A small high-bandwidth ATM switch

Adisak Mekkittikul, Nick McKeown, Martin Izzard*

Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, CA 94305-4070

*Core Network Communications Laboratory
DSP R&D Center, Corporate Research & Development Technology
Texas Instruments, Incorporated, PO Box 655474, MS446, Dallas, TX 75265

ABSTRACT

The *Tiny Tera* is an all-CMOS 320 Gbps, input-queued ATM switch suitable for non-ATM applications such as the core of an Internet router. The *Tiny Tera* efficiently supports both unicast and multicast traffic. Instead of using optical switching technology, we achieve a high switching-bandwidth by using less expensive and proven CMOS technology. Because of limitations in memory and interconnection bandwidths, we believe that to achieve such a high-bandwidth switch requires an innovative architecture. By using Virtual Output Queueing (VOQ) and novel scheduling algorithms, the *Tiny Tera* will achieve a maximum throughput close to 100% without the need for internal speedup.

Keywords — ATM switch, packet switch, scheduling algorithms, input-queueing, VOQ.

1. INTRODUCTION

The demand for network bandwidth is growing at a phenomenal rate; the Internet continues to grow in both the number of users and per-user traffic with no apparent end in sight. Due to the growing demand, high-speed networks such as ATM have gained much attention.

In an attempt to provide industry with novel switching techniques, we are developing and building the *Tiny Tera*: a small, high-bandwidth switch. The *Tiny Tera* switch has 32 ports, each operating at the OC-192 rate of approximately 10 Gbps and switches ATM cells or fixed-size packets with a length equal to any multiple of 64 bytes. The switch distinguishes four classes of traffic, and includes efficient support for multicast. We aim to demonstrate that it is possible to build a compact switch with an aggregate bandwidth of 320 Gbps built entirely from currently available CMOS ASIC technology, using a central core no larger than a can of soda.

1. This work is supported by Texas Instruments, Inc. and the Center for Telecommunications at Stanford University.

Increasingly, memory and interconnection bandwidths are becoming bottlenecks in high-bandwidth switches. However, our aim here is not to rely on more advance technology than is currently available, but to make efficient use of the bandwidths.

Input-queueing and parallelism provide the key to such a high-bandwidth switch. Input-queueing reduces the memory bandwidth requirements; when cells are queued at the input, the buffer memories need run no faster than the line rate, i.e., there is no need for the speedup required in output-queued switches. However, the long-standing view has been that cells in an input-queued switch suffer poor performance due to *head of line* (HOL) blocking [2]. We have developed novel scheduling algorithms to reduce the effects of HOL blocking for both unicast and multicast traffic. For unicast traffic, we use a well-known buffering scheme called Virtual Output Queueing (VOQ) [1] in which each input maintains a separate queue for each output. Motivated by DEC's PIM algorithm [1], we use a novel fast, fair and efficient scheduling algorithm called *i*SLIP, that achieves a throughput close to 100% [6] [7], yet is able to make a scheduling decision in just 40ns. For multicast traffic, we are developing algorithms based on the ideas of fanout-splitting and residue concentration [3].

The rest of this paper is arranged as follows. In Section 2 we describe the *Tiny Tera* switch architecture, with an emphasis on the main switch datapath, the bit-sliced crossbar and the architecture of the ports. In Section 3 we outline the queueing policies and scheduling algorithms used for unicast and multicast traffic.

