueing starts. If this is the only packet in the buffer, and it belongs also to the output chosen by the output scheduler, the dequeueing of the packet may start right away, thus implicitly executing a *cut-through* operation, in order to reduce latency. Cut-through will work correctly as long as the output clock frequency does not exceed the input clock frequency by more than the synchronization delay divided by the maximum-size packet duration. Cut-through significantly reduces the buffer size required to get satisfactory performance, compared to store-and-forward; cut-through also reduces latency under light load.

Each output port must have a scheduler, which can be as simple or as sophisticated as desired. The inputs to each scheduler are the packet counts for the buffers of its column. Notice, however, that buffer occupancies in terms of bytes are not known, because enqueue and dequeue pointers are in different clock domains and cannot be subtracted from each other. In our prototype implementation we assume plain round-robin output schedulers (oblivious of packet size) that execute a very simple algorithm: serve the next crosspoint with a non-zero packet count, following the last-served crosspoint in circular order. For fancier output schedulers see [20].

3.3. *Input Scheduling and Credit Flow Control*

On its input side, a buffered crossbar chip communicates with the ingress line cards that contain the VOQ's (figure 1(b)). A scheduler per port selects the VOQ from which the next packet will be forwarded to the crossbar; eligible VOQ's are those that *(a)* are non-empty, and *(b)* will not cause their corresponding crosspoint buffer to overflow. If these schedulers were placed in the crossbar chip, they would have inexpensive and fast access to crosspoint buffer occupancy information, but *(i)* these schedulers would add to the cost of the crossbar chip; *(ii)* ingress line cards would need to communicate to the crossbar the size of the head packet of each VOQ[3]; and *(iii)* the scheduler's decision would need to travel to the line card before the next packet can depart from the line card to the crossbar, effectively increasing the scheduler's latency.

Instead, it is preferable to place input schedulers in their corresponding ingress line cards. We then need to communicate the occupancy of the crosspoint buffers from the crossbar to the line cards. The easiest method to do that is to notify the line card every time a packet departs from the crossbar. The notification (credit) must specify the output port of the departure, but does not need to specify the packet size: the line card can remember the sizes of all the packets that it has recently sent to the crossbar [17]. Thus, the only module needed on the input side of the switch chip is the generator of the sop and eop signals in fig. 2(b), henceforth called *enqueue controller* (enqC).

We prefer credit-based flow control, to the popular start/stop flow control, because the latter requires an additional RTT-window (plus a hysteresis safety margin) of buffer space per crosspoint [17]. A non-empty VOQ is eligible when the size of its head packet does not exceed the credit count of its

[3] One message every time a VOQ is served, plus one message every time a packet arrives into an empty VOQ.

Module (number of instances)	Gates (K)	Flip-Flops (K)	SRAM 2-port (bits)	Area 0.18μm (mm^2)	Area 0.13μm (mm^2)	Power 0.18μm (Watt)	Power 0.13μm (Watt)
XPD (1024)	21.6	93.0		9.9	4.18	0.8	0.5
XPM (1024)			16 M	286.0	130.00	1.0	0.2
enqC (32)	2.2	0.6		0.1	0.03	0.3	0.2
OS (32)	41.5	11.0		2.5	1.49	0.5	0.3
CRS (32)	22.3	6.6		1.2	0.32	0.4	0.2
Wiring				117.7	60.94	3.6	2.2
Subtotal	87.6	111.2	16 M	417.5	197.40	6.6	3.6
Estimated power consumption of 32 link SERDES and pad drivers						23.0	14.4

TABLE I

COMPLEXITY OF THE SUBMODULES OF THE SWITCH (TOTAL NUMBERS SHOWN, INCLUDING ALL INSTANCES), AND TOTAL COST (INCLUDING WIRING).

desired output port. Choosing among the eligible VOQ's is an issue of QoS support, outside the scope of this paper; in our simulations (sec. 4) we assumed round-robin input scheduling.

