A SCHEDULER ASIC FOR A PROGRAMMABLE PACKET SWITCH

WE DESIGNED A GENERIC, SINGLE-QUEUE SCHEDULER ENGINE FOR USE IN A PROGRAMMABLE PACKET SWITCH/ROUTER TO HANDLE IP PACKETS, ATM CELLS, OR A COMBINATION OF BOTH. COMPRISING 275,000 GATES, THE 0.35-MICRON ASIC IS INCORPORATED INTO A PROTOTYPE PROGRAMMABLE PACKET SWITCH.

L. Louis Zhang
QoS Express Inc.

Brent Beacham
University of Toronto

Massoud Reza Hashemi
QoS Express Inc.

Paul Chow
Alberto Leon-Garcia
University of Toronto

While the Internet is successful in supporting traditional data-only traffic, an integrated services Internet is inevitable with the emergence of new applications such as voice, video, multimedia, and interactive video conferencing. Such an integrated services network should support a wide range of applications with diverse quality of service requirements and traffic characteristics.[1] Provision for quality of service in packet networks in general, and in the Internet in particular, is the focus of most of the recent developments in switching and routing system design.[2]

Packets of sessions belonging to different service classes interact with each other when they are multiplexed at the same output link of a switch. The scheduling algorithm at the switching nodes plays a critical role in controlling the interaction among different traffic streams and different service classes, thus determining the service quality for each stream. A common mechanism is important to control the interaction of all of the service classes, rather than having multiple mechanisms for different classes.

The single-queue switch[3] is an integrated scheduling engine for use for all the output ports of a switch. It can implement a wide range of scheduling schemes, including those that can be translated into a logical relationship between a set of numerical or logical values.[4] Examples are priority-based and rate-based scheduling as well as differentiated services. Originally designed for fixed-length ATM cells, the single-queue switch can support the scheduling of variable-length packets based on a conventional, shared-memory switch architecture.[4]

Here, we present the architecture and implementation of a single-queue-based scheduling engine to support quality of service in high-speed switches and routers. The engine supports hardware- or software-based implementations of Internet Protocol (IP) switches and routers with multicasting capability. A hardware scheduling implementation as well as hardware implementation of multicasting in a switch or router increases performance dramatically.[5] In this system, after the packets have been processed with the routing protocols, they are fed into the queuing and scheduling hardware for appropriate scheduling before transmission. See Figure 1.

We designed and fabricated a scheduler ASIC for use in a programmable packet switch currently under construction in the University of Toronto's Network Architecture Lab.

0272-1732/00/$10.00 © 2000 IEEE

Hardware scheduling engine

The single-queue switch can be used as the scheduling engine in any packet switch and router, particularly IP routers and ATM switches. The specific mechanism of handling the packets also allows mixed-mode operations. For example, the same scheduler can support ATM cells and IP, or other packets.

In this system, minicells represent IP packets. A minicell consists of several fields containing information about the output port, priority, and length of the original packet. An index representing the service class or connection number and a pointer relating the minicell to the original packet in the memory of the switch/router also accompany the minicell.

The tagging unit in the scheduler first tags the minicells so the single queue can schedule the packets. The tagging unit determines a tag based on the information in the minicell: class or connection index, priority, and possibly the packet size. Tagged minicells are then sent to the single-queue switch where they are scheduled into other queues. At each time slot dedicated for an output port, the scheduler outputs the minicell that should be serviced next.

As shown in Figure 2, the scheduler also has two other parts associated with it: the multicast handler stores the destination lists of the multicast packets, and the discard handler collects the packets that are discarded in the scheduler as a result of aging or buffer management. The multicast and discard handlers share logic and can be combined to form one unit called the multicast/discard cell handler, or M/D handler.

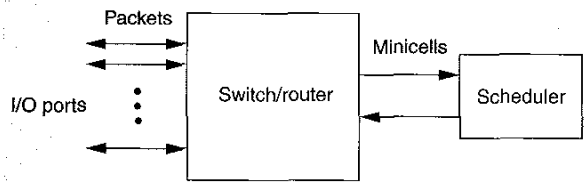

Figure 1. The scheduler in a switch/router.

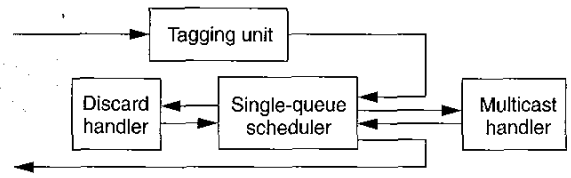

Figure 2. The system architecture.