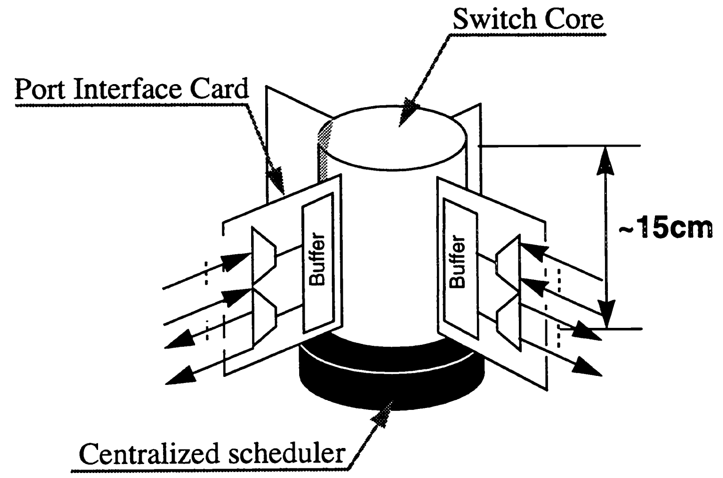
2. SWITCH ARCHITECTURE

The switch consists of three main parts: 32 ports, a parallel bit-slice switching fabric and a scheduler as shown in Figure 1(a). The cylindrical-shaped switching fabric is situated at the center of the switch allowing all ports to be attached radially. The central scheduler is located below the switching fabric. Cells are buffered upon arrival at the input ports, which then generate requests to the scheduler. Based on the requests, the scheduler selects a configuration for the switching fabric, creating a set of conflict-free connections between input and output ports.

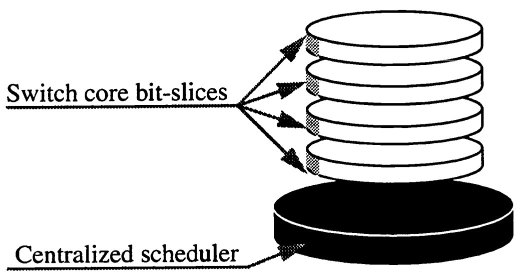
2.1. Port architecture

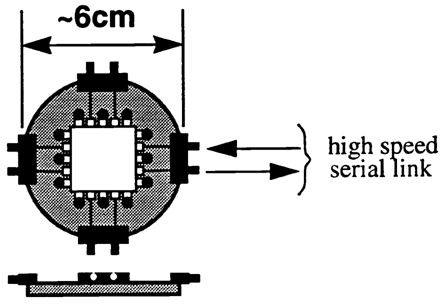
In order to operate at 10 Gbps line rate, the *Tiny Tera* port is divided into two parts, as shown in Figure 2: an ATM-independent datapath consisting of data slices and SRAMs, and an ATM-dependent port processor. The primary function of the datapath portion is to perform simple but high bandwidth tasks such as buffering, forwarding and receiving cells. More complex ATM-specific tasks such as VCI lookup, traffic and queue management, which do not involve a high bandwidth data transfer, are handled by the port processor. In addition, the data rate and cell size of the *Tiny Tera* port are both designed to be scalable so that the switch can easily support non-ATM applications. The basic switching unit is 64-bytes — all cells must be the same length which can be any multiple of 64 bytes.

Shown in Figure 2, cells arrive from the external interface in parallel over eight high-speed serial links [8], each operating at a nominal rate of 1.5 Gbps. This external interface is a higher-speed version of the ATM-Forum UTOPIA interface [10]. A 64-bit portion of



a) The switch consists of a central, vertical hub. Each port interface card connects radially into the hub. This example shows a 4-port switch. Each port interface card may be connected to multiple external lines.





b) The central switching core is comprised of a number of identical bit-slices. This example shows a 4-slice core.

c) Each bit-slice contains a single switch bit-slice chip. This example shows a single 4x4 bit chip.

Figure 1 Architecture and detail of the *Tiny Tera* bit-slice, parallel switch.

the cell — an ATM cell plus some additional switch-specific control bits— is buffered by each of data slice chips. We call this 64-bit unit a *chunk*. Each data slice sends the header portion of the arriving chunk to the port processor. After VCI, VPI, and output port look-ups from the header of the arriving cell, the port processor sends the data slices a new header and a memory address to which the cell to be written into the SRAMs. The SRAMs are dynamically partitioned into FIFO queues as shown in Figure 3. The port processor maintains the head and tail pointers of every queue.

Through another serial link, every port processor communicates with the scheduler, informing it of newly arriving cells. Once the scheduler has decided which output an input can forward a cell to, i.e., which queue to transmit from, it informs the port processor of the decision. The port processor then issues a read request to the data slices, indicating which cell is to be dequeued from the memory. The cell is then forwarded over eight serial links to the switching fabric where it is routed to the destination output port.