3.4. Silicon Cost Estimates

In order to measure the silicon characteristics and power consumption of such a switch, we designed, synthesized, and placed and routed a 32×32 variable-packet-size buffered crossbar switch. Table I shows gate, flip-flop, SRAM, area, and power consumption cost for our chip which was targeted to UMC 0.18μm [23] and 0.13μm low power [24] CMOS technology. Circuit and area cost concern the chip core and include wiring, but exclude bonding pads, their drivers, and serializer-deserializer (SERDES) circuits. Placement, routing, and timing optimizations were performed for the 0.18μm technology; the 0.13μm figures have been computed by extrapolation based on UMC's datasheets. The internal datapath width of the device is 32 bits per port. Periphery cost (I/O pads, SERDES) is only included for power consumption, using estimates based on [13]; we assume 4 differential pairs for every input or output link and 1 differential pair for the credit line. In order to measure the above mentioned characteristics of the core, the circuit was designed using the Verilog Hardware Description Language, synthesized using Synopsys [25] and placed and routed using Silicon Encounter [26]. The functionality of our final netlist has been fully verified, using various traffic patterns, including: (a) all inputs send packets to random outputs, back-to-back; (b) all inputs send packets to a specific output, back-to-back. In both cases, we used three classes of packets: only minimum size packets (40 bytes), only maximum size packets (1500 bytes) and packets with random sizes. In the longest run, each input sends 3000 packets to various outputs. The results also proved that the implemented output scheduler can schedule packets in a back-to-back manner in all cases.

The lines of Table I refer to: crosspoint datapath (XPD); crosspoint memory (XPM); enqueue controller (enqC); output scheduler (OS); and credit sequencer (CRS): Cost figures are for the entire chip (all block instances). As seen, everything else besides crosspoint memories and wiring occupies just 5% of the area, indicating the simplicity of the architecture. Since crosspoint memories cost 68.5% of the total area, we decided to use crosspoint buffers of 2 KByte each (based on sec. 4 results), and not to support separate buffers for the different priorities in this prototype. Consider that doubling the number of crosspoint buffers to support an additional class of priority will increase the area of the chip core by 68.5%, which is only feasible in 0.13μm technology; future technologies or embedded DRAM will improve the situation. Those future technologies will also allow "Jumbo Frames" (10KB packet) support. Using current technology, Jumbo Frames, would limit the number of input and output ports to 24 or less.

	Gates	FF	SRAM (bits)	Area (μm^2) in 0.18μm
per-input	27.7 x i + 69	8.1 x i + 18		1310 x i + 2689
per-output	53.5 x i	12 x i		2630 x i
per-crosspoint	23.1	94.4	16K	289206

TABLE II

PER INPUT/OUTPUT/CROSSPOINT COSTS; i IS THE NUMBER OF THE INPUT PORTS OF THE SWITCH

In general, it is claimed that power consumption can be the primary limiting factor for switch chip throughput. The power consumption figures in Table I are based on a total incoming throughput of 300Gb/sec for the 32×32 switch; given the datapath width of 32 bits, the corresponding clock frequency of the core is 300MHz. According to the synthesis tool, our design, after placement and routing, does achieve this speed in 0.18μm technology. The typical steady state consumption of the whole switch core is 6.6 W in 0.18μm technology and 3.6 W in 0.13μm. This was measured assuming uniform traffic at load 100% and consisting of minimum size packets. In this case, 32 crosspoint buffers are active at the same time together will all the input and output schedulers and peripheral circuitry.

Worst-case instantaneous memory power consumption occurs when all inputs receive multicast packets destined to all outputs: all 1024 crosspoint buffers perform write operations in that case. This situation can last up to 1.6μseconds –the time needed to fill up a 16 Kbit buffer at 10 Gbps. Following that, backpressure stops the incoming traffic, and buffer memories perform read accesses only, one memory per output at a time. The *read* power consumption of crosspoint buffers corresponds to their read throughput, which never exceeds 32 buffers operating in parallel. *Write* power consumption corresponds to write throughput; the long-term average write throughput cannot exceed read throughput, since every byte that is ever written once must also be read once.

