Variable-Size Multipacket Segments in Buffered Crossbar (CICQ) Architectures

Manolis Katevenis and Georgios Passas

Institute of Computer Science (ICS), 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,passas}@ics.forth.gr`

***Abstract*— Buffered crossbars can directly switch variable size packets, but require large crosspoint buffers to do so, especially when jumbo frames are to be supported. When this is not feasible, segmentation and reassembly (SAR) must be used. We propose a novel SAR scheme for buffered crossbars that uses variable-size segments while merging multiple packets (or fragments thereof) into each segment. This scheme eliminates padding overhead, reduces header overhead, reduces crosspoint buffer size and is suitable for use with external, modern DRAM buffer memory in the ingress line cards. We evaluate the new scheme using simulation, and show that it outperforms existing segmentation schemes in buffered as well as unbuffered crossbars. We also study how the size of the maximum segment affects system performance.**

***Index Terms*— crossbar switches, routing, segmentation, simulation.**

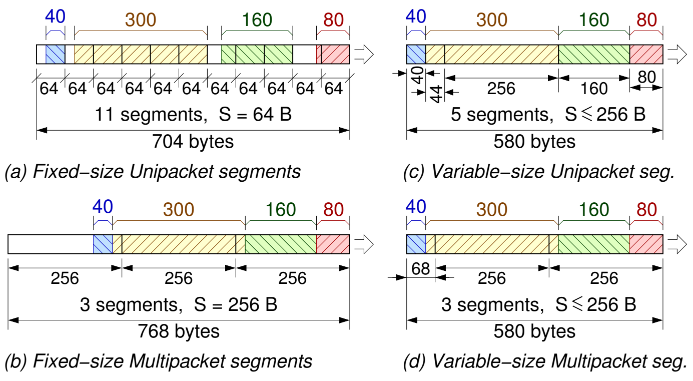

Fig. 1. Four packets in a flow, of sizes 80, 160, 300, and 40 bytes respectively, as they traverse a crossbar under four different segmentation schemes.

I. Introduction and Related Work

Crossbars are the building blocks for modern switching fabrics and router systems. Traditional crossbars were unbuffered –with buffering provided only on the ingress and egress line cards– hence transmissions over the crossbar ports had to be synchronized with each other, leading to the use of fixed-size segments (cells). Recently, *buffered crossbars*, which have small buffers at their crosspoints (CICQ – combined input-crosspoint queueing), have emerged as an advantageous architecture, mainly due to their scheduling efficiency [1] [2] [3] [4] [5] [6]. A subsequent observation was that buffered crossbars can directly switch *variable-size* units [1] [7]. In [8] we studied the details of such a buffered crossbar operating directly on variable-size packets, without first segmenting them into fixed-size cells; this eliminates packet reassembly at the egress line card, as well as the need for speedup to cope with segmentation and reassembly (SAR), thus also eliminating egress queueing, and, overall, greatly reducing cost or increasing port speed. This architecture [8], however, has two limitations which we seek to remedy in the present paper.

First, for switch operation without SAR, the required buffer size per crosspoint, as shown in [8], is at least one maximum-size packet plus one round-trip-time (RTT) worth of data; Reference [8] assumed 1500-byte maximum packets (old ethernet limit) and RTT $\leq$ 500 B, thus using 2 KByte buffers. For a 32×32 switch, the resulting silicon area is expensive in current technologies; hence, for the high port-count switches of the next few years, it is desirable to limit buffer size per crosspoint to probably 1024 bytes or less. Second, when "jumbo frames" (ten or more KBytes per packet) are to be switched without SAR, the required buffer size is very large, thus limiting port count below a dozen or so.

