Design of a Weighted Fair Queueing Cell Scheduler for ATM Networks

Yuhua Chen
Department of Electrical Engineering
Washington University
St. Louis, MO 63130
Tel: (314) 935-5579
E-mail: yuhua@arl.wustl.edu
Fax: (314) 935-7302

Jonathan S. Turner
Department of Computer Science
Washington University
St. Louis, MO 63130
Tel: (314) 935-8552
E-mail: jst@cs.wustl.edu
Fax: (314)935-7302

Abstract

Today, ATM networks are being used to carry bursty data traffic with large and highly variable rates, and burst sizes ranging from kilobytes to megabytes. Obtaining good statistical multiplexing performance for this kind of traffic requires much larger buffers than are needed for more predictable applications or for bursty data applications with more limited burst transmission rates. Large buffers leads to large queueing delays, making it necessary for switches to implement more sophisticated queueing mechanisms in order to deliver acceptable quality of service. This paper describes a 2.4 Gb/s ATM queue management chip that has practically unlimited buffer scaling and which supports dynamic per VC queueing, an efficiently implementable form of weighted round-robin scheduling, a novel packet-level discarding algorithm and the ability to support multiple output links. We give a detailed description of our weighted round-robin scheduling method, which we call the *Binary Scheduling Wheels* (BSW) algorithm. The BSW algorithm is well-suited to hardware implementation. It allows cells to be scheduled in essentially constant time, and only requires a small increment in cost over a simple two level priority design.

1 Introduction

When ATM network technology was first developed in the 1980s, its developers envisioned a comprehensive traffic management methodology, with explicit reservation of resources, end-to-end pacing of user data streams to conform to resource reservations and network-level enforcement mechanisms to protect against inadvertent or intentional violation of resource reservations. In the context of such a methodology, efficient statistical multiplexing performance could be achieved without large amounts of buffering in the network and with very simple queueing mechanisms.

As ATM was deployed in the 1990s, the original expectations for traffic management were found to be unrealistic. ATM is now being used largely to support internet data traffic which is highly unpredictable and for which the traffic management philosophy of ATM is difficult to apply. In the current application context, resources are generally not explicitly reserved, end systems do not pace their transmissions and most network equipment cannot enforce resource usage limits. In this environment, to obtain good statistical multiplexing performance and high link utilization, one needs large buffers. In particular, one needs buffers that are at least comparable, and preferably an order of magnitude larger than user data bursts, which range in size from kilobytes to megabytes. Unfortunately, the use of large buffers with simple FIFO queueing disciplines leads to poor performance for real-time traffic and allows "greedy" applications to appropriate an unfair portion of network resources. Providing good quality of

service (QoS) to real-time applications and fair treatment to bursty data applications requires more sophisticated queueing and cell scheduling mechanisms [6][7][8][9][10].

This paper describes a design for an ATM queue manager that supports separate queues for each application data stream and buffer sizes that are limited only by the cost of memory. The design can be implemented with a single application-specific integrated circuit in 0.35 micron CMOS technology together with SRAM components. The design will support a total output rate of 2.4 Gb/s and can support either a single OC-48 link, or a combination of lower speed links.

Section 2 provides an overview of the ATM queue management chip, detailing its principle features and its overall architecture and operation. Section 3 contains a detailed description of our novel implementation of weighted round-robin scheduling called the *Binary Scheduling Wheels* (BSW) algorithm. The BSW algorithm is well-suited to hardware implementation and requires only a small increment in hardware complexity over a simple two priority design. It schedules and forward cells in essentially constant time, and can accommodate a large range of weights.

2 Overview of the Dynamic Queue Manager

The *Dynamic Queue Manager* (DQM) is designed to connect to the output side of a high performance ATM switch, such as the Washington University Gigabit Switch, described in [1]. The major features of the DQM chip are listed below:

Dynamic Queue Assignment -- The DQM implements per VC queueing using dynamic assignment, which allows the chip to support virtual path and virtual circuit connections with arbitrary choices of VPIs and VCIs and no explicit configuration of VCI ranges to particular VPIs. This greatly simplifies the use of the chip and enables optimal use of the chip's per channel data structures. Details can be found in [11].

Unlimited Buffer Scaling -- The DQM chip is designed so that the cell buffer can be scaled up to very large sizes without increasing the chip complexity significantly. Both the cell buffer and all information to maintain the cell buffer (that is, all the links for the linked list queues and the free slot list) are stored in external memory. The only constraint that the DQM chip places on the buffer capacity is through the choice of pointers. With 20 bit pointers, the chip can support buffer sizes over 50 Mbytes, 24 bit pointers would allow for up to 800 Mbytes. For all practical purposes, the buffer capacity is not constrained by the DQM chip.

Efficient Implementation of Weighted Round-Robin Scheduling -- The DQM chip implements weighted round robin scheduling [2] using a novel approach we call the *Binary Scheduling Wheels* (BSW) algorithm. The BSW algorithm is allows cells to be scheduled and forwarded in essentially constant time. Power of 2 weights can be assigned to individual virtual circuit connections. These weights determine the relative frequency with which cells are forwarded, allowing link bandwidth to be allocated appropriately during congestion periods. With 32 distinct weights, the BSW algorithm can assign bandwidth in amounts ranging from 2.4 Gb/s to less than one bit per second. Unlike naive implementations of weighted round-robin scheduling, the BSW algorithm interleaves cells from different channels as much as possible, minimizing the burstiness of the output data streams. The algorithm can be implemented in hardware in a very cost-effective way, requiring just a small increment in cost over a simple two level priority design.

Packet Level Discarding for per VC queues -- To preserve packet integrity during overload, ATM switches often use packet level discard mechanisms such as Early Packet

Discard [3][4], which were designed for use with FIFO queues. New algorithms are needed for per VC queueing, to minimize memory usage and preserve fairness and QoS properties of output scheduling algorithms. The DQM chip incorporates a new packet level discarding scheme for per VC queues, called the *Weighted Fair Goodput algorithm* (WFG). The combination of the WFG and the BSW allows all virtual circuits to forward cells at reserved rates during overload periods and ensures that "well-behaved" virtual circuits (those that do not exceed their allocated rate) do not lose any data, and that data is discarded from "misbehaving" virtual circuits on a packet-by-packet basis, avoiding wasted link capacity during overload periods.

Zero Overhead Memory Management -- At gigabit speeds, the bandwidth of the external memory used by the DQM to store cells is a precious resource. A certain portion this bandwidth must be used to manage the free space list that is stored in the external memory, along with the waiting cells. The DQM chip incorporates an on-chip cache that allows the free space list to be maintained using only memory cycles that would otherwise go unused. This cache stores the location of a number of available cell storage slots. Storage slots can usually be assigned to arriving cells from the cache and departing cells can usually return their cell slots to the cache, rather than accessing the off-chip free space list. The off-chip list is only accessed to refresh or free up space in the cache, but these operations can be performed during periods when there are guaranteed to be unused memory cycles available.

Figure 1 is a block diagram of the DQM chip and its associated memory. ATM cells are received on a 32 bit wide interface, similar to the Utopia interface, used for connecting ATM devices to SONET transmission circuits [5]. The DQM stores cells in the external memory and forwards them to one of possibly several output links. To support the required output bandwidth of 2.4 Gb/s, the chip operates with an internal clock speed of 120 MHz. This allows cells to be received at a rate that is roughly 1.5 times the cell rate of an OC-48 link.

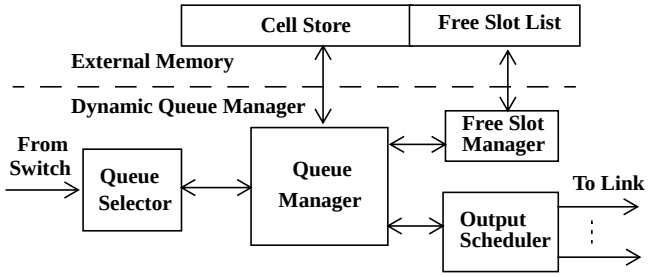

Figure 1 Simplified Block Diagram of Dynamic Queue Manager

There are six major functional blocks: the *Queue Selector*, the *Queue Manager*, the *Output Scheduler*, the *Free Slot Manager*, the *Cell Store*, and the *Free Slot List*. The *Queue Selector* dynamically assigns queues to virtual circuits. The *Queue Manager* maintains a list of all queues, keeping track of the first and the last cell in each queue. The *Output Scheduler* schedules the transmission of cells from the various channels and allocates the chip's output bandwidth among the different output links. The *Cell Store* buffers all incoming cells before transmission. The *Free Slot List* stores unused cell slots in the *Cell Store* and the *Free Slot Manager* manages the *Free Slot List.*

3 Binary Scheduling Wheels Algorithm

Two priority levels can be used to distinguish real-time traffic from non-real-time traffic, and minimize delay for real-time traffic. However, we may require greater flexibility in allocating bandwidth among virtual circuits. In this case, weighted round-robin scheduling can be used to allocate bandwidth among virtual circuits. The *Binary Scheduling Wheels* (BSW) algorithm used in the DQM implements weighted round-robin scheduling at minimal cost, providing a wide range of rate options. In addition, because bursty virtual circuits with high peak-to-average ratio are more likely to cause congestion in the downstream switches, the BSW algorithm distributes cells from the same channel evenly, minimizing the burstiness of the output streams.

3.1 Binary Scheduling Wheels

The Output Scheduler uses the per VC based Binary Scheduling Wheels algorithm to implement weighted round-robin scheduling in a very cost efficient way. All virtual circuits have power of 2 weights and during overload periods, share the link bandwidth in proportion to their weights. Instead of forwarding as many cells as specified by the weight once a queue is selected, the BSW algorithm places queues on scheduling wheels with different weights and alternates among wheels.

For implementation efficiency, we restrict weights to be powers of 2. Suppose we support W different weights: $2^0, 2^1, 2^2, \ldots, 2^{W-1}$. We construct W binary scheduling wheels, one for each weight factor. Each VC queue is placed on a corresponding scheduling wheel. The scheduling wheel with weight 2^0 is visited twice as frequently as the scheduling wheel with weight 2^1, four times as frequently as the one with weight 2^2, and so forth. W power of 2 weights can be coded using $\log W$ bits.

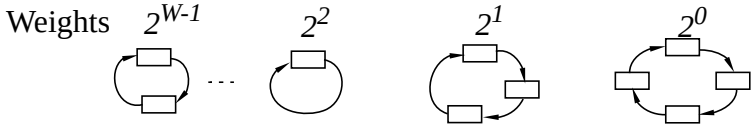

Figure 2 Binary Scheduling Wheels

Figure 2 shows an example with W binary scheduling wheels. Each little box in the figure represents a list node containing a queue identifier that identifies a non-empty per virtual circuit queue. Once a scheduling wheel is selected, all queues on that scheduling wheel can forward one cell to the output.

A W-bit binary counter can be used to select binary scheduling wheels with W weights. In a W-bit binary counter, the least significant bit of a binary counter changes twice as fast as the next lowest order bit, four times as fast as the next bit, and so forth. This property matches nicely with our scheduling wheel selection procedure.

Suppose the counter value is $C = c_{W-1} \ldots c_1 c_0$. When the counter advances, a change in bit i triggers servicing of the scheduling wheel with weight 2^i. We organize the transmission schedule into a series of passes. At the start of each pass, the counter is incremented. During the

pass, all scheduling wheels corresponding to changing counter bits are serviced.

3.2 Fast Forward Mechanism

The binary counter used in the above implementation is increased by one at the start of a scheduling pass. However, if all scheduling wheels that are enabled in a given pass are empty, we must increment the counter again to find a queue from which to send. In the worst-case it may take many increment steps to find a non-empty queue and during these steps, link bandwidth may be lost. To avoid this, we introduce a *fast forward* mechanism for the counter.

The idea is to increment the counter with a carry-in at the position of the right-most non-empty scheduling wheel. We keep a mask register to indicate non-empty wheels that have not been served. We also keep a carry-in register with only one bit set at the position corresponding to the least significant '1' bit of the mask register. After each pass, the value the carry-in register is added to the counter. The resulting right-most changing bit always corresponds to a non-empty scheduling wheel. The fast forward algorithm is shown below.

Initially,
PreviousCounter = 0;
CurrentCounter = 0;
Mask: Bit *i* is set to '1' if scheduling wheel *i* is non-empty, '0' if scheduling wheel *i* is empty;

Loop:
CarryIn = Position of the least significant '1' bit of (*Mask*);
PreviousCounter = *CurrentCounter*;
CurrentCounter = *CurrentCounter* + *CarryIn*;
ChangingBits = *PreviousCounter* XOR *CurrentCounter*;
CurrentMask = *Mask*;
While ((*CurrentMask* & *ChangingBits*) != 0)
CurrentWheel = Position of the least significant '1' bit of (*CurrentMask* & ChangingBits);
Serve all queues in the scheduling wheel CurrentWheel;
CurrentMask[CurrentWheel] = 0;
If (scheduling wheel CurrentWheel becomes empty)
Mask[CurrentWheel] = 0;
If (New queue is added to an empty scheduling wheel *j*)
Mask[*j*] = 1;

The following example shows how the algorithm works. Table 1 gives the parameter values of the fast forward counter at the beginning of a pass. Table 2 shows the selection process.

Table 1: Parameters of the Fast Forward Counter

Current Counter	*Previous Counter*	*Changing Bit*	*Mask*	*CarryIn*
0100	0011	0111	1011	0001

Table 2: Binary Scheduling Wheel Selection Process

Current Counter	*Current Mask*	*CurrentMask & ChangingBit*	*CurrentWheel*
0100	1011	0011	0001
0100	1010	0010	0010
0100	1000	0000	-

The value of the mask register *Mask* is $(1011)_2$, which means only Wheel 2 is empty. So the carry-in register *CarryIn* has the value $(0001)_2$. The previous value of the counter is $(0011)_2$. The current value of the counter is equal to the sum of the previous value and the carry-in value, which is $(0100)_2$. The '1' bits in the register *ChangingBit* indicate the changing bits of the counter. *CurrentMask* is set to the value of *Mask* initially. (*CurrentMask & ChangingBit*) gives all scheduling wheels to be served in a pass. Since only one wheel can be visited at a time, *CurrentWheel* specifies the current scheduling wheel to be served. When a scheduling wheels is visited, all the queues on that wheel can send one cell to the outgoing link. After a scheduling wheel is served, the corresponding bit in *CurrentMask* is then cleared. This process continues until all selected scheduling wheels have been served once.

With the fast forward mechanism, the selection time becomes essentially independent of the total number of weights. While the time to select the least significant '1' bit does require more than constant time, hardware implementation can easily be made fast enough that this does not becomes an issue for realistic values of *W*. Consequently, cells can be selected and forwarded in essentially constant time.

3.3 Design Analysis

We need to understand how the number of weights and the number of virtual circuits in the system affect the hardware complexity. Figure 3 shows the structures in the *Output Scheduler* that implement the Binary Scheduling Wheels algorithm. Let *N* be the number of queues supported by the DQM. Let *W* be the number of weight levels. The DQM implements the BSW at multiple outputs. Let *m* be the number of outputs.

At each output, we construct *W* scheduling wheels. We need a queue list with *N* entries to represent the queues in scheduling wheels. Since a virtual circuit has only one destination, a queue can only be placed on one of the scheduling wheels. Therefore, the queue list can be shared by all output wheels. The actual scheduling wheels are constructed by linking corresponding entries in the queue list. An additional bit is added to entries in the queue list to distinguish the first queue in a wheel. To access the scheduling wheels, a wheel table is used to store the pointers to the scheduling wheels. Both the queue list and the wheel table are implemented as on-chip SRAM. The memory requirement for the BSW for *m* outputs is $(N + mW)\ (1 + \lceil \log N \rceil)$.

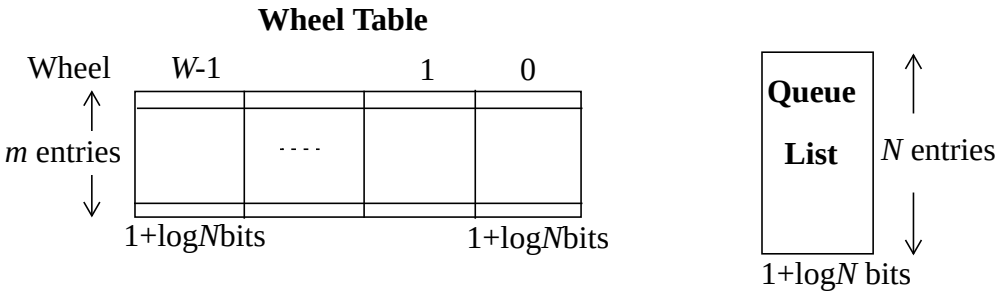

Figure 3Structures in the Output Scheduler

Figure 4 shows the memory size for various parameters with $m = 16$. Note that only the wheel table depends on the number of weights. The increment in memory size to implement 64 weights over a simple two priority design is less than 2 KBytes. 32 distinct power of 2 weights are sufficient to specify bandwidths ranging from 2.4 Gb/s to less than one bit per second. Even with 8192 virtual circuits in the system, the total memory requirement for the scheduler is less than 15 KBytes. Therefore, the BSW algorithm implements the weighted round-robin scheduling in a very efficient way.

Because a VC queue only belongs to one scheduling wheel, adding and removing a queue from a scheduling wheel can be done in constant time. This is the major reason for restricting weights to be powers of 2. The algorithm can be extended to more general weights by allowing each queue to appear in multiple wheels. If a queue can appear in j wheels, the ratio between successive weights is $\frac{2^j}{2^j - 1}$, but both the scheduling time and the size of the queue list increase by a factor of j.

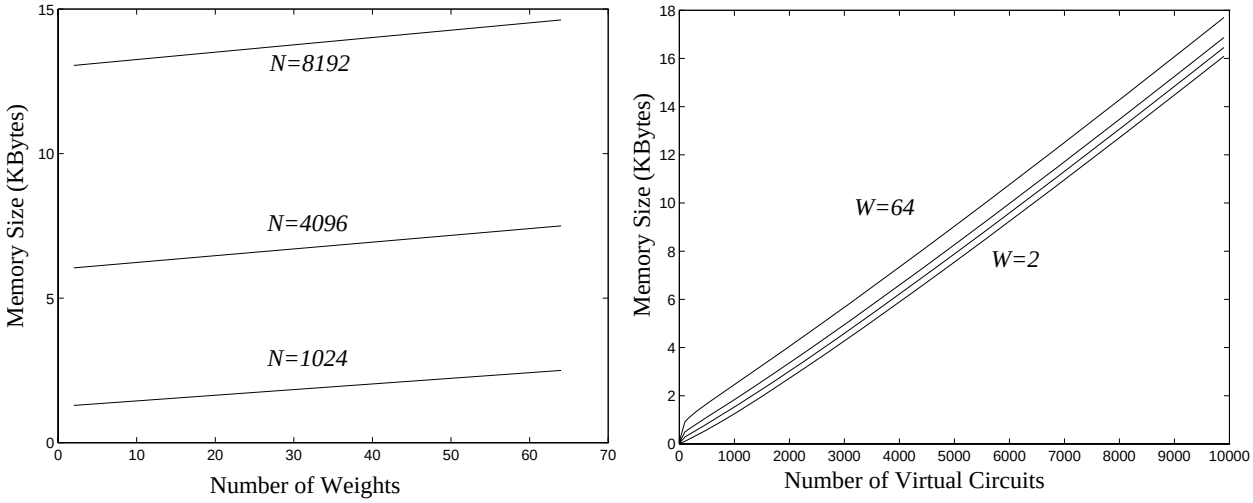

Figure 4 Memory Required for Cell Scheduling

4 Summary

In this paper, we have described a dynamic queue manager for gigabit ATM networks and presented a detailed description of the Binary Scheduling Wheels (BSW) algorithm used in the design. The BSW algorithm implements weighted round-robin scheduling in a cost efficient way. VC queues are placed on different scheduling wheels based on power of 2 weights. A fast forward mechanism allows cells to be scheduled in essentially constant time. The BSW algorithm only requires a small increment in hardware cost over a two priority design, and is suitable for hardware implementation.

References

[1] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, J. S. Turner, “Design of a Gigabit ATM Switch”, INFOCOM 1997.

[2] Manolis Katevenis, Stefanos Sidiropoulos, and Costas Courcoubetis, “Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip,” *IEEE Journal on Selected Areas in Communication*, vol. 9, No. 8, Oct. 1991, pp. 1265-1279.

[3] Allyn Romanow, Sally Floyd, “Dynamics of TCP Traffic over ATM Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 4, May 1995, pp. 633-641.

[4] J. S. Turner, “Maintaining High Throughput During Overload in ATM Switches,” I*NFOCOM* ’96, pp. 287-295.

[5] The ATM Forum Technical Committee, UTOPIA Level 2, v1.0, af-phy-0039.000.

[6] Haoran Duan, J. W. Lockwood, et al., “A High-Performance OC-12/OC-48 Queue Design Prototype for Input -Buffered ATM Switches,” *INFOCOM* ’97.

[7] J. C. R. Bennett, D. C. Stephens and H. Zhang, "High Speed, Scalable, and Accurate Implementation of Packet Fair Queueing Algorithms in ATM Networks", *Proceedings of ICNP* ’97, pp7-14.

[8] Dallas E. Wrege, Jorg Liebeherr, “A Near-Optimal Packet Scheduler for QoS Networks,” *INFOCOM* ’97.

[9] Yoshihiro Ohba, “QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM networks,” *INFOCOM* ’97.

[10] H. Jonathan Chao, “An ATM Queue Manager Handling Multiple Delay and Loss Priorities,” *IEEE/ACM Tran. on Networking*, vol. 3, no. 6, Dec. 1995, pp 652-659.

[11] Yuhua Chen, Jonathan S. Turner, “Dynamic Queue Assignment in a VC Queue Manager for Gigabit ATM Networks,” submitted to *IEEE ATM ‘98 Workshop*.