HPQ: A High Capacity Hybrid Priority Queue Architecture for High-Speed Network Switches

Imad Benacer, François-Raymond Boyer
Dept. of Computer and Software Engineering
Polytechnique Montréal
Montréal, Canada
E-mail: {imad.benacer, francois-r.boyer}@polymtl.ca

Yvon Savaria
Dept. of Electrical Engineering
Polytechnique Montréal
Montréal, Canada
E-mail: yvon.savaria@polymtl.ca

***Abstract*—This paper presents a fast hybrid priority queue architecture usable in todays high-speed networking devices. This architecture can be used for scheduling and prioritizing packets in the network data plane. Due to increasing traffic, a high capacity priority queue, with constant latency and guaranteed performance is highly needed. In this work, an important goal is to reduce latency to best support the upcoming 5G wireless standards. The proposed hybrid priority queue architecture enables pipelined en/dequeue operations with O(1) time complexity. The proposed architecture is implemented in C++ and is synthesized with the Vivado High-Level Synthesis tool. The reported results show the feasibility of the proposed solutions and are compared across a range of priority queue depths and performance metrics with existing approaches. An implementation of the proposed architecture on a ZC706 FPGA board works at 60 MHz and supports links operating at 10 Gb/s, with a total capacity of ½ million packet tags spread over 512 independent queues.**

***Keywords*—priority queue; networking devices; high-level synthesis; FPGA**

I. INTRODUCTION

With the upcoming next generation cellular communication infrastructure (5G) [1] and the rising number of connected devices, one of the major challenges facing network operators and Internet providers is in scheduling, prioritizing packets of different applications, and routing this traffic in a minimum time. Applications with real-time traffic, such as video streaming, require stringent quality of service (QoS) guarantees. QoS include but are not limited to average throughput, end-to-end delay and bandwidth. To provide QoS guarantees for large number of connected devices, users, etc., high capacity priority queues must be used to maintain real-time sorting of queue elements at link speeds.

Previously reported priority queues have been used in many applications such as event driven simulation [2], scheduling [3], real-time sorting [4], etc. A priority queue (PQ) can be represented as an abstract data structure that allows insertion of new items and extraction in priority order. Different types of PQs have been proposed. Reported solutions span between the following: calendar queues, binary trees, shift registers, systolic arrays, register-based arrays, and binary heaps. In general, PQs can be divided in two classes: priority queues with O(1) time complexity operations, independently of the queue depth (number of nodes), and those with variable processing time.

In many modern routers, switches, line cards, etc., we find Network Processing Units (NPUs). They provide dedicated processing stages for traffic management and buffering. Traffic management includes policing, scheduling, shaping and queuing. For high-speed network switches and devices, queuing may represent a bottleneck. One of the feasible solutions to reduce queuing latencies is to tag the packets. This tagging will hold concise packet information for fast processing, while the actual packets are buffered independently by the NPU, thus reducing the queuing latencies between the different network processing stages [5].

In this work, a high capacity hybrid PQ (HPQ) architecture is proposed. It can contain half a million packet tags of 32-bit in a Field Programmable Gate Array (FPGA). This architecture proposed for high-speed networking devices operates in a constant 2-cycle latency per packet. Unlike existing PQ architectures proposed for different contexts such as traffic managers, task schedulers, sorting, etc. (see Section II), the performance of the proposed architecture is independent of the PQ depth. Also, the proposed HPQ is entirely coded in C++, providing easier implementation and more flexibility than some reported works in literature, which use low-level coding, mostly in Verilog, VHDL, and targeting ASIC implementations.

The remainder of this paper is organized as follows. In Section II, we present a general switch architecture. That switch architecture provides a meaningful context where the proposed HQP would fit. Then we detail some related work on PQs found in the literature. In Section III, we present the architecture of the proposed HPQ. In Section IV, we detail the HLS methodology and considerations leading to the best performances. Section V reports hardware implementation results and compares them to previous work. Finally, Section VI draws conclusions from this work.

II. RELATED WORK

In this section, we present the general architecture of a shared memory switch. Then, we detail the related work about priority queuing for scheduling.

