Dynamic Queue Assignment in a VC Queue Manager for Gigabit ATM Networks

Yuhua Chen
yuhua@arl.wustl.edu
Department of Electrical Engineering
Washington University
St. Louis, MO 63130

Jonathan S. Turner
jst@cs.wustl.edu
Department of Computer Science
Washington University
St. Louis, MO 63130

Abstract

Today, ATM networks are being used to carry bursty data traffic with large and highly variable transmission 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 lead to large queueing delays, making it necessary for switches to implement more sophisticated queueing mechanisms in order to deliver acceptable quality of service (QoS). This paper describes a 2.4 Gb/s ATM queue management chip that has practically unlimited buffer scaling and also 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 the dynamic queue assignment mechanism, which allows the chip to support both virtual path and virtual circuit channels. This mechanism is self-configuring, eliminating the need to configure the range of VCIs associated with a given VPI, and makes optimal use of the chip's per-channel data structures, since these data structures can be assigned only when cells are present in the queues.

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][13][14][15].

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 dynamic queue management chip, detailing its principle features. Section 3 describes the architecture and operation of the *Dynamic Queue Manager* in more detail. Section 4 contains a detailed description of the dynamic queue assignment mechanism, which allows the chip to support both VP and VC connections without explicit configuration of VCI ranges for individual VPIs. The dynamic queue management mechanism also makes optimal use of the chip's per channel data structures, since it can assign these data structures to channels only when data is present. This makes it possible for the chip to support more connections than it has per channel data structures.

2 Overview of 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.

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 [17]. The BSW algorithm is well-suited to hardware implementation and allows cells to be scheduled and forwarded in essentially constant time. Binary 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 a very cost-effective way, requiring just a small increment in cost over a simple two priority level 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* (WFG) algorithm. The combination of WFG and 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.

Efficient Maintenance of Free Space List in External Memory -- At gigabit speeds, the bandwidth of the external memory used by the DQM to store cells is a precious resource. A certain portion of 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.

3 Operation of Dynamic Queue Manager

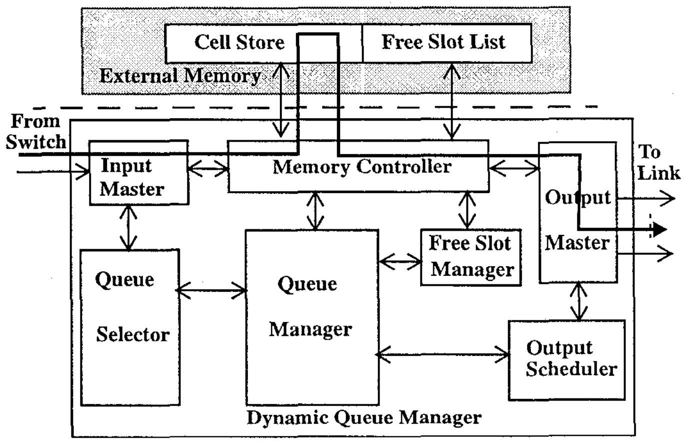

Figure 1 Block Diagram of the Dynamic Queue Manager

Figure 5 shows a block diagram of the DQM chip and its associated memory. The solid line illustrates the data flow from the switch fabric to the output links. 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.

There are nine functional blocks: the *Queue Selector*, the *Queue Manager*, the *Output Scheduler*, the *Free Slot Manager*, the *Cell Store*, the *Free Slot List*, the *Input Master*, the *Output Master*, and the *Memory Controller*. 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 various channels and allocates the chip's output bandwidth among 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* maintains an on-chip cache and manages the Free Slot List. The *Input Master* receives cells from the switch and retrieves control information. The *Memory Controller* interfaces to the external memory and handles the necessary format conversions, needed to map cells into memory. The *Output Master* forwards cells to the output links. The major blocks are described below.

3.1 Queue Selector (QSEL)

The Queue Selector (QSEL) maps the VPI and VCI fields of incoming cells to dynamically assigned queue identifiers. It contains a *Queue Lookup Table* (QLT), which operates like a *Content-Addressable Memory* (CAM). Logically, one can think of the QLT as a set of entries, each containing a (VPI,VCI) and a queue identifier. When a cell arrives, the VPI and VCI fields of the cell are compared to the stored entries. If a matching entry is found, the queue identifier in the entry gives the number of the queue that the cell is to be appended to. If there is no matching entry, a free queue is allocated and a new entry is created, and initialized with the VPI and VCI of the incoming cell and the identifier of the queue just allocated.

Figure 2 illustrates the Queue Selector data structures. In addition to the QLT, the Queue Selector contains a *Free Queue List* and an *Address Map*. Queues are allocated from the Free Queue List as needed. When the transmission of a cell by the Queue Manager causes a queue to become empty, the identifier for that queue is returned to the Queue Selector, which adds it back into the Free Queue List. The Address Map specifies the QLT entry where a specified queue identifier is stored. It is used to remove an entry from the QLT when a queue becomes empty. Specifically, whenever a queue identifier is returned to the Free Queue List, the Queue Selector uses the queue identifier to select an entry from the Address Map. The value returned is then used to deallocate the specified QLT entry. To make the implementation more cost-effective, the QLT is implemented not with a CAM, but with a *Set-Associative Memory* (SAM). The implementation and its performance are described in detail, in Section 4.