This paper examines buffered crossbars that use SAR in order to overcome one or both of the above limitations. We introduce a *novel segmentation* scheme that exploits the buffered crossbar capability to switch variable-size units, thus avoiding the speedup requirements of traditional SAR schemes. This novel scheme, *variable-size multipacket segments*, is illustrated in Fig. 1(d), and is the combination of two existing schemes, (b) and (c). Fig. 1 first reviews the existing segmentation schemes; for simplicity, we do not show the per-segment headers, used for the routing inside the crossbar:

(a) Fixed-size Unipacket segments : this is the traditional segmentation scheme. Each segment contains a single packet or fragment thereof. Segment size is fixed, so the last segment of a packet often contains useless padding bytes. This overhead is the first disadvantage, because it requires crossbar speedup (i.e. port speed must be lower than crossbar speed, often by as much as a factor of 2 to 3). Segment size is usually small, in order for the (quite frequent) minimum-size (40-byte) IP packets not to incur excessive padding overhead. This small segment size is the second disadvantage, because it requires high scheduler rate (which is a problem in centrally-scheduled unbuffered crossbars); additionally, it increases the relative overhead of per-segment headers.

(b) Fixed-size Multipacket segments [9] [10] (envelopes)

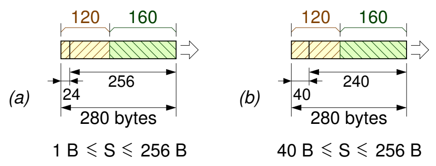

Fig. 2. (a) Naive segmentation; (b)ensuring 40-byte minimum segment size.

[11] (cell-merging)**:** packets that share a common destination (belonging to the same flow) are packed inside segments, contiguously one after another; each segment may now contain one or more packets or fragments thereof. Segment size can now be increased, thus relaxing scheduler rate and reducing header overhead. In order to control the padding overhead, especially given the larger segment size, partially filled segments are held in the input queues until more packets arrive or until a timer expires or excessive bandwidth is available. The disadvantages are increased delay until segments are filled, and the padding overhead needed whenever partially filled segments are transmitted due to timer expiration.

(c) Variable-size Unipacket segments [1]**:** assuming a buffered crossbar, segments can have a variable size. This eliminates padding overhead. Unipacket segments were used in [1], so scheduler rate and header overhead were not reduced; additionally, this scheme is not well adapted to the use of DRAM for ingress line card buffering (section II).

(d) Variable-size Multipacket segments : this is the novel segmentation scheme introduced and evaluated in the present paper; it is the combination of (b) and (c). Variable-size segments eliminate padding overhead, like (c); thus, unlike (b), packets don't need to wait for segments to fill up. Multipacket segments reduce header overhead and average scheduler rate, like (b). Peak scheduler rate can still be high, when partially filled segments are released. Unlike [1], we bound peak scheduler rate by imposing a minimum segment size, as shown in figure 2(b).

We assume a minimum segment size equal to the minimum (IP) packet size (40 bytes), so that no waiting is required to fill a segment with a single 40-byte packet; buffered crossbar scheduling is simple and efficient, so a 40-byte scheduling time is reasonable. Because segments are multipacket, all segments but the last two in a queue have a fixed size, unlike (c). This fixed segment size is large enough, so peak DRAM throughput can be achieved; hence our scheme is suitable for line cards with DRAM buffers (section II-A).

Following the above discussion of related work, the rest of this paper is organized as follows. Section II describes the overall system architecture, precisely defining our scheme, and focusing on the ingress line card and DRAM buffer organization; we show that transmitting multipacket segments to the crossbar simplifies the ingress packet buffers. Section III presents the results of our simulations, comparing the performance of our scheme to that of previous schemes, and studying how segment size affects system performance. An interesting point is that satisfactory performance is achieved with a crosspoint buffer size of just one (maximum) segment size, unlike [8] where that buffer size should be one maximum packet size *plus* one RTT worth of data.

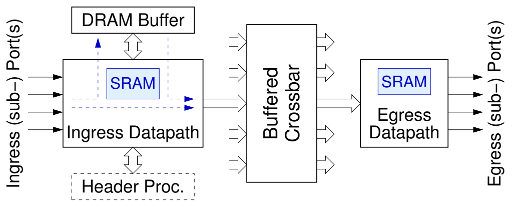

Fig. 3. Overall datapath architecture.