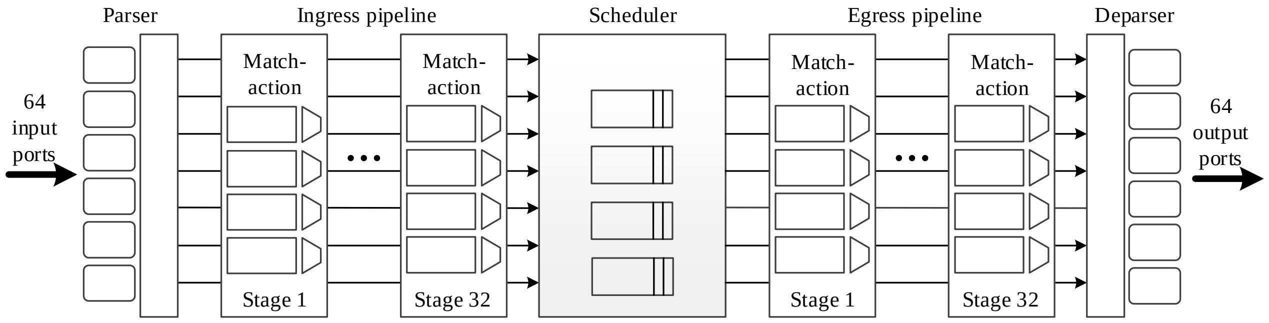

Fig. 1. A general switch architecture with 64 input/output ports [21].

A. Network Switches

Today's switches provide various sets of functionalities, from parsing, scheduling and buffering of the network traffic. These functionalities can be supported by transformations applied to the traffic from the moment packets are received on input ports up to their transmission through destination ports. The architecture of a shared memory switch, with its internal modules, is depicted in Fig. 1. This switch architecture is used by Broadcom in its Trident series [6]. From the requirements of today's networks, switches must run at line rates to make local buffering manageable. For instance, 10 Gb/s is a popular line rate. According to Fig. 1, switches must provide scheduling capabilities. Some of the well-known scheduling algorithms are deficit round robin (DRR) [7], strict priority [8], and traffic shaping [8]. In this work, we are mainly interested in the scheduling functionality through strict priority, while providing large buffering capacity implemented using on-chip memories in an FPGA, from which a high capacity HPQ is proposed.

B. Priority Queuing

Several PQs have been proposed in the literature. These works can be classified as software- and hardware-based solutions. In software based-solutions, we find mainly heaps and binary search trees [9, 10]. However, these implementations cannot handle large priority queues with very low latency, due to the inherent O(log n) complexity per queue operation.

Reported solutions for hardware PQs are based on calendar queues, binary trees, shift registers, systolic arrays, and binary heaps. Moon [11] analyzed four scalable priority queue architectures based on: FIFOs, binary trees, shift registers and systolic arrays. Moon showed that the shift register architecture suffers from heavy bus loading, and that the systolic array overcomes this problem at the cost of doubling the hardware complexity. Meanwhile, the total capacity previously investigated by Moon is 1024 (1 Ki) elements. Also, the hardware approaches that were adopted limit queue size scalability due to limited resources. This motivated the research reported in the present paper to explore alternatives for building large priority queues that can offer high throughput and low latency. The reported solution is a hybrid PQ. Basically, hybrid PQs combine dedicated hardware approaches extended using on-chip or off-chip memories. In this work, we target to use only on-chip memories available in FPGAs.

In [4, 12], a hybrid priority queue architecture is proposed, based on a p-heap (which is similar to binary heap). However, the proposed priority queue supports en/dequeue operations in $O(\log n)$ time against a fixed time for the systolic array and shift register, where n is the number of keys. Also, these two implementations of pipelined PQs offer scalability and achieve high throughput, but at the cost of increased hardware complexity and performance degradation for larger priority values and queue sizes. The reported solutions implemented on ASICs had 64 Ki and 128 Ki maximum queue capacities, respectively. Kumar [13] proposed a hybrid priority queue architecture based on a p-heap implemented on FPGA supporting 8 Ki elements that can handle size overflow. Huang [9] proposed an improvement to the binary heap architecture. Huang hybrid PQ combines the best of register-based array and BRAM-tree architectures. It achieves a performance close to 1 cycle per replace operation. With this solution, the en/dequeue operations are performed in O(log n), and the total capacity is also 8 Ki elements.