Note that average power consumption of the chip is dom-

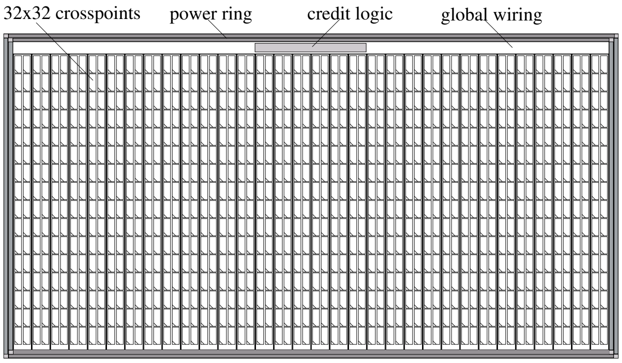

Fig. 3. 32×32 crossbar layout

inated by pad drivers and serializers-deserializers (SERDES); core consumption is just around 20% of total chip consumption. In turn, core consumption is dominated by long-wire drivers: driving 1024 input data wires across the full chip width, and driving 1024 output data wires to the bottom of the chip (32 links × 32 bits/link = 1024 wires). Buffer memories end up consuming, on the average, only 5 to 15% of the core power or 2 to 3% of the chip power.

Table II provides cost figures on a per-input, per-output, and per-crosspoint basis; area cost includes wiring and 2 KB of 2-port SRAM per crosspoint. This is useful in evaluating switch configurations with different numbers of ports. These figures were derived by designing four different switches based on the proposed architecture: 4×4, 8×8, 16×16, and 32×32; then, we averaged the area, gate, and flip-flop sums per input, output and crosspoint, respectively. Since each input and output port contains a scheduler whose complexity depends on the numbers of input ports, the per-port complexity depends on the fan-in of the switch. The per-input cost is dominated by the credit sequencer (CRS), and includes the necessary enqC module; the per-output cost consists mostly of the output scheduler (OS); the crosspoint cost includes XPM and XPD.

Chip layout can be seen in figure 3. The chip was placed and routed hierarchically by organizing it in columns and by placing the 32 crosspoint memories of each column in pairs, which proved to be the optimal organization. Data input lines are fed from the west side of the chip, data outputs come out the south side, and credits are sent to the north side.

4. Performance Evaluation

4.1. Simulation Environment

We implemented an event driven simulator in C++, that models a buffered crossbar switch under backbone IP traffic, with packet size varying between 40 and 1500 bytes. In all experiments we have assumed a 32x32 switch, a port speed of 10 Gbps, no internal packet header overhead and no internal speedup. Our input line-cards and crosspoint buffers support cut-through operations. When a packet starts being transmitted towards the output lines of the crossbar, the corresponding credit/acknowledgment is generated. The credit line rate is such that the duration of a credit transmission equals a minimum packet transmission time [27] [4]. Credits destined to the same input line-card are sent in FIFO order. The RTT between input line-cards and switch fabric has been set to 500 byte times (corresponding to 400 ns at 10 Gbps line rate), resulting as the sum of the following delays:

- input scheduling time, 30ns;
- VOQ memory access time, 80ns;
- packet propagation time, including time of flight, pipeline logic, and serialization/deserialization delay, 114ns;
- output scheduling time, similarly, 30ns;
- credit propagation time, similarly, 114 ns;
- credit transmission time, 32ns.

For additional information on design issues and on the simulator refer to [17].

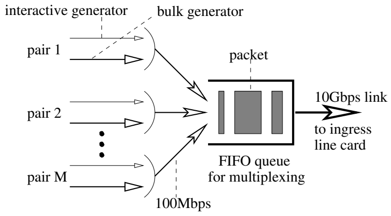

Fig. 4. Basic Traffic Generator. The aggregate load of a pair of sources is 100Mbps. For $M \times$ 100Mbps load we multiplex M pairs.

We model variable-size packet arrivals at the input ports, using mostly two distinct traffic patterns: **PoisPar**, a poisson process arrival distribution with packet sizes that follow the bounded pareto distribution (min 40, max 1500, average 370 bytes); **SynthBackb**, a synthetic pattern that we created based on internet statistics sources [29], so as to emulate as much as possible realistic, backbone IP traffic.