II. System and Line Card Architecture

Crossbar chips usually operate in connection with line cards, in an overall system as illustrated in Fig. 3. We are concerned with the datapath of the system; header processing may occur on the same or in other chips. Each block in the figure corresponds to a single chip, except for the "SRAM" blocks which are assumed to be inside the datapath chips. As usual in combined input-crosspoint queueing (CICQ) architectures, the largest queues are on the input side, consisting of virtual-output queues (VOQ) residing in the ingress datapath; the buffers in the crossbar hold small queues, and rely on back-pressure (credit flow control) to avoid overflow [8]. Because input queues may grow large, most switch and router systems include off-chip DRAM in the ingress datapath to provide adequate buffer capacity for these VOQ's.

Output queues exist and may grow large in systems that use crossbar speedup. When buffered crossbars operate directly on (variable-size) packets, output queues can be eliminated altogether [8], so the egress datapath could be null. In the present system, an egress datapath chip is needed to provide segment-to-packet reassembly; additionally, it is needed if multiple egress sub-ports are to be provided. However, in the lack of crossbar speedup, the required buffer capacity for both purposes is limited, so on-ship SRAM should suffice, avoiding off-chip buffers.

DRAM buffer memory on the ingress side is responsible for a major portion of system cost, in terms of chips, pins, and power consumption; its throughput may often constitute a major performance bottleneck, especially when it has to support short-packet accesses. Our scheme expressly eliminates short-segment DRAM accesses; it also streamlines external memory traffic so as to ensure peak DRAM throughput utilization, as explained next.

A. *Ingress Linecard Queueing Architecture*

The virtual-output queues (VOQ) in the ingress datapath are implemented as linked lists consisting of fixed-size data blocks and the block size equals the *maximum* segment size; see Fig. 4. Arriving packets are classified, then written into the proper VOQ. Packets in a VOQ are written contiguously one after another, as in the fixed-size multipacket configuration (Fig. 1(b)). Notice that fixed-size blocks are only used in the ingress datapath for memory management purposes –when

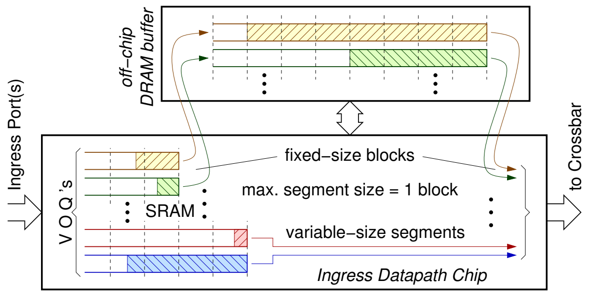

Fig. 4. VOQ datapath in the ingress line card.

transmitted through the crossbar they are converted to variable-size segments. The worst-case memory space waste due to partially filled blocks is one block per VOQ.

A VOQ can reside partly in the SRAM on-board the ingress datapath chip, and partly in the off-chip DRAM, as in [13]. Tail blocks must be in the SRAM, while other blocks can be in the DRAM. All transfers to and from DRAM occur at block granularity[1]. Block size should be chosen to ensure peak DRAM throughput all the time. For a typical modern SDRAM buffer providing 45 Gb/s of aggregate throughput, a block size of 512 bytes will ensure full throughput utilization[2]. Such a 512-byte block size, in our system, entails a 512-byte maximum segment size, which in turn demands 512-byte crosspoint buffers (section II-B); this is a comfortable size for modern buffered crossbars [8].

Iyer e.a. [13] assumed that all VOQ's have their tail *and* their head blocks in SRAM, while their middle blocks migrate to DRAM. By contrast, our SAR scheme allows the head blocks to remain in DRAM until ready to be switched, at which time they can move directly from DRAM to crossbar, as illustrated in Fig. 4; this halves SRAM occupancy. The reason is that all segments but the last two in a queue have a fixed size, equal to the maximum segment size, hence equal to one DRAM block (Fig. 1(d) and 2(b)).

Another optimization relative to [13] is to allow low-occupancy VOQ's (such as the bottom queues in Fig. 4) to reside entirely in SRAM; this reduces DRAM throughput by twice the aggregate throughput of the flows that bypass DRAM. High-occupancy VOQ's must use DRAM to avoid overflowing the SRAM; each such VOQ only needs up to two[3] tail blocks in the SRAM, while the rest of them can migrate to DRAM.