Zhuang [14] proposed a hybrid PQ system exploiting an SRAM-DRAM-FIFO queue called a Hybrid Priority Queue System (HPQS), allowing the extension of the PQ using both SRAM and DRAM for increasing its capacity and reducing cost. Chandra [15] proposed an extension of the shift register based PQ proposed by Moon [11] that was implemented using a software binary heap. Bloom [16] proposed an exception-based mechanism used to move the data to secondary storage (memory) when a hardware PQ overflows.

McLaughlin [17] and Wang [18] proposed pipelined heaps using extended SRAM/DRAM. This complex architecture is implemented in ASIC. Afek [19] proposed a PQ using TCAM/SRAM. This author showed the efficiency of the proposed solution and its advantages over other ASIC designs [14, 17], but the overall rate degrades with larger queue sizes because of the software logarithmic complexity.

III. The Hybrid Priority Queue Architecture

In this work, packets are sorted in ascending order of priority based on PQ presented in [20, 22]. The supported queue operations are enqueue and dequeue. An enqueue enables insertion of an element to the queue, while a dequeue

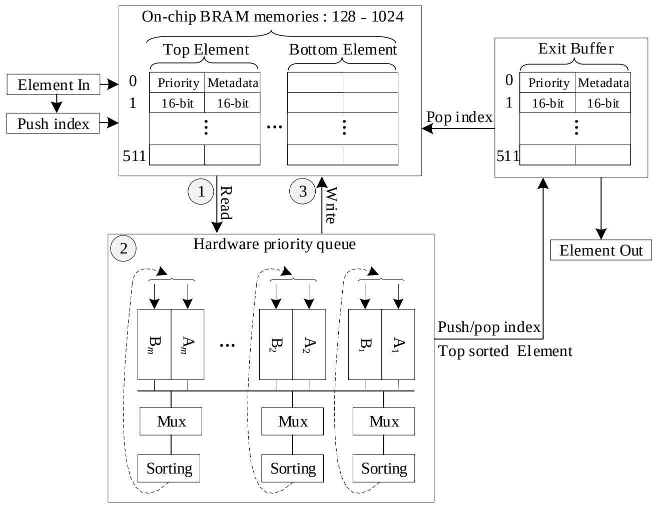

Fig. 2. The proposed hybrid priority queue architecture.

removes the highest priority element (lowest in priority value). Fig. 2 depicts the proposed HPQ architecture. The system architecture assumes one input port representing the operation to perform and the pushed packet tag (*Element In*), and one output port representing the dequeued packet tag (*Element Out*). The HPQ is devised in two parts; the first part is the storage area of the queues implemented with on-chip Block RAMs (BRAMs) while the second part contains the hardware PQ used to sort out packets in a single clock cycle. The whole HPQ design is described at high-level using the C++ language. The code is written in a way that allows efficient hardware implementation and hardware prototyping in a FPGA platform.

From a conceptual point of view, the hybrid PQ is intended to work in a pipelined fashion. Each load/store of one index from all BRAMs can be done in a single clock cycle. The HPQ operates as follows, in the first cycle (see circled numbers on Fig. 2), according to the operation to perform either an enqueue or a dequeue, an index is calculated. This index corresponds to the BRAMs line (together the respective lines of the parallel BRAMs contain 128 to 1 Ki elements, according to the queue capacity) to be loaded onto the hardware PQ. The ordering of the active queue is done in the second clock cycle, while a write back to the same BRAMs line to store the result is done in the third clock cycle. It should be noted that each BRAM holds 512 categories, and each category can have 128 up to 1 Ki elements. This leads to 512 distinct queues with at most 1 Ki capacity. The load and store operations require two distinct buses of 1024 × 32-bit necessary to pass the elements stored in the BRAMs to the hardware priority queue and vice versa. When the HQP is generated with its full queue depth, 2^{16} nets are instantiated, and this impacts performance (more details are given in the experimental results Section V). To use the total capacity of the HPQ, proper priorities repartition is advised.