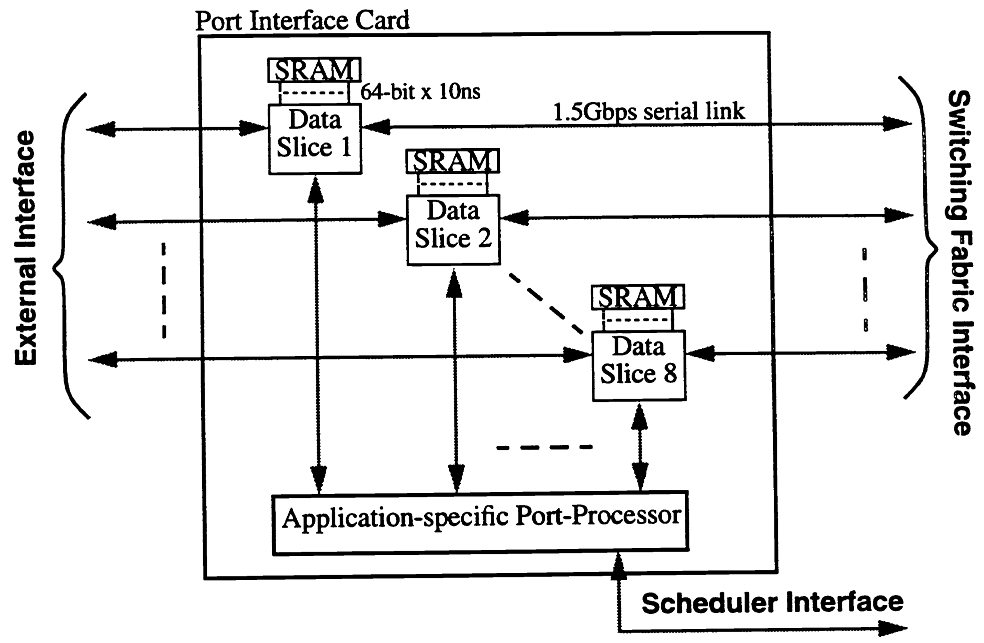


Figure 2 Architecture of port interface card. The data slice components are application-independent, and switch 64-bit *chunks*. The port processor is application-dependent and processes, for example, ATM cell headers or IP addresses.

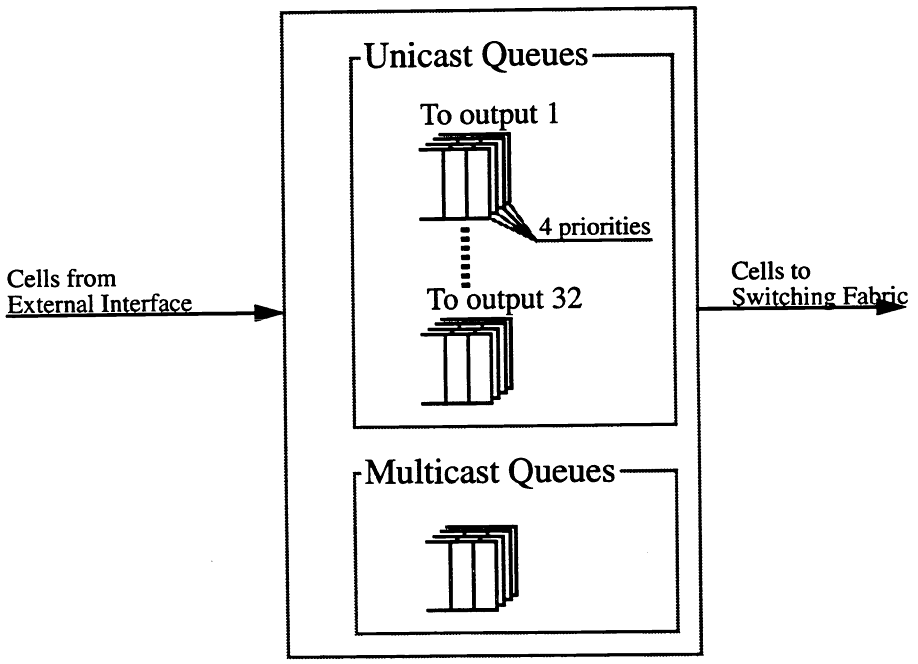


Figure 3 Input-queueing structure at each port interface.