[1]often, DRAM block content will be "aligned" with SRAM block content, and the queue tail portion in the SRAM will start at block boundaries, as illustrated in Fig. 4. However, this is not always the case: in Fig. 2(b), where block size is 256 bytes, assume that the 240-byte segment departs to the crossbar, and, while the 40-byte segment is delayed, the queue grows and its head has to migrate to DRAM; then, 256-byte chunks from the queue head will migrate, and these chunks will not be aligned to SRAM block boundaries.

[2]e.g. [14]: 128-bits wide $\times$ 200 MHz DDR = 51.2 Gb/s -10% turn-around overhead -2% refresh overhead; each block is laid out as four 128-byte sub-blocks, with each sub-block residing in a separate DRAM bank; bank interleaving with 8-word (4-clock) bursts per bank provide peak throughput.

[3]two tail blocks need to stay in the SRAM in case their occupancy is as in Fig. 2(b) and the corresponding segments have to depart to the crossbar before any new packet arrives and is added to them.

B. *Segment Size Adaptivity*

While memory management for the queues in the line card datapaths uses fixed-size multipacket blocks, as in Fig. 1(b), traffic is forwarded to the crossbar in *variable-size* multipacket segments, as in Fig. 1(d). When a segment is forwarded to the crossbar, a crossbar-specific header is prepended to it, containing the crossbar output port ID and the segment length; in our performance evaluations (sec. III), we assumed a 4-byte size for this header. The crossbar buffers and forwards the segment as a unit, ignoring the structure of its contents; in particular, the crossbar does not know the location of packet boundaries, if any, inside the segment. As explained in section I and figure 2, segment sizes (excluding their header) range between a minimum value, normally equal to the minimum packet size, and a maximum value[4], normally equal to the DRAM transfer block size (sec. II-A).

The crossbar-specific per-segment header represents an undesirable overhead for crossbar throughput purposes. To minimize the effects of this overhead, we prefer most segments to have a large size relative to this header. If all segments had 40-byte minimum-IP size, the overhead would be a sizeable 10 percent; if most segments are above 200 bytes, the overhead is reduced below a comfortable 2 percent.

The segmentation scheme introduced in this paper has a nice *adaptivity* property with respect to the above overhead. Under light load, most segments contain a single packet, because packets are forwarded as soon as they arrive. Hence, small packets, which occur frequently in the Internet, generate small segments, which entail higher overhead; However, because the traffic is light, the increased overhead does not matter. Under heavy load, on the other hand, queue occupancy grows; when queue size exceeds one or two blocks, segment size is maximized, due to the multipacket nature of segments. Hence, under heavy traffic, the overhead is minimized and the crossbar operates very close to the maximum efficiency. In conclusion, the crossbar speedup that is needed for line-rate operation under uniform traffic[5] corresponds to the *minimum* value of the overhead, i.e. just 1 to 2 % with 512- or 256-byte maximum segments.

C. *Crosspoint Buffer Size*

Buffered crossbars operating directly on variable size packets need a crosspoint buffer size of at least one round-trip time (RTT) worth of data *plus* one maximum-size packet in order to achieve line-rate operation under single active flow and worst-case packet size conditions [8]. Worst-case conditions are: alternating maximum-size M and small-size S packets, where $S < RTT$ but $M + S > bufferSize$, hence credit is insufficient for both an M and an S packet in the crosspoint. The same effect can occur with variable-size unipacket segments (Fig. 1), thus requiring a buffer size of one RTT *plus* one maximum-size segment.

[4]assumed to exceed twice the minimum value

[5]unbalanced traffic is known to require additional speedup, for orthogonal reasons, though; see section III-D.2

By contrast, the segmentation scheme introduced in this paper yields line-rate operation under single active flow and all packet size conditions with a crosspoint buffer size of just the *maximum of* one RTT worth of data or one maximum-size segement. The reason is that when queue occupancy grows under heavy load, the packets, *independent of their sizes*, are merged into maximum-sized segments, and the crossbar only sees a traffic consisting of such fixed-size units. Under such traffic, a buffer size of just one maximum-size segment suffices (provided it is also larger than one RTT worth of data).