When a packet tag is received (through *Element In* as depicted in Fig. 2), it contains its priority and required metadata information like the actual packet address, egress destination port (it may contain other attributes). In this work, the priority and metadata lengths are 16-bit each. The configuration of the data type limit can be modified for tag fields. The received packet tag is pushed into the HPQ that is subsequently sorted. The on-chip memory BRAM can hold a maximum of 512 × 32 bit elements. Upon an enqueue operation, the priority of the received packet tag is checked to calculate the line index. This architecture supports 16-bit priority spread over 512 (2^9) distinct independent priority queues. Each line index can be determined by a simple division by 128 (shift by 7 bits). Then, a push is performed after sorting the upcoming tag with the existing elements in the HPQ (load all elements from the BRAMs to the hardware PQ). In the case of a dequeue operation, a parallel comparison is made between the elements stored in the exit buffer. The exit buffer holds the highest priority element of each line, in priority order. A priority encoder finds the index of the highest priority element in the exit buffer, used as pop index. From the corresponding pop index, the content of on-chip memories are sorted to complete the dequeue operation the same way an enqueue operation is performed.

IV. HLS Methodology and Considerations

High-Level Synthesis (HLS) enables raising the design abstraction level while providing more flexibility by automatically generating synthesizable Register Transfer Logic (RTL) from C/C++ models compared to Hardware Description Language (HDL) hand-written designs. Also, HLS require less design effort, when performing a broad design space exploration, as many derivative designs can be obtained with a small incremental effort. In addition, design space exploration can be performed through different available directives and constraints provided by the tool. Using directives, the user can guide the HLS tool during C-synthesis. Thus the designer can focus on the algorithmic design aspects, rather than on low-level details required when using HDL.

The HLS process can be described in two steps. First, extraction of data and control paths from the high-level design files is performed. Second, scheduling and binding of the RTL in the hardware, targeting a specific device library is performed to match available resource in the FPGA. In this work, we performed all experiments with version 2016.2 of Vivado HLS while the design is coded in C++.

The proposed HPQ algorithm was first coded. Then, appropriate design iterations were applied to refine (code optimization and enhancement) the design. The code was thoroughly tested. Finally, design implementation metrics are defined such as the target resource usage, desired throughput, clock frequency, and design latency. We performed a thorough design space exploration for the HPQ through HLS targeting minimum latency, equivalent memory usage (BRAMs) and the highest throughput. The total considered HPQ capacity is 512 Ki. The main metrics used to measure performance in HLS designs are area, latency, and Initiation Interval (II). The partition directive was used to guide the tool to use only logic resources and not the BRAMs available in the FPGA, to reduce design latency by cutting down the memory access time for the hardware PQ and priority encoder. The unroll directive leads to parallelized designs. These directives are used for the hardware PQ. To map the storage to the on-chip BRAMs, the resource directive is used with the option “true dual port RAM”. The pipeline directive is used to target an II of 1 clock cycle. All the

TABLE I. RESOURCE UTILIZATION OF THE TAFFIC MANAGER

Queue depth	Resources / Usage			
	BRAM_18K	***FFs***	***LUTs***	***Latency***
64 Ki 512 × 128	128 / 12%	16594 / 4%	16560 / 8%	2 cycles
128 Ki 512 × 256	256 / 24%	24828 / 6%	41070 / 19%	2 cycles
256 Ki 512 × 512	512 / 47%	41224 / 9%	76133 / 35%	2 cycles
512 Ki 512 × 1024	1024 / 94%	73938 / 17%	150630 / 69%	2 cycles

above directives are used together to generate the HPQ design. More details on the experimental results of placement and routing in the FPGA for different HPQ capacities are provided in the next section.

V. EXPERIMENTAL RESULTS

The proposed HPQ was implemented on a Xilinx Zynq-7000 board (based on the xc7z045ffg900-2 FPGA). The resource utilization of the proposed HPQ architecture for different queue capacities is reported in Table I. Each tag or element of the PQ stores 32 bits, and the hardware queue depth is varied from 128 to 1024, while the HPQ height is 512. In the HPQ implementation, only flip-flops (FFs) and look-up tables (LUTs) were used to obtain a fast pipelined architecture, while the BRAM_18K usage reflects directly the width of the HPQ, as each on-chip memory is mapped to hold an entire column of the proposed HPQ storage (see Fig. 2).