Cells emerging from the switching fabric are once again buffered by the data slices. Under the control of the port processor, the cells are stored into output queues which share the same SRAMs as the input queues. Cells removed from the output queues leave the system over the external parallel interface.

2.2. Parallel bit slice switching fabric

Similar to the port architecture, the 32×32 non-blocking switching fabric achieves a 10 Gbps rate using several bit-slices of crossbars in parallel, each running at the line rate of 1.5 Gbps. As shown in Figure 1(b), the switching hub comprises a stack of identical bit-slices. Plan and side elevations of each bit-slice are shown in detail in Figure 1(c). Each bit-slice is a simple printed circuit board containing a 1-bit 32×32 switching chip; Figure 1(c) shows a simplified example of a 4×4 switch. The bit-slices are connected to each port interface card using high-speed chip-to-chip communication links [8] operating at approximately 1.5 Gbps.

Every cell-time, each port sends the switching fabric a five-bit *reverse-path* routing tag. The tag tells an output which input it is connected to. Note that if a forward-path routing was used, a 32-bit routing tag would be needed for multicast cells.

The advantages of this switching fabric are:

- The bit-slice is extremely simple. No leads need to cross, reducing crosstalk and allowing the slice to be constructed from a simple printed circuit board.
- The lead lengths connecting each port to the central switch are all of identical and minimum length. This reduces skew and the effect of reflections, enabling high data rates to be achieved. It also means that each bit-slice can be very small. The *Tiny Tera* slices will be just 2 inches in diameter.
- Extremely high aggregate bandwidths are achievable by switching multiple bits in parallel. As a result, the switch can be easily scaled for different throughput requirements by simply selecting the number of bit-slices.

In addition, the crossbar switch is capable of routing multicast cells by simply connecting a single input to multiple outputs. The *Tiny Tera* takes advantage of this built-in capability in order to achieve a high throughput for multicast traffic.

3. SCHEDULING CELLS

A non-blocking switching fabric, e.g., a crossbar switch, requires a scheduling algorithm to select a set of per-cell-time conflict-free connections. The *Tiny Tera* maintains two separate scheduling algorithms: one for unicast traffic and one for multicast traffic.

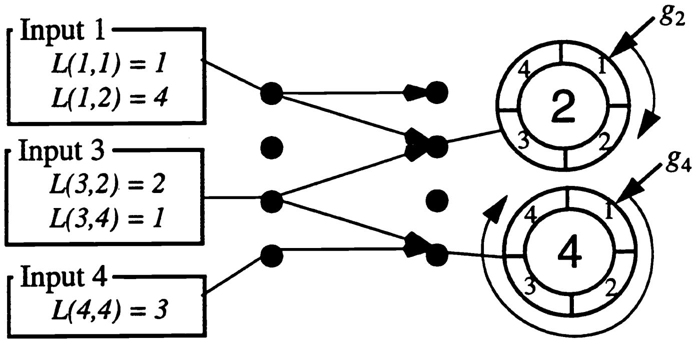
3.1. Scheduling unicast traffic

Input-queueing has been widely believed to suffer from low throughput because of HOL blocking, yet HOL blocking can be entirely eliminated by a simple buffering scheme called Virtual Output Queueing (VOQ). With VOQ, each input maintains a separate FIFO queue for each output, instead of a single FIFO for all outputs. Although slightly more complex (for unicast traffic, an $N \times N$ switch now maintains N^2 input FIFOs), note that no additional memory bandwidth is required; at most one cell can arrive and depart from each input in a cell time.

When VOQ is used, the switch requires a scheduling algorithm [6] that examines the contents of the N^2 input-queues at the beginning of each cell time, deciding which ones will be served. A good scheduling algorithm must be fast, simple, fair, and efficient. (e.g. if the input-lines operate at 10 Gbps, take an ATM cell as the worst case, the scheduling algorithm must make its decision within 42ns).