For **SynthBackb**, the traffic arriving at each ingress line-card is generated by multiplexing M pairs of sources in a FIFO queue (see fig. 4). The first source in each pair (interactive-generator -IG), generates sessions (i.e. streams of packets), emulating *(a)*, interactive applications which are dominated by small packets (e.g. TELNET) and *(b)*, TCP acknowledgements. The sessions of the second source (bulk-generator -BG) emulate bulk transactions such as FTP transfers or HTTP page responses. The duration of a session generated by IG and BG follows the pareto distribution with mean value 125 packets and 8 Kbyte respectively [28]. All sessions are delimited by an idle period and they are generated according to a Poisson process. Packets within IG sessions vary from 40 to 44 bytes and their interarrival time follows the exponential distribution. A BG session consists of a burst of back-to-back packets, having the same size –1500 (x%) or 552 (y%) or 576 (z%) bytes– except for the last one; x, y, z and the ratio of rates between IG and BG are selected so that 60% of all generated packets have sizes between 40-44 bytes, 18% 552 or 576 bytes, and 18% 1500 bytes [29]. Each pair of sources has an aggregate rate 100 Mbps; so for a M * 100 Mbps load we multiplex M pairs.

[4] More conservative than the hardware assumption described in the last section.

4.2. Simulation Experiments and Results

1) Round-Trip-Time Experiment: In this experiment we assume a single, persistent flow, i.e. load is 10 Gbps. For crosspoint buffer size (B) varying from 1400 to 2400 bytes we measure the output utilization as a fraction of 10Gbps. We repeat the experiment for RTT values varying from 250 to 700 byte times. First, in what we believe to be the most realistic case, the packets of the flow are generated by **SynthBackb**.

Next, we experiment with a worst case scenario where we continuously alternate between packets $p1$ and $p2$ with sizes $s1$, equal to 1500 and $s2$, equal to max(B-1499, 40) bytes; $s1$ and $s2$ have been selected so that *(a)* $s2$ is as small as possible, while *(b)*, $p2$ is able to block $p1$ at the input. Condition *(b)* creates the necessary condition for underutilization, and *(a)* maximizes the duration of this possible underutilization.

Figure 5 shows that output underutilization occurs for every B less than $1500+RTT\times Line\_Rate$; however, by employing a crosspoint buffer size equal to $1500 + RTT \times Line\_Rate$ full output utilization is achieved. This happens because if B equals $1500 + RTT \times Line\_Rate$, we impose that $p1$ will be blocked at the input only if $s2$ is greater than $RTT \times Line\_Rate$. But in this case, when $p1$ will be ready for transmission at the output after receiving $p2$'s credit (i.e. RTT times after starting transmitting $p2$), the output will still be busy transmitting $p2$, because its size is greater than $RTT \times Line\_Rate$. So, with this buffer size, full output utilization is guaranteed.

Under **SynthBackb** arrivals the knee at crosspoint buffer size $1500 + RTT \times Line\_Rate$ is also observable (fig. 5), but not as strongly as with the aforementioned worst-case scenario.

2) Delay Experiment: We run simulations both under uniformly destined and hotspot traffic, using the traffic generator **SynthBackb**. Under uniform traffic, the destination port of each session is chosen uniformly; all packets in a session have the same destination port. For the hotspot traffic, we follow the methodology in [19]: each destination belonging to a designated set of "hotspots" receives traffic at 100% collective load, uniformly from all sources; the rest of the destinations receive uniform traffic. Without loss of generality, we assume that the hotspots are ports 0, 1, 2 and 3. The reported delay is the time between the packet's first byte exit-time minus the packet's first byte enter-time, averaged over all packets; for the hotspot case, we take into account only the packets destined to non-hotspot outputs. The results are compared to output queueing (OQ), which is the reference model, and to CIOQ using iSLIP, one of the most representative and efficient examples of the unbuffered crossbar family. For iSLIP we consider one iteration, 64 byte segments and various speedup factors.