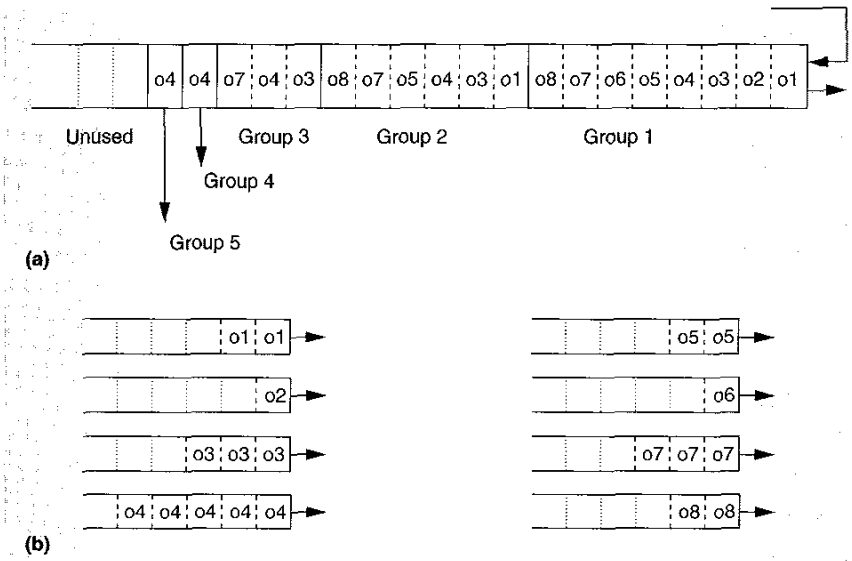

Figure 3. Grouping algorithm for physical (a) and logical (b) queues.

The single-queue scheduler

As shown in Figure 3, the scheduler's grouping algorithm puts the minicells of each output port in a separate logical queue, interleaving each logical output queue in the same physical buffer.[3]

The minicells are organized in a series of groups with the first group containing the first minicell of each output queue. Similarly, the second group contains the second minicell of each output queue and so on. If a group does not have a minicell for an output port, the scheduler does not reserve buffering space for the missing minicell in that group. However, when a minicell arrives for that output port, the scheduler inserts it in the correct place,

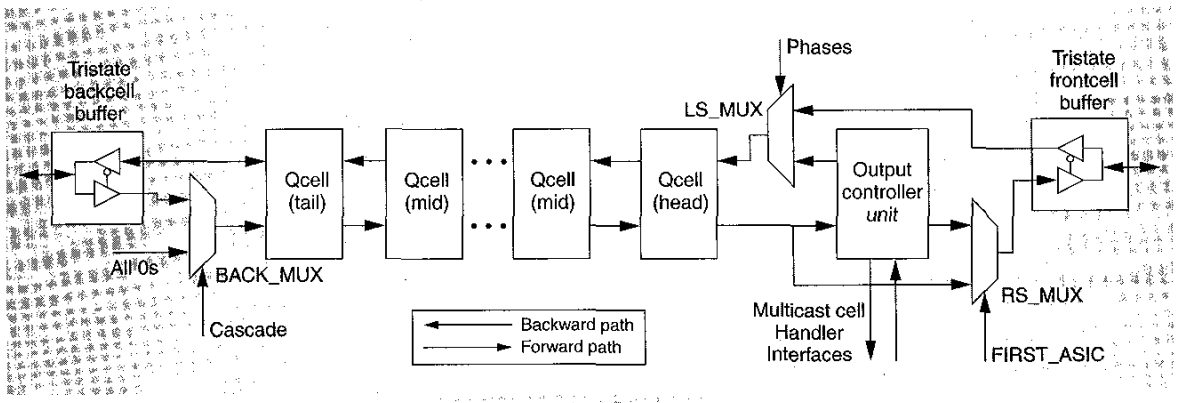

Figure 4. ASIC block diagram.

pushing the rest of the minicells one step back. The interleaving mechanism is based on the underlying physical architecture of the single-queue scheduler. The single-queue scheduler is in essence a generalized sequencer circuit.[6]

The generalized sequencer circuit is a chain of similar buffering and comparison units. Data can travel between the adjacent units in forward and backward directions, as seen in Figure 4. Each unit can store two minicells. The tags of the two minicells are compared, and the winner is sent to the forward output, while the loser is sent to the backward output. This comparison and forwarding takes place at every cycle, simultaneously in every unit.