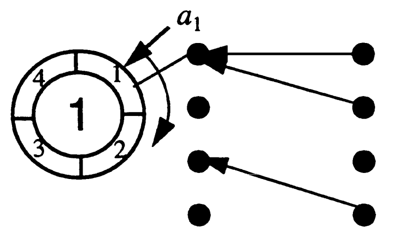
In previous work, we have shown that the maximum size bipartite matching algorithm and the maximum weight bipartite matching algorithms (the longest queue first (LQF) and the oldest cell first (OCF)) can achieve 100% throughput [7]. Unfortunately, these algorithms are known to be impractical for implementation in fast and simple hardware and require a running time of complexity $O(N^3 \log N)$ [9].

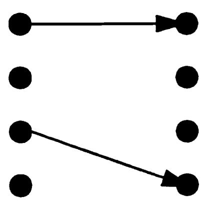
However, we have developed practical, heuristic scheduling algorithms: *i*SLIP, *i*LQF, and *i*OCF. The *i*SLIP is an iterative algorithm that provides high efficiency for best-effort traffic and yet is simple to implement in hardware. *i*SLIP achieves fairness using independent round-robin arbiters at each input and output. Simple round-robin arbiters [1] experi-



a) Step 1: *Request.* Each input makes a request to each output for which it has a cell.

Step 2: *Grant.* Each output selects the next requesting input at or after the pointer in the round-robin schedule. Arbiters are shown here for outputs 2 and 4. Inputs 1 and 3 both requested output 2. Since $g_2 = 1$ output 2 grants to input 1. g_4 is successful in Step 3 and is updated to favor the input after the one that is granted.





b) Step 3: *Accept.* Each input selects at most one output. The arbiter for input 1 is shown. Since $a_1 = 1$ input 1 accepts output 1. a_1 is updated to point to output 2.

c) When the arbitration has completed, a matching of size two has been found. Note that this is less than the maximum sized matching of three.

Figure 4 Example of *one iteration* of the *i*SLIP algorithm.

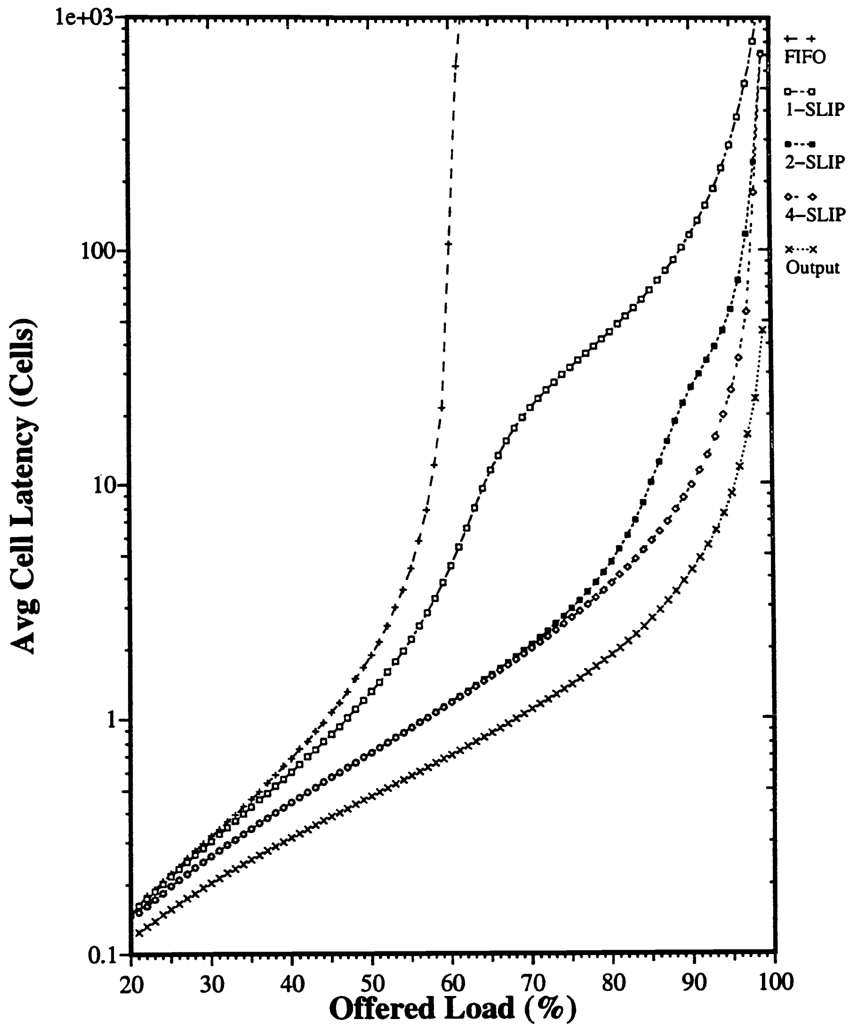
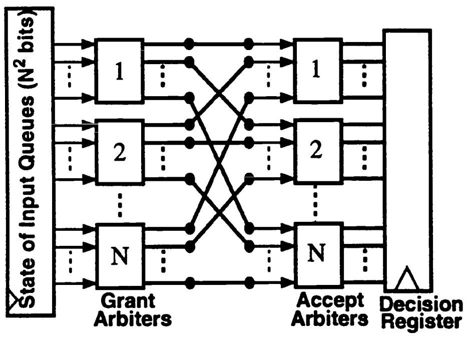


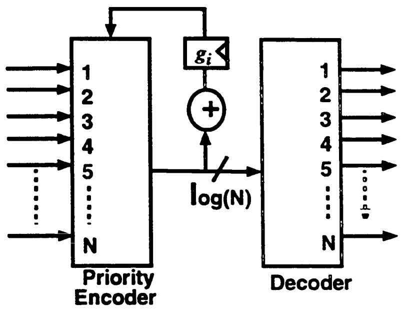
Figure 5 Performance of *i*SLIP for 1,2 and 4 iterations compared with FIFO and output queueing for independent arrivals with destinations uniformly distributed over all outputs. Results obtained using simulation for a 16x16 switch. The graph shows the average delay per cell, measured in time slots, between arriving at the input buffers and departing from the switch

ence output contention, which limits a throughput [5] to just $\left(1-\frac{1}{e}\right) \approx 63\%$. With a simple modification, *i*SLIP overcomes this problem by causing the arbiters to slip with respect to each other — a match in one cell time leads to a larger and faster match in the next cell time. This leads to a maximum throughput of 100% for uniform and independent arrivals *with just a single iteration.* A delay-throughput performance comparison of *i*SLIP with different numbers of iterations is shown in Figure 5.

The algorithm additionally provides fair access among contending flows and can reduce the burstiness of traffic as it transits the switch. Yet despite these features, a centralized scheduler for a 32×32 switch can be implemented in a single chip. Exploratory design-work suggests that *i*SLIP, implemented in current CMOS technology, can make a decision in less than 20ns.



a) Interconnection of 2N arbiters to implement *i*SLIP for an NxN switch.



b) Round-robin *grant* arbiter for the *i*SLIP algorithm. The priority encoder has a programmed highest-priority, g_i. The *accept* arbiter at the input is identical.

Figure 6 Implementation of *i*SLIP

Figure 4 illustrates the three-step arbitration of *i*SLIP for one iteration. A straightforward implementation of *i*SLIP is shown in Figure 6(a). The two building-blocks in this implementation are a round-robin arbiter and a register. A round-robin arbiter as shown in Figure 6(b) consists of a programmable-highest-priority priority encoder, a register holding a highest-priority pointer, and a binary-to-10-line decoder. The value of each priority pointer is updated to one location beyond its matched input or output only when the matching occurs in step three of the arbitration, otherwise the value remains unchanged.

3.2. Scheduling multicast traffic

A growing proportion of traffic on the Internet is multicast. If very high bandwidth, and hence input-queued, switches are to find widespread use in the Internet, it is important that they be able to handle multicast traffic efficiently. Furthermore, the crossbar switching fabric used in the switch can easily perform multicasting by connecting an input to multiple outputs. However, unlike the unicast case, we have found that it is infeasible to employ VOQ to eliminate HOL blocking — an 32 output switch would require $2^{32} - 33 > 4$ billions queues for all possible sets of outputs to which a multicast cell can go. Unless the number of queues approaches 2^{32}, we have found little benefit of maintaining multiple queues. Thus, for multicast traffic, the *Tiny Tera* maintains only a single FIFO queue at each input.

For multicast traffic, a scheduling algorithm can choose to either allow fanout-splitting or no fanout-splitting. When no fanout-splitting is allowed, all copies of a cell — some of which can be sent — must wait until all can be sent at the same time. Therefore, in order to achieve a high throughput, the *Tiny Tera* employs a fanout-splitting discipline because it is work-conserving. Figure 7 shows the performance improvement of fanout-splitting over no fanout-splitting.

In addition to allowing fanout-splitting, a multicast scheduling algorithm can employ residue-concentrating to achieve a higher throughput. To describe the concept of residue-

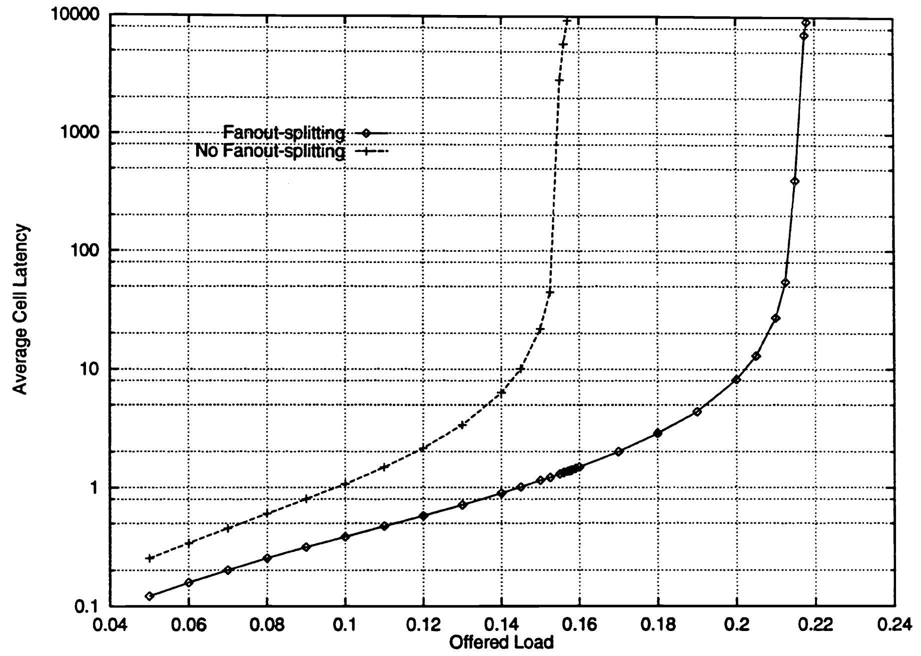


Figure 7 Average cell latency as a function of offered load, with uncorrelated input traffic with out destinations uniformly distributed over all outputs, and average fanout of four.

concentration, we define a term called an *output-cell.* A multicast cell at the input of the switch is a collection of output-cells each with an identical payload but different destination. A cell is said to be completely serviced, thus leaves its queue, when all of its output-cells are forwarded. Work-conservation requires that an input has to forward output-cells whenever the corresponding outputs are free, i.e., allow fanout-splitting.

Because of fanout-splitting, it is possible that some output-cells are left behind to be delivered in later slots. These output-cells are called the *residue*. For a given set of multicast cells, all work-conserving algorithms leave the same residue. It is up to a scheduling algorithm to decide on a residue placement, which uniquely defines the algorithm, i.e., deciding where to place the residue is equivalent to determining the transmission schedule.

A residue-concentrating algorithm is an algorithm which concentrates the residue on the smallest number of inputs. Concentrating the residue on the smallest number of inputs allows more HOL cells to be completely served, thus brings more new cells forward to the head of the line, i.e., brings more work forward. For the case of a 2xN switch, it is proved in [3], subject to a natural fairness constraint, that a residue-concentrating algorithm is optimal. We have found using simulation that a residue-concentrating algorithm will generally maximize throughput [3] for an $M \times N$ switch.

Unfortunately, the optimal residue-concentrating algorithm is too complex to be practicable. Moreover, a residue-concentrating algorithm can lead to starvation for cells that have a large number of destinations. The *Tiny Tera* will therefore employ one of two practical scheduling algorithms that we are developing. The first is called TATRA which uses the fact that the $M \times N$ scheduling problem can be mapped onto a Tetris-like game. The second is called WBA (Weight Based Algorithm) which is designed to use similar hardware to the *i*SLIP unicast scheduling algorithm. Both algorithms attempt to be fair, yet achieve a high throughput [3]. We are currently developing practical implementations of both algorithms.

4. CONCLUSION

Using only off-the-shelf CMOS technology, the *Tiny Tera*, an all-CMOS, input-queued, 32×32 port, ATM switch, will have an aggregate bandwidth of 320 Gbps. It will support a wide range of applications such as a high-performance ATM switch or the core of an IP router. In anticipation of increased multicast traffic, the *Tiny Tera* has dedicated support for both unicast and multicast traffic.

The *Tiny Tera* achieves a high aggregate bandwidth using an innovative architecture. It makes efficient use of memory bandwidth by employing input-queueing. It obtains high-bandwidth switching by having several crossbar bit-slices in parallel. With high-speed serial links, we have illustrated that the parallel bit-sliced switching core can be made no larger than a can of soda; yet is fast and simple. In order to make a cell forwarding decision at high speed, the *Tiny Tera* scheduling algorithms are simple, yet fair and efficient. Furthermore, they are implemented in hardware.

For unicast traffic, we eliminate HOL blocking entirely by employing Virtual Output Queueing (VOQ). We have demonstrated that, by using VOQ with a suitable scheduling algorithm, we can achieve close to 100% throughput. For multicast traffic, although not able to eliminate HOL blocking, we exploit the ideas of fanout splitting and residue-concentration to achieve a high throughput.

5. ACKNOWLEDGMENTS

We wish to thank Richard Edell of the University of California at Berkeley for discussions with Nick McKeown that lead to an earlier version of the *Tiny Tera* architecture, as well as Balaji Prabhakar and Ritesh Ahuja who helped in the development of multicast scheduling algorithms. We also thank other members of the *Tiny Tera* team: Mark Horowitz, William Ellersick, Pankaj Gupta, Ken Chang, Shang-Tse Chuang, Jeff Hsieh, Youngmi Joo, Brain Stark, and Rolf Muralt at Stanford; and Ah-Lyan Yee, Helen Chang, P.N.(Ani) Anirudhan, and Sharat Prasad at Texas Instruments.

6. REFERENCES

1. Anderson, T.; Owicki, S.; Saxe, J.; and Thacker, C. "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems.* Nov 1993 pp. 319-352.
2. Karol, M.; Hluchyj, M.; and Morgan, S. "Input versus output queueing on a space division switch," *IEEE Trans. Communications*, 35(12) (1987) pp.1347-1356.

3. McKeown, Nick; and Prabhakar, Balaji; "Scheduling Multicast Cells in an Input-Queued Switch," Proceedings of *IEEE Infocom '96*, March 1996, vol. 3, pp.271-278.

4. McKeown, Nick; Varaiya, Pravin; and Walrand, Jean; "Scheduling Cells in an Input-Queued Switch," *IEE Electronics Letters*, Dec 9th 1993, pp.2174-5.

5. McKeown, Nick; and Anderson, Tom E.; "A Quantitative Comparison of Scheduling Algorithms for Input-Queued Switches," submitted for publication. Available on request.

6. McKeown, Nick; "Scheduling Cells in Input-Queued Cell Switches," PhD. Thesis, University of California, Berkeley, 1995.

7. McKeown, Nick; Anantharam, Venkat; and Walrand, Jean; "Achieving 100% Throughput in an Input-Queued Switch," Proceedings of *IEEE Infocom '96*, March 1996, vol. 3, pp. 296-302.

8. McKeown, Nick; Martin Izzard; Adisak Mekkittikul; William Ellersick; Mark Horowitz; "The Tiny Tera: A Packet Switch Core," Proceedings of *Hot Interconnects IV*, Stanford, Aug 1996, pp. 161-173.

9. Tarjan, R.E. "Data structures and network algorithms," *Society for Industrial and Applied Mathematics,* Pennsylvania, Nov 1983.

10. "Utopia, An ATM-PHY Interface Specification," The ATM Forum Technical Committee, June 1995.