D. Egress Queueing and Reassembly

The egress datapath chip collects segments until a full packet is reassembled for transmission. We assume that packet transmission to the egress port starts when the segment that contains the tail of the packet starts arriving from the crossbar; that is, we assume cut-through operation at the last-segment level. Cut-through operation, at the segment level, is also assumed inside the crossbar, as in [8].

For simplicity and economy, no flow control is needed from the egress line cards backwards: the output schedulers in the crossbars are assumed to operate at egress line rate[6]. A memory space of $N \times P \times MaxPktSize$ per egress port, shared among the reassembly queues, ensures that reassembly buffers do not overflow; N is the number of ingress line cards and P is the number of priority levels (queues) supported.

III. Performance Evaluation

A. Simulation Environment

In order to verify our system we have developed an event-driven simulator in C, operating at a byte time granularity. Except for the buffered crossbar architecture with variable-size multipacket segments (**mps_CICOQ**), we have also modeled the following systems for comparison purposes:

- the buffered crossbar switching variable size, unipacket segments (**ups_CICOQ**)(see Fig. 1(c)),
- the bufferless crossbar operating on fixed-size, multi-packet segments (**mps_CIOQ**)(see Fig. 1(b)),
- ideal output queueing architecture (**OQ**), as a reference system.

In our experiments we assume 32-port switches with the round trip time between the ingress linecards and the switch fabric being 500 byte times. We claim this is a reasonable value including propagation delays, scheduling times and memory accesses.

Every segment forwarded to the crossbar chip has a 4-byte header prepended to it. However, the crossbar lines operate at the speed of external lines, i.e. no speedup is considered. A single priority level is assumed.

The reported delay is averaged over all packets, unless otherwise mentioned. We define the delay of a packet to be the time interval between its first byte arriving to the ingress and its first byte departing from the egress linecard, excluding any constant delays such as propagation times.

[6] to support egress sub-ports, one needs per-subport VOQ's, and: either per-subport flow control from egress to ingress line card, or per-subport crosspoint buffers and output schedulers.

For the bufferless crossbar, we consider the iSLIP scheduling algorithm with five iterations [16]. An input port requests a matching to an output port if the backlog of the corrsponding VOQ is greater or equal to the segment size, or a fixed time interval has passed since the last time it held a partialy filled segment. This interval is controlled by a timer. For the buffered crossbar we assume round-robin service at all contention points: inputs, crosspoints and outputs.

B. Traffic Models

In our simulations we have used the following traffic models :

[-] **Synthetic1500Max traffic :** A synthetic workload resulting from the statistical multiplexing of several streams of packets that simulate application level conversations, such as Telnet or WWW browsing. The distribution of packet size in this mixture is 64% 40-byte packets, 18% 552 or 576-byte packets and 18% 1500-byte packets. The traffic is uniformly destined or unbalanced, according to [4]. Bursts of similarly-destined packets at the sub-flow level, simulate Pareto-sized application level messages.

[-] **Synthetic&JumboFrames traffic :** In order to test our system with ethernet Jumbo Frames, as well, we developed a workload similar to the previous one, but with maximum IP packet size extended to 64KB. The distribution of packets in this mixture is 95% 40-byte packets and 5% packets with size following the bounded Pareto distribution with minimum = 1.5KB, maximum = 64KB, and mean = 10KB. The traffic is uniformly destined.

[-] **MinPkt traffic :** A traffic stream consisting of constant-minimum-size IP packets (40 bytes) with exponentially distributed interarrival times. The destination output ports are uniformly selected.

C. Multipacket segments in CICOQ vs. CIOQ

We explore the performance aspects of the *mps_CICOQ* and *mps_CIOQ* architecture compared to the ideal *OQ*. We simulated *mps_CIOQ* for various timer values and 512-byte envelopes. For the *mps_CICOQ* system we assume 512-byte maximum segment size and 512-byte crosspoint buffer. We fed the systems with the uniform *Synthetic&JumboFrames traffic*; the results are presented in Fig. 5. For the *OQ* cut-through is considered at the output queues, which reduces delay at low loads. On the other hand, our system requires the storage of all the fragments of the packet but the last one. This contributes to a reassembly delay offset which is obvious mostly at light loads. *mps_CIOQ* adds, further to the reassembly delay offset, the timer interval delay. In Fig 5 we observe that the timer interval must be long in order to prevent the release of partially filled envelopes at high loads resulting to increased delay at low loads too.