Figure 6 shows the results for uniform traffic. The iSLIP switch with no speedup saturates at input load 0.65; for speedup equal to 1.2 it saturates near load 0.8. Observe that the performance of the proposed architecture with a crosspoint buffer of 2KB is very close to OQ; iSLIP with speedup 2 also performs close to the ideal system. Under hotspot traffic, in the buffered-crossbar system, we observed that non-hotspot traffic stays unaffected by the presence of hotspots (the uniform and the hotspot plots actually match), due to

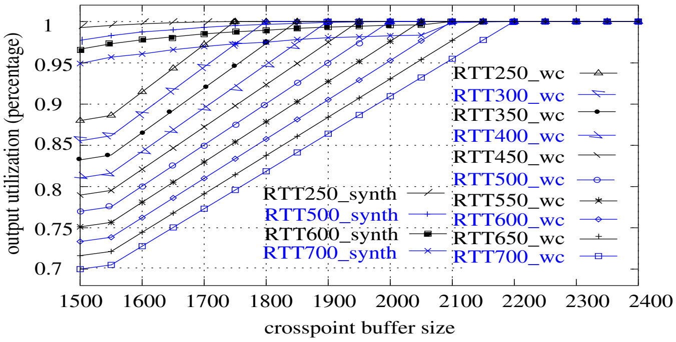

Fig. 5. RTT Experiment. For buffer size less than 1500 bytes, all measurements for output utilization are zero. Upper curves stand for SynthBackb traffic, lower curves for worst case.

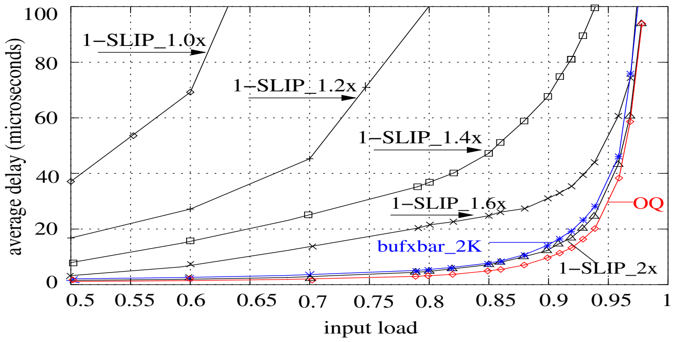

Fig. 6. Delay Experiment.32 $\times$ 32 switch, uniformly destined traffic. Maximum load is 98%.

the isolation/protection that is provided to flows (input/output pairs) by the crossbar/queueing architecture. On the other hand, when we apply hotspot traffic to the iSLIP switch, all flows' performance degrades considerably due to the absense of any flow control. The corresponding diagrams are not shown here, due to space limitations, but they can be found in [30], along with information regarding the simulation environment.

3) Throughput Experiment: Next, we experiment with unbalanced traffic (i.e. non-uniform destinations), considering an unbalance factor f, as in [12]: input i sends to output i with probability $f + \frac{1-f}{N}$, and to all other outputs uniformly with collective probability $1 - f$. In this experiment we use the **PoisPar** arrival model and we measure the switch throughput as a fraction of the maximum possible one (320 Gbps in our simulation scenario). For iSLIP (1 iteration, 64 bytes segment, speed-up equal to 1.0) packets have sizes equal to $k \times 64 bytes$ (k integer), so as to eliminate segmentation overheads. Even under this assumption, we find (see fig. 7) that the CICQ architecture with variable-size packets considerably outperforms the CIOQ (iSLIP) switch. With 2KB crosspoint buffer size and under this scenario, the maximum switch load efficiently supported by the buffered crossbar is 0.90 versus 0.58 for the iSLIP switch.