A new minicell enters the sequencer from the head of the queue, and travels inside the queue step by step to find its correct place in the queue. A new minicell can enter the queue at each step regardless of the situation of the previous minicells traveling in the queue. Similarly, at each step, a minicell leaves the queue, minimizing the latency of the queue to the latency in the first unit only. In this procedure each time slot dedicated to an output line will have one backward shift and one forward shift for the basic operation of the single-queue scheduler. A blocking mechanism can stop the forward shift if necessary. In the case of blocking, the winner of the comparison is kept in the same unit instead of being sent to the forward output at each unit. Blocking is used for empty logical queues in the group.

The single-queue scheduler uses the output port number of each minicell for comparison and the grouping algorithm that determines the logic of comparison to achieve interleaving of logical output queues. At the same time, it uses the tag of each minicell and secondary comparison logic—which should be designed based on the required scheduling algorithm—to appropriately order the minicells in each logical output queue.[3]

Multicasting

To implement multicasting in the single-queue switch, we used an extra logical queue for multicast minicells. At the beginning of each round, the scheduler fetches a multicast minicell for the I/O controller. During the remaining time slots of the round, a copy of the multicast minicell is sent to the output if it is in the destination list of the multicast minicell and if the head-of-line unicast minicell for that output port has lower priority. Otherwise, the scheduler sends out the unicast cell. In either case the loser, which is either the unicast minicell or the copy of the multicast minicell, recycles back to the queue. We introduced an extra backward shift for the recycling path.

Variable-length packets

For variable-length packets such as IP packets, fixed-length minicells are used, but the service times of the minicells are not fixed. During a packet's transmission to an output port, it is important not to service a logical output queue. In the single-queue switch, one group is serviced at each round of serving the output lines; however, the head-of-line minicell of a logical output queue must return to the queue if the output port is not ready. An I/O control unit at the head of the single-

queue switch loops such minicells back into the queue. Based on the mechanism of grouping and sequencing, the looped-back cell finds its correct place in the new head-of-line group, pushing its relative logical queue one logical group back.

Architecture

We designed the programmable scheduler ASIC for a 16 × 16 switching system. (See Figure 4 again.) There are 110 unit cells (Qcells) cascaded to buffer and queue the minicells. The chip can store 220 minicells with each Qcell containing two sets of storage elements. The output controller unit implements the logic for handling the multicast cells. The maximum queue length can be easily expanded by adding more Qcells within the limits of the chip's process technology, or by cascading ASICs to form an ASIC chain. Therefore, part of our design goal was to make both the Qcell and the scheduler ASIC modular and cascadable.

In our design, the time slot dedicated to an output line is divided into three cycles (or phases): two backward shifts (or left shift) and one forward shift (or right shift). The cells enter the Qcells in the backward path through the left-shift multiplexer (LS_MUX), as shown in Figure 4. The LS_MUX selects either the new incoming minicell or the feedback cell, depending on a particular phase. The feedback cell is the loser of the multicast cell and the unicast cell, as determined by the output controller unit. This unit is only used for the head ASIC of the ASIC chain. The right-shift multiplexer (RS_MUX) is used for the head Qcell winner in all but the lead ASIC of the chain to bypass the output controller unit. The BACK_MUX shifts cells from the downstream ASIC, except for the last ASIC of the chain where an invalid cell (all zeroes) is shifted in. These three multiplexers provide the flexibility of cascading the same ASIC anywhere in the ASIC chain with no impact on switch operation. Operation speed is completely independent of the buffer size within an ASIC.

Qcell unit

This unit is one of the two basic building blocks of the scheduler ASIC. Figure 5 shows its structure, which is quite simple and highly modular and cascadable. The Qcell is composed of two pairs of multiplexers, one pair of Agers, one pair of D flip-flops for storage elements, and a Qcell comparator. The phases of the system control the Top_MUX and the Bottom_MUX. During the backward shift phases, the Top_MUX selects the top incoming cell and directs it to the Qcell, while the Bottom_MUX selects the winner of the Qcell from the previous phase. During the right-shift phase, the Bottom_MUX selects the bottom incoming cell, while the Top_MUX selects the loser of the Qcell from the previous phase.