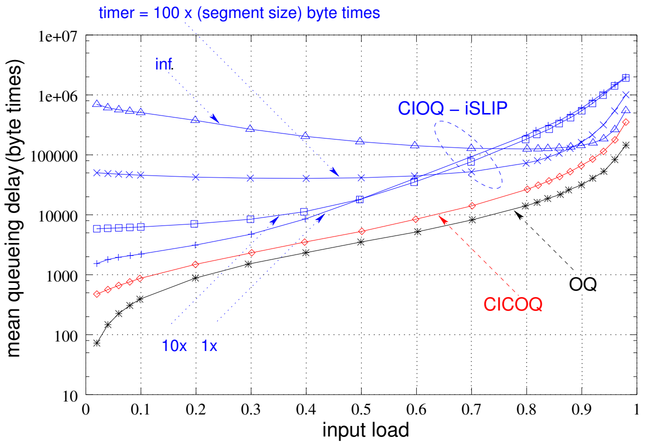

Fig. 5. Buffered crossbar with variable-size multipacket segments. Unbuffered crossbar with fixed-size multipacket segments, 5 iterations iSLIP. Segment size is 512B, crosspoint buffer size is 512B. Synthetic&JumboFrames traffic.

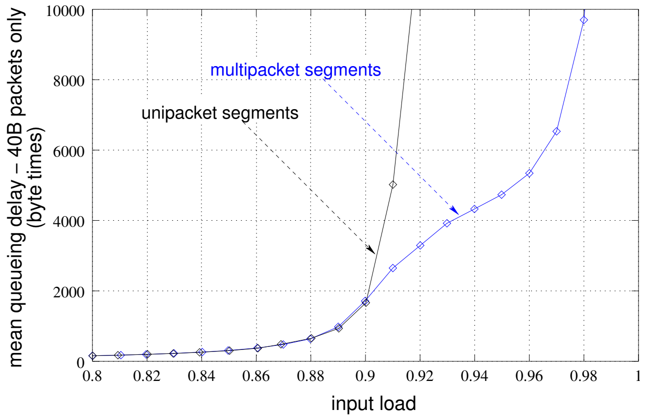

Fig. 6. Buffered crossbar with variable-size multipacket and unipacket segments. Maximum segment size is 512B, crosspoint buffer is 520B. MinPkt traffic.

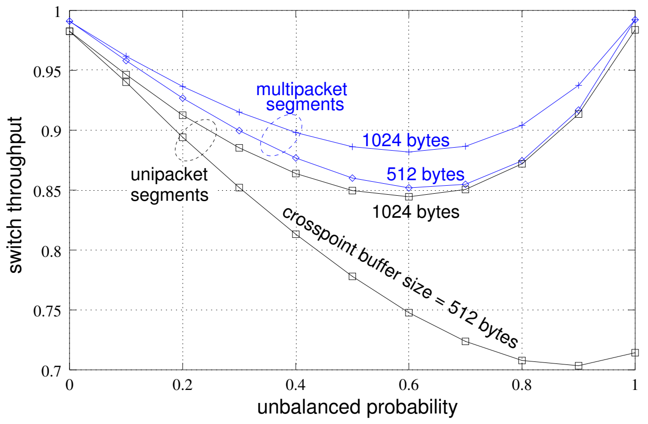

Fig. 7. Buffered crossbar with variable-size multipacket and unipacket segments. Maximum segment size is 512B, crosspoint buffer size is 1 or 2 maximum segment sizes. Unbalanced Synthetic1500Max traffic.