Conclusion

We presented the architecture of an innovative buffered crossbar switching variable-size packets. The crossbar chip organization that we propose is fairly simple and cost-effective. Using standard cell technology at 0.13μm, a 32 $\times$ 32 switch

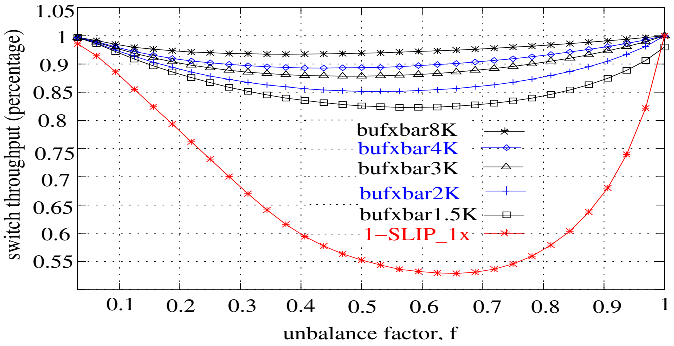

Fig. 7. Throughput Experiment.32 × 32 switch, unbalanced traffic. Load is 100%. For crossbar, buffer size is 1.5K, 2K, 3K, 4K and 8K.

supporting more than 300 Gbps of aggregate input throughput can be implemented within a single chip, in less than $200mm^2$ and at core power dissipation below 4 Watts. Through simulations, we demonstrated that the proposed organization, using no speedup, performs very close to the ideal output queuing system, while it outperforms practical unbuffered crossbar architectures with speedup less than 2×.

ACKNOWLEDGMENTS

The authors would like to thank Georgios Sapunjis, for his contribution to the development of the ideas presented, Dionysios Pnevmatikatos and Georgios Kalokairinos for their useful suggestions and Michalis Ligerakis for his technical support.

REFERENCES

[1] T. Anderson, S. Owicki, J. Saxe, C. Thacker: "High-Speed Switch Scheduling for Local-Area Networks", *ACM Trans. on Computer Systems*, vol. 11, no. 4, Nov. 1993, pp. 319-352.

[2] R. LaMaire, D. Serpanos: "Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues", *IEEE/ACM Trans. on Networking*, vol. 2, no. 5, Oct. 1994, pp. 471-482.

[3] N. McKeown: "The iSLIP Scheduling Algorithm for Input-Queued Switches", *IEEE/ACM Trans. on Networking*, vol. 7, no. 2, April 1999, pp. 188-201; http://tiny-tera.stanford.edu/∼nickm/papers/ToN_April_99.pdf

[4] N. Ni, L. N. Bhuyan: "Fair scheduling for Input Buffered Switches", citeseer.nj.nec.com/482342.html

[5] P. Krishna, N. Patel, A. Charny, R. Simcoe: "On the Speedup Required for Work-Conserving Crossbar Switches", *IEEE J. Sel. Areas in Communications*, vol. 17, no. 6, June 1999, pp. 1057-1066.

[6] D. Stephens, H. Zhang: "Implementing Distributed Packet Fair Queueing in a scalable switch architecture", *Proc. INFOCOM'98 Conf.*, San Francisco, CA, March 1998, pp. 282-290.

[7] M. Nabeshima: "Performance Evaluation of a Combined Input and Crosspoint Queued Switch", *IEICE Trans. Commun.*, vol. E83-B, no. 3, Mar. 2000, pp. 737-741.

[8] T. Javidi, R. Magill, and T. Hrabik: "A High-Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric" *Proc. IEEE Int. Conf. on Communications (ICC'2001)*, Helsinki, Finland, June 2001, vol. 5, pp. 1586-1591.

[9] G. Georgakopoulos: "Few buffers suffice: Explaining why and how crossbars with weighted fair queuing converge to weighted max-min fairness", Dept. of Computer Science, University of Crete, Heraklion, Crete, Greece, July 2003, 6 pages. http://archvlsi.ics.forth.gr/bufxbar/

[10] S. Nojima, E. Tsutio, H. Fukuda, M. Hashimoto: "Integrated Services Packet Network Using Bus Matrix Switch", *IEEE J. Sel. Areas in Communications*, vol. 5, no. 8, October 1987, pp. 1284-1292.