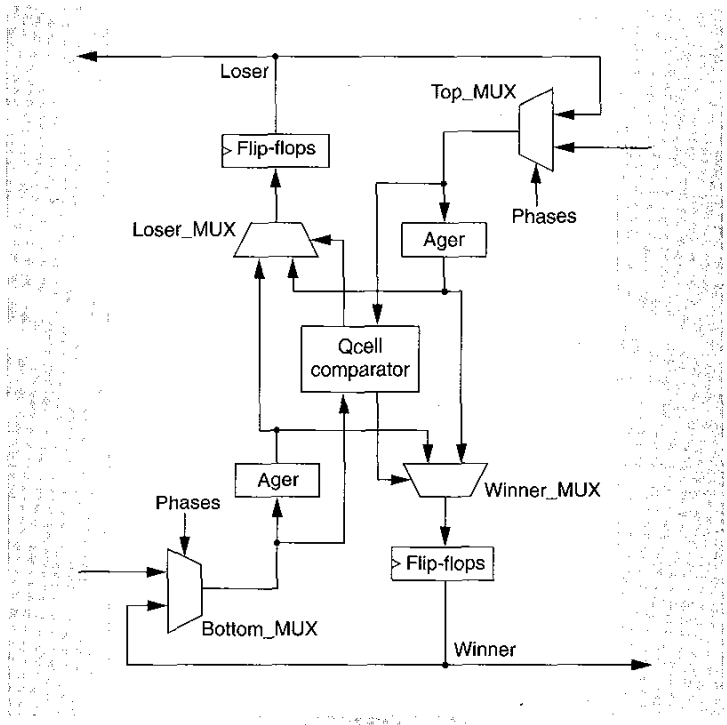

Figure 5. Qcell unit.

The D flip-flops are storage elements that store the winner or loser of the logical comparison. The pair of Agers decrement the age of the minicell when enabled. The Qcell comparator is the heart of the Qcell and contains all the logic to manage the logical queues. The comparator examines the two cells selected by the Top_MUX and the Bottom MUX, and instructs the Winner_MUX and the Loser_MUX to control the directions the two cells should travel. To achieve higher speeds, the inputs to the Agers rather than the outputs of the Agers are fed into the Qcell comparator so that the comparator and the Agers

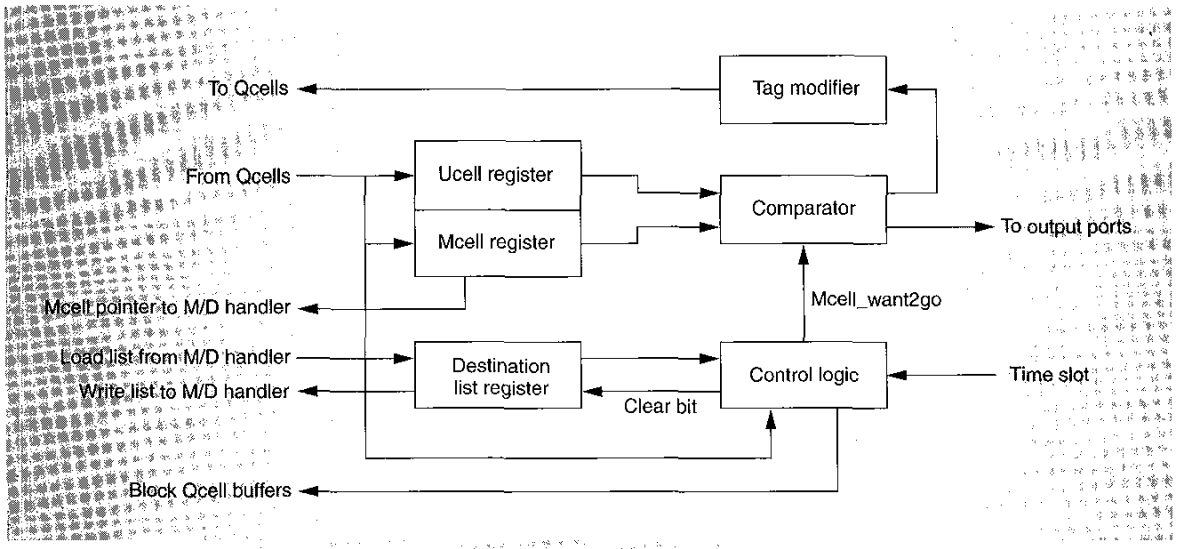

Figure 6. Output controller.

can operate in parallel. Each Qcell contains approximately 2,400 gates.

Output controller

The output controller adds multicasting capability to the scheduler engine. Figure 6 shows the block diagram of the output controller.