D. Multi vs. Unipacket segments in CICOQ

1) Delay Experiment: In order to verify our adaptivity hypothesis (section II-B), we measured the mean packet delay for the *mps_CICOQ* and the *ups_CICOQ* system, when *MinPkt traffic* arrives at the switch input ports. In both cases, the buffer at crosspoints is 520 bytes [7] and the maximum segment size is 512 bytes. Fig. 6 depicts the results. We observe that, as expected, the *ups_CICOQ* system saturates at a load of $\frac{40}{44} = 91\%$, because the internal header is added to minimum-size packets-segments. However, in the *mps_CICOQ* system the header is added to maximum-size segments and thus the system is stable for all loads up to $\frac{512}{516} = 99\%$. Note that the above (worst-case) scenario is important, considering that today the minimum-size packets constitute a large portion of the real world IP traffic [17].

2) Throughput Experiment: We used the unbalanced *Synthetic1500Max traffic* and measured the switch throughput for the *mps_CICOQ* and the *ups_CICOQ* system; we assume 512-byte maximum segment size, while crosspoint buffer size is 1 or 2 maximum segment sizes. The results are presented in Fig. 7. Observe that the throughput of *ups_CICOQ* with buffer size = 512B is a decreasing function of the unbalanced probability. The reason is that alternating small-large packets in our workload create alternating small-maximum size segments inside the crossbar. So, as the unbalanced probability increases, each output is loaded by mostly a single input, the phenomenon explained in section II-C takes place and switch throughput is reduced. The performance of the *mps_CICOQ* is close to the respective one of the buffered crossbar when the traffic consists of fixed size cells and 1 cell crosspoint buffer is used [4]. Observe that when the unbalanced probability is 1, i.e. each output is 100% loaded by a single input, our system yields full output utilization. When crosspoint buffer size is 1024B, i.e. one maximum-size segment plus the round trip window, the *ups_CICOQ* achieves satisfactory performance, but the *mps_CICOQ* is still superior.

[7]ups_CICOQ needs 13x40=520B buffer space because 12x40 is less than the assumed RTT (500 byte times).

E. Performance as a function of maximum segment size

The choice of the maximum segment size in the *mps_CICOQ* system actually affects several performance factors. Increasing the size of the maximum segment,

i) we reduce the overhead of the internal header addition.
ii) we reduce the packet reassembly delay and thus the total packet delay.
iii) we increase the delay for small packets, due to the larger segments being transmitted ahead of them, at the VOQs or the crosspoint buffers.

Obviously by increasing the segment size we also increase the cost of the crossbar chip. We experimented with the uniform *Synthetic1500Max traffic*. In Fig. 8 we verify the hypothesis (ii); we assume the crosspoint buffer size is one maximum sized segment and, especially for this experiment, the round trip time between the ingress linecard and the crossbar chip is equal to the transmission time of a maximum segment. In Fig. 9 we verify the hypothesis (iii). We consider two segment

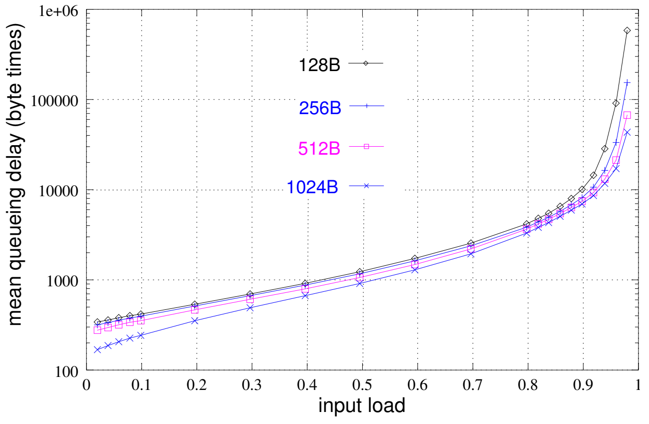

Fig. 8. Buffered crossbar with variable size multipacket segments. Maximum segment size is 128, 256, 512 or 1024B. Crosspoint buffer size equals maximum segment size. RTT equals maximum segment size. Synthetic1500Max traffic.