[11] M. Katevenis: "Fast Switching and Fair Control of Congested Flow in Broad-Band Networks", *IEEE J. Sel. Areas in Communications*, vol. 5, no. 8, October 1987, pp. 1315-1326.

[12] R. Rojas-Cessa, E. Oki, and H. Jonathan Chao: "CIXOB-k: Combined Input-Crosspoint-Output Buffered Switch", *Proc. IEEE GLOBECOM*, 2001, vol. 4, pp. 2654-2660.

[13] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, I. Iliadis: "A Four-Terabit Packet Switch Supporting Long Round-Trip Times", *IEEE Micro Magazine*, vol. 23, no. 1, Jan./Feb. 2003, pp. 10-24.

[14] K. Yoshigoe, K. Christensen: "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2001)*, Dallas, TX, USA, May 2001, pp. 271-275; http://www.csee.usf.edu/∼christen/hpsr01.pdf

[15] K. Yoshigoe and K. Christensen: "An Evolution to Crossbar Switches with Buffered Cross Points", *IEEE Network Magazine*, Vol. 17, No. 5, pp. 48-56, September-October 2003.

[16] K. Yoshigoe, K. Christensen, A. Jacob: "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed", *Proc. IEEE Int. Performance, Computing, and Communications Conf.*, April 2003, pp. 481-485; http://www.csee.usf.edu/∼christen/ipccc03.pdf

[17] N. Chrysos: "Design Issues of Variable Packet Size, Multiple Priority Buffered Crossbars" *Technical Report, ICS FORTH Hellas, Department of Computer Science, University of Crete* October 2003; http://archvlsi.ics.forth.gr/bufxbar

[18] N. Chrysos, M. Katevenis: "Multiple Priorities in a Two-Lane Buffered Crossbar", Inst. of Computer Science, FORTH, Heraklion, Crete, Greece, March 2004, 7 pages; http://archvlsi.ics.forth.gr/bufxbar

[19] G. Sapountzis, M. Katevenis: "Benes Switching Fabrics with O(N)-Complexity Internal Backpressure", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2003)*, Torino, Italy, June 2003, pp. 11-16; http://archvlsi.ics.forth.gr/bpbenes/

[20] K. Harteros: "Fast Parallel Comparison Circuits for Scheduling", *Institute of Computer Science, FORTH*, Technical Report FORTH-ICS/TR-304, 78 pages, March 2002; http://archvlsi.ics.forth.gr/muqpro/cmpTree.html

[21] Quantum Flow Control (QFC) Alliance: *"Quantum Flow Control: A cell-relay protocol supporting an Available Bit Rate Service"*, version 2.0, July 1995; originally at http://www.qfc.org but no longer there – copy at http://archvlsi.ics.forth.gr/∼kateveni/534/qfc/

[22] G. Kornaros e.a.: "ATLAS I: Implementing a Single-Chip ATM Switch with Backpressure", *IEEE Micro*, vol. 19, no. 1, Jan/Feb. 1999, pp. 30-41; http://archvlsi.ics.forth.gr/atlasI/hoti98/

[23] UMC 0.18 micron Libraries; http://www.umc.com/english/design/b_3.asp

[24] UMC 0.13 micron Libraries; http://www.umc.com/english/design/b_1.asp

[25] Synopsys Design Compiler; http://www.synopsys.com

[26] Cadence Design Systems; Silicon Encounter Place-and-Route Reference Manual; http://www.cadence.com

[27] Ferdinand Gramsamer e.a.: "Flow Control Scheduling" , revised and extended version of *ICCCN 2002*, Oct. 14-16, Miami, FL, pp438-443

[28] Bruce A. Mah: "An empirical Model of HTTP Network Traffic", *INFOCOM'97*

[29] "Cooperative Association for Internet Data Analysis"; http://www.caida.org

[30] G. Passas: "Performance Evaluation of Variable Packet Size Buffered Crossbar Switches", *Technical Report FORTH-ICS/TR-328, Inst. of Computer Science, FORTH, Heraklion, Crete, Greece; B.Sc. Thesis, Univ. of Crete;* November 2003; http://archvlsi.ics.forth.gr/bufxbar