The proposed HPQ supports enqueue and dequeue operations. The number of cycles between successive en/dequeue (hold) operations is 4 clock cycles as reported in Table II. Indeed, each operation takes 2 cycles to finish. This is less than the binary heap [9] and p-heap architectures [12]. The reported shift register and systolic architectures in Moon's work [11] have a latency of 2 clock cycles for en/dequeue. In case of the shift register proposed by Chandra [15], the performance degrades logarithmically. Compared to the p-heap architecture [12], even though it accepts a pipelined operation each clock cycle (except in case of successive deletions), the latency is O(log *n*) in terms of the queue capacity, against O(1) time latency for our proposed architecture.

Even though the hardware PQ is fast, achieving 3.31 ns per operation (see Table II), the BRAMs distribution in the FPGA span many columns. During placement and routing, long net delays tend to be generated by the PQ to BRAMs connections (recall that the full architecture requires 2^{16} nets to connect the BRAMs to the hardware PQ as explained in Section III). This impacts directly the overall performance of the design as the clock period of the whole HPQ is 16.8 ns after routing. However, it enables to provide a total capacity of ½ million elements in a single FPGA, against a few thousands in previously published works. In our proposed HPQ, we achieved a guaranteed performance and latency due to the fixed number of cycles in our design. This number of cycles is independent of the queue depth, unlike the O(log *n*) time for the dequeue operation observed with the heap [14, 17, 18] and binary heap [9], where *n* is the number of nodes (keys).

TABLE II. PERFORMANCE COMPARISON FOR DIFFERENT PRIORITY QUEUE ARCHITECTURES

Architecture	Queue depth	# cycles per hold operation	Clock (ns)	Priority level / Platform	Throughput (Mpps)
Shift register [11]	1024	2 (no replace)	20.0	16 /ASIC	25.0
Systolic array [11]	1024	2 (no replace)	22.2	16 / ASIC	22.5
Shift register [15]	1024	2 (no replace)	4.92	16 / FPGA (Cyclone II)	102
	2048		6.64		75.3
Hybrid register/binary heap [9]	1024	4 : replace 11 : others	~7.5	13 / FPGA (Zynq-7)	Best: 33.3 Worst: 12.1
	2048	2 : replace 11 : others	~8.0		Best: 62.5 Worst: 11.4
	4096	1 : replace 11 : others	13.1		Best: 76.3 Worst: 6.9
	8192	1 : replace 11 : others	15.0		Best: 66.7 Worst: 6.1
Pipelined Heap [12]	16 Ki	2 - 17 (no replace)	5.56	18 /ASIC	Best: 90 Worst: 10.6
PIFO [21]	1024	2: replace 4: others	13.9	16 / FPGA (Zynq-7)	Best: 36.0 Worst: 18.0
Hardware PQ [22]	1024	2 (no replace)	3.31	16 / FPGA (Zynq-7)	151
Proposed HPQ	512 Ki	4 (no replace)	16.8		14.9

The throughput achieved with the proposed solution is 14.9 million packets per second (Mpps) with 512 Ki total capacity. From works depicted in Table II, only some queues with 2 Ki and less capacity have throughputs better than our proposed HPQ. Beyond this capacity, either the designs have problem fitting in the targeted FPGA like in [11, 15, 21], or the throughput degrades under our achieved throughput [9, 12]. Compared to [4, 5, 9, 11, 19], the reported throughput of the HPQ is independent of the queue capacity.

VI. CONCLUSION

This paper proposes and evaluates a new hybrid priority queue architecture intended to support the requirements of todays high-speed networking devices. The proposed HPQ was coded in high-level language and synthesized using Vivado HLS. The resource usage of this implementation is very promising as it supports up to ½ million elements in a single FPGA. Also, the achieved performance is comparable to similar related works in the literature, while supporting 10 Gbps links. The achieved latency is in O(1) time for en/dequeue operations. Future work will explore improving resource usage and performance.

ACKNOWLEDGMENTS

The authors thank the Natural Sciences and Engineering Research Council (NSERC) of Canada, Prompt Québec, and Ericsson Research for financial support. They also thank Normand Bélanger for his suggestions and technical guidance.