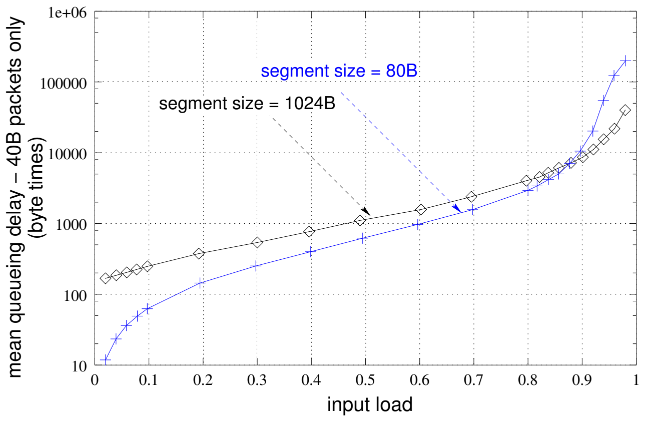

Fig. 9. Buffered crossbar with variable size multipacket segments. Maximum segment size is 80 or 1024B. Crosspoint buffer size is 1024B. Synthetic1500Max traffic. Delay measured for small packets only.

sizes : 80 and 1024 bytes. In both cases the crosspoint buffer is 1024 bytes. Hypothesis (i) is verified in Fig.8 and Fig. 9 as well. Observe that the packet delay at heavy loads increases as the segment size decreases because the induced internal header overhead is greater.

Conclusion

We presented an innovative segmentation scheme for buffered crossbar based switches. The scheme combines the known ability of buffered crossbars to directly operate on variable-size units and the aggregation of multiple packets or fragments of them into each segment. We showed that this scheme is well adapted to the use of DRAM for ingress buffering, and presented an overall system architecture. Through simulation we compared our segmentation scheme to its ancestors and found it superior in terms of packet delay and switch throughput. Lastly, we studied our system performance under various maximum-segment sizes.

Acknowledgments

Euricles Kounalakis, Dionysios Pnevmatikatos, Ioannis Papaefstathiou, and Nikos Chrysos, among others, helped us in this work. This work was performed within the project “Scaleable Intelligent Video Server System (SIVSS)”, supported by the European Union FP6 IST programme (contract 002075).

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] N. Chrysos, M. Katevenis: “Weighted Fairness in Buffered Crossbar Scheduling”, *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR 2003)*, Torino, Italy, June 2003, pp. 17-22.

[7] 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

[8] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, N. Chrysos: “Variable Packet Size Buffered Crossbar (CICQ) Switches”, *Proc. IEEE International Conference on Communications (ICC 2004)*, Paris, France, 20-24 June 2004, vol. 2, pp. 1090-1096.

[9] M. Katevenis: “Small, Variable-Size Packets in Large Blocks”: *exercise 5.2 of course CS-534 (Packet Switch Architecture)*, Univ. of Crete, March 2000; http://archvlsi.ics.forth.gr/∼kateveni/534/00a/exer/ex5.html

[10] K. Kar, T. V. Lakshman, D. Stiliadis, and L. Tassiulas : “Reduced Complexity Input Buffered Switches”, *Proc. HOT Interconnects VIII*, Stanford University, Stanford, CA, August 2000 .

[11] Christensen, Yoshigoe, Roginsky, Gunther: “Performance Evaluation of Packet-to-Cell Segmentation Schemes in Input Buffered Packet Switches”, *Proc. IEEE Int. Conf. on Communications (ICC’2004)*, Paris, France, 20-24 June 2004

[12] A. Nikologiannis, M. Katevenis: “Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques”, *Proc. IEEE Int. Conf. on Communications (ICC’2001)*, Helsinki, Finland, June 2001, pp. 2048-2052 (5 pages).

[13] Sundar Iyer, Ramana Rao Kompella, and Nick McKeown : “Analysis of a Memory Architecture for Fast Packet Buffers” *Proc. IEEE High Performance Switching and Routing (HPSR 2001)*, Dallas, Texas, May 2001, pp. 368-373

[14] Micron DDR2 SDRAM product documentation, available at http://www.micron.com/products/dram/ddr2sdram/

[15] 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.

[16] N. McKeown: “The iSLIP Scheduling Algorithm for Input-Queued Switches”, *Proc. 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

[17] “Cooperative Association for Internet Data Analysis”; http://www.caida.org