As shown in Figure 3, the order of the cells of logical queues in each group is based on their output port numbers ($1 - N$, where N is the number of output ports), and output port 0 (or time slot 0) is used for the multicast queue. At time slot 0, a multicast cell is shifted in from the Qcells and is stored in the Mcell register of the output controller, as shown in Figure 6.

The destination list containing the bit-mapped list of the destinations of the multicast cell is loaded from the M/D handler to the destination list register. During the other time slots, a unicast cell is shifted in from the head Qcell and is stored in the Ucell register. In the Qcells, at each time slot, the output controller scans the head-of-line group. If the head-of-line slot for the current output line (including the multicast line at slot 0) is empty, the output controller blocks the Qcells, and the queue is not read out for that time slot. During the dedicated time slot of each output line, if the output line is among the destinations of the multicast cell (indicated by the signal Mcell_want2go), the head-of-line unicast cell for the output line is compared to the multicast cell.

The comparator inside the output controller compares the multicast cell with the unicast cell based on their priority and age fields. The winner is then sent to its related output line in a synchronous TDM (time-division multiplexing) fashion. The loser is recycled back to the Qcells just as the new incoming cell, but during the second backward shift cycle. If the loser is a multicast cell, a copy of the multicast cell with its output address field set to the corresponding time slot is recycled back to the Qcells. The minicell is set as a copy cell. Then, when it comes back to the output controller again and wins, it will instruct the M/D handler to clear that bit in the destination list of the original multicast cell. If the winner is a multicast cell, the output controller itself clears that bit in the destination list of the multicast cell. The destination lists of the multicast cells must be cleared so that new packets can reuse their corresponding memory locations. The output controller has approximately 5,600 gates.

Implementation and results

Although a full-custom approach would have produced a better design in terms of speed and area, we adopted an ASIC design flow to achieve a shorter design cycle and

Table 1. Prototype chip characteristics.

Characteristic	Description
Technology	TSMC three-layer-metal, 0.35-micron CMOS
Power supply	3.3 V
Core size	8.15 mm × 7.66 mm
Die size	9.65 mm × 9.36 mm
Gate count (transistor count)	275,000 standard cell gates (or 1.1 million transistors)
I/O pads	208

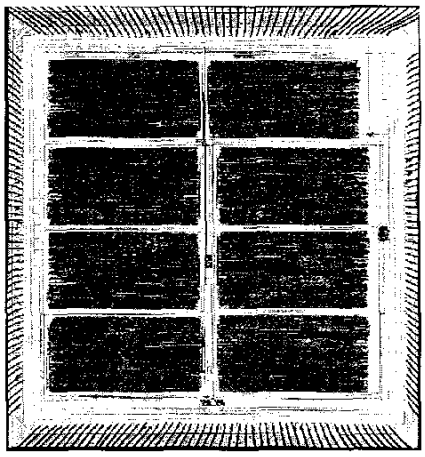

Figure 7. Chip microphotograph.

more flexibility for future design changes. We wrote RTL VHDL code for the scheduler ASIC as well as the tagging unit and the M/D cell handler. We also designed a C program in parallel and compared its output against the VHDL simulation output to provide an extra level of confidence in the correctness of the design. We used a Synopsys[7] synthesis tool and a Cadence[8] place-and-route design tool. We used Cadence Pearl,[9] the physical design static timing analyzer, to check the final timing of the physical design.

The scheduler ASIC was fabricated in TSMC's (Taiwan Semiconductor Manufacturing Co., Ltd.) 0.35-micron process. Table 1 summarizes its key characteristics, and Figure 7 is a microphotograph of the chip. We built a simple PCB that includes one scheduler ASIC and one Xilinx FPGA to test the fabricated chip. We programmed the Xilinx FPGA with the M/D handler circuits to provide some system-level testing. To provide test vectors, we used an HP VXI tester.[10] Extensive tests indicate that the ASIC is fully functional at 20 MHz, the highest speed at which the University of Toronto HP VXI tester can run. We designed the ASIC to target only 30 MHz, as required by our prototype switch. The core of the final design runs at 80 MHz, as reported by the Synopsys static timing analyzer and Cadence Pearl.

Our plan is to initially incorporate a number of the scheduler ASICs in a programmable packet switch currently being developed at the University of Toronto. In our original design, we noted that each time slot of the output line consists of three phases: two backward shifts (one for the incoming cells and the other one for the recycle back path due to multicasting) and one forward shift to output minicells. As mentioned earlier, we need an extra loop-back path to accommodate the variable-length packets. By adding a simple circuit outside the ASIC to loop back the minicell, we can use the same ASIC without modification for scheduling variable-length packets.

The simplicity and capability of our patent-pending scheduler design makes it a cost-effective solution for providing quality of service in a wide range of switches and routers. With slightly more aggressive design and technologies, we believe a single-chip, 120-MHz scheduler ASIC containing over 1,000 buffer spaces could deliver 30 million packets per second. MICRO

Acknowledgments

We thank Bhupinder Parhar, Phong Nguyen, and Dan Sladic for their contributions during the early stages of the chip design. We also thank the Canadian Microelectronics Corporation for its fabrication, infrastructure, and technical support and Jaro Pristupa for his local CAD support, which we very much appreciate.

References

1. P. White and J. Crowcroft, "The Integrated Services in the Internet: State of Art," *Proc. IEEE*, IEEE Press, Piscataway, N.J., Vol. 85, No. 12, Dec. 1997, pp. 1934-1946.
2. S. Keshav and R. Sharma, "Issues and Trends in Router Design," *IEEE Communications*, Vol. 36, No. 5, May 1998, pp. 144-151.
3. M.R. Hashemi and A. Leon-Garcia, "The Single-Queue Switch: A Building Block for Switches with Programmable Scheduling," *IEEE J. Selected Areas in Communications*, Vol. 15, No. 5, June 1997, pp. 785-794.

4. M.R. Hashemi, "Scheduling Engines for ATM Switches," PhD thesis, Dept. of Electrical and Computer Eng., Univ. of Toronto, June 1998.
5. D. Decasper et al., "Router Plugins: A Software Architecture for Next Generation Routers," *Proc. IEEE Sigcom 98*, IEEE Press, 1998, pp. 229-240.
6. M.R. Hashemi and A. Leon-Garcia, "A General Purpose Cell Sequencer/Scheduler for ATM Switches, *Proc. IEEE Infocom 97*, IEEE Press, 1997, pp. 23-30.
7. *Design Compiler Reference Manual*, Synopsys Inc., Mountain View, Calif., 1998.
8. *Preview Cell Ensemble Reference*, Cadence Inc., San Jose, Calif., 1997.
9. *Pearl User Guide*, Cadence Inc., 1997.
10. Hewlett-Packard Company, *Model D20 E1496A Reference Manual*, 1994.

L. Louis Zhang is a cofounder of QoS Express Inc. in Toronto, Ontario, Canada. He has worked at the Network Architecture Lab of the University of Toronto on a programmable switch and at Hewlett-Packard as a key designer of HP's first digital camera ASIC. His research interests include VLSI design in the communications network area. Zhang received his BASc degree from the University of Waterloo and his MASc degree from the University of Toronto, both in electrical engineering.

Brent Beacham is currently working toward a MASc degree in the Electronics Group at the University of Toronto. His current area of research is in high-speed chip interconnection. Beacham received his BASc degree in electrical engineering from the University of Waterloo. He is a student member of the IEEE.

Massoud Reza Hashemi is a cofounder of QoS Express Inc. He is also a postdoctorate fellow at the Network Architecture Lab of the University of Toronto, where he is the architect of a programmable switch. He has made numerous contributions in the areas of ATM and packet switch technology and has two pending patents. Hashemi received his PhD degree in electrical engineering from the University of Toronto.

Paul Chow is an associate professor in the Department of Electrical and Computer Engineering at the University of Toronto. He is currently on leave with QoS Express Inc., which he cofounded. He is also chair of the Technical Advisory Committee for the Canadian Microelectronics Corporation. Previously, he was a major contributor to early RISC microprocessor work at Stanford University. Chow received a BASc degree in engineering science, and MASc and PhD degrees in electrical engineering from the University of Toronto. He is a member of the IEEE and the ACM.

Alberto Leon-Garcia, a professor in the Department of Electrical and Computer Engineering at the University of Toronto, is currently on leave with QoS Express Inc., which he cofounded. His research interests are in switch architectures and traffic management. He is the author of *Probability and Random Processes for Electrical Engineering* and coauthor of the upcoming textbook, *Communication Networks: Fundamental Concepts and Key Architectures*. He is an IEEE Fellow and holds the Nortel Institute Chair in network architecture and services.

Direct comments about this article to the authors at the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 3G4; {lzhang, brent, pc}@eecg.toronto.edu and {hashemi, alg}@nal.toronto.edu.