References

[1] N. Panwar, S. Sharma, A. K. Singh, “A survey on 5G: The next generation of mobile communication,” Physical Communication, vol. 18, part 2, 2016, pp. 64-84.

[2] R. Brown, “Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem,” Communications of the ACM, vol. 31, no 10, 1988, pp. 1220-1227.

[3] Y. Xu, K. Li, J. Hu, and K. Li, “A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues,” Information Sciences, vol. 270, 2014, pp. 255-287.

[4] R. Bhagwan, and B. Lin, “Fast and scalable priority queue architecture for high-speed network switches,” Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2000, pp. 538-547.

[5] Q. Zhang, R. Woods, A. Marshall, “An on-demand queue management architecture for a programmable traffic manager,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, no 10, 2012, pp. 1849-1862.

[6] Broadcom. “High-Capacity StrataXGS Trident II Ethernet Switch Series,” Web site: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series.

[7] M. Shreedhar and G. Varghese, “Efficient Fair Queuing using Deficit Round Robin,” IEEE/ACM Transactions on Networking, vol. 4, issue 3, 1996, pp. 375–385.

[8] R. Giladi, “Network processors, architecture, programming and implementation”, The Morgan Kaufmann Series in Systems on Silicon, 2008.

[9] M. Huang, K. Lim, and J. Cong, “A scalable, high-performance customized priority queue,” IEEE 24th International Conference on Field Programmable Logic and Applications (FPL), 2014. pp. 1-4.

[10] H. Wang, and B. Lin, “Pipelined van Emde Boas tree: Algorithms, analysis, and applications,” The 26th IEEE International Conference on Computer Communications, 2007, pp. 2471-2475.

[11] S. W. Moon, J. Rexford, K. G. Shin, “Scalable hardware priority queue architectures for high-speed packet switches,” IEEE Transactions on Computers, vol. 49, no 11, 2000, pp. 1215-1227.

[12] A. Ioannou, and M. GH Katevenis, “Pipelined heap (priority queue) management for advanced scheduling in high-speed networks,” IEEE/ACM Transactions on Networking (ToN), vol. 15, no 2, 2007, pp. 450-461.

[13] C. N. G. Kumar, S. Vyas, J. A. Shidal, R. K. Cytron, C. D. Gill, J. Zambreno, and P. H. Jones, “Hardware-software architecture for priority queue management in real-time and embedded systems,” International Journal of Embedded Systems, vol. 6, no 4, 2014, pp. 319-334.

[14] X. Zhuang, and S. Pande, “A scalable priority queue architecture for high speed network processing,” Proceedings of the 25TH IEEE International Conference on Computer Communications (INFOCOM), 2006, pp. 1-12.

[15] R. Chandra, and O. Sinnen, “Improving application performance with hardware data structures,” IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1-4.

[16] G. Bloom, G. Parmerl, B. Narahari, and R. Simha, “Shared hardware data structures for hard real-time systems,” Proceedings of the tenth ACM international conference on Embedded software, 2012, pp. 133-142.

[17] K. McLaughlin, S. Sezer, H., Blume, X. Yang, F. Kupzog, and T. Noll, “A scalable packet sorting circuit for high-speed WFQ packet scheduling,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no 7, 2008, pp. 781-791.

[18] H. Wang, and B. Lin, “Succinct priority indexing structures for the management of large priority queues,” The 17th IEEE International Workshop on Quality of Service (IWQoS), 2009, pp. 1-5.

[19] Y. Afek, A. Bremler-Barr, and L. Schiff, “Recursive design of hardware priority queues,” Computer Networks, vol. 66, 2014, pp. 52-67.

[20] I. Benacer, F.-R. Boyer, N. Bélanger, and Y. Savaria, “A fast systolic priority queue for a flow-based traffic manager,” 14th IEEE International New Circuits and Systems Conference (NEWCAS), 2016, pp. 1-4.

[21] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S. T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, “Programmable packet scheduling at line rate,” In Proceedings of the ACM SIGCOMM Conference, 2016, pp. 44-57.

[22] I. Benacer, F.-R. Boyer, and Y. Savaria, “A fast, single instruction multiple data, scalable priority queue,” IEEE Transactions on Very Large Scale Integation (VLSI) Systems, 2018.