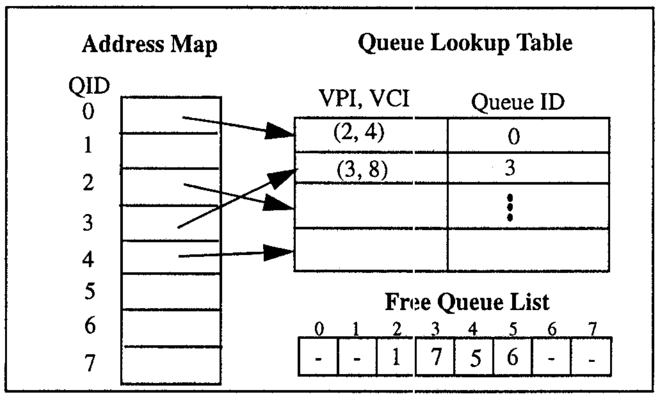

Figure 2 Queue Selector Data Structures

3.2 Queue Manager (QMGR)

The Queue Manager (QMGR) maintains a queue list for every possible virtual path queue and virtual circuit queue by keeping pointers to the head and tail of the queues. The actual queues are organized as linked lists and are stored in the external memory (Cell Store). Each cell slot in the Cell Store contains one cell and a pointer to the next cell in the queue. In order to simplify the design,

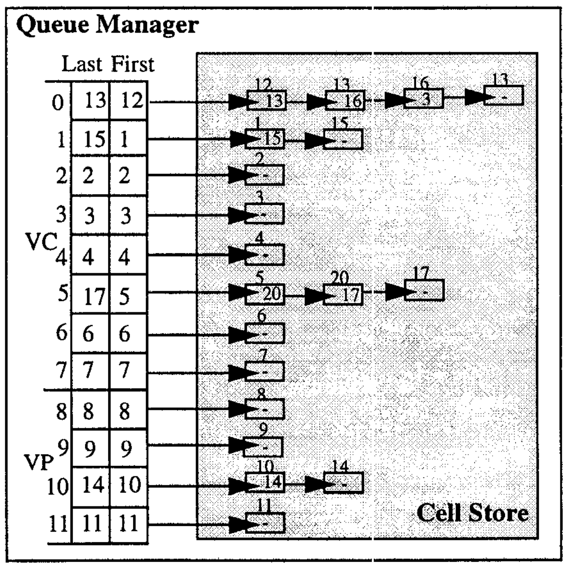

Figure 3 Queues Maintained by the Queue Manager

the last cell in the queue always points to an empty cell slot in the Cell Store. Figure 3 shows the queues maintained by the QMGR for a scaled-down configuration.

The queue identifiers generated by the Queue Selector are used to index the queue list. For an incoming cell, the QMGR informs the Cell Store to store the cell in the cell slot which the last cell in the queue points to. The QMGR also obtains a free slot from the Free Slot Manager and updates its queue list. The free slot number is written into the Cell Store along with the cell. For an outgoing cell, the QMGR sends the cell slot number of the first cell in the queue to the Cell Store. The pointer stored with the outgoing cell is used to update the queue list.

3.3 Output Scheduler (QSCHL)

The Output Scheduler (OSCHL) determines the order in which different non-empty queues are selected for transmission. Because the chip is designed to support multiple outputs (up to 16 OC-3 interfaces, in some configurations), the Output Scheduler has separate scheduling data structures for each output. Each queue is assigned a weight from a fixed set of binary weights. For each output, the OSCHL maintains an array of scheduling lists, with one list for each weight. This is illustrated in Figure 4, which shows a scaled-down version of the OSCHL data structures, configured for four outputs and four weights. The actual chip supports up to 16 outputs and 32 weights.

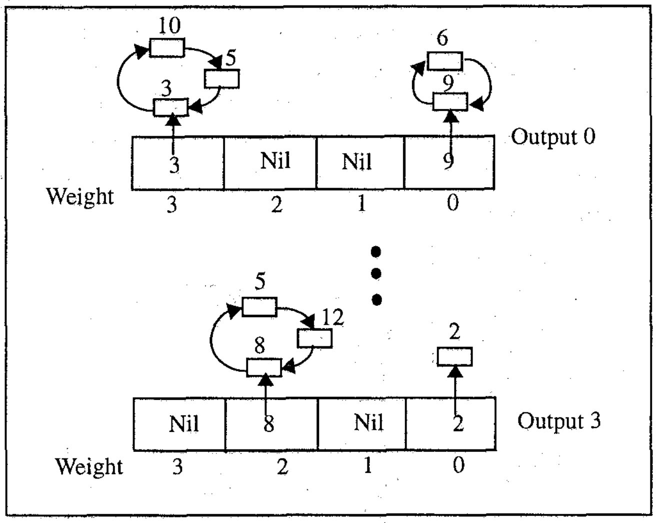

Figure 4 Scheduling Wheels in the Output Scheduler

The scheduling lists are organized cyclically and are referred to as scheduling wheels. The weights for the different scheduling wheels determine the relative frequency with which the queues are scheduled. Queues on the weight 0 wheel are visited twice as often as queues on the weight 1 wheel, four times as often as queues on the weight 2 wheel, and so forth. The scheduling algorithm used by the OSCHL is called the Binary Scheduling Wheels algorithm and is described in detail in [17].

4 Dynamic Queue Assignment

The ATM cell format allows for as many as 2^{28} distinct virtual circuit connections on a single ATM link. Real switches implement only a small fraction of the full spectrum of possibilities, and often impose limitations on the choices of VPIs and VCIs. Many switches support only VP or only VC connections and those that only support VC connections usually restrict the VPI to be zero. In switches that support VC connections with different VPIs, it is generally necessary to configure the switches to specify which VCIs may be used with a given VPI. The DQM avoids this by using set associative lookup to assign queues to channels identified by the combination of a VPI and a VCI.

4.1 Set-Associative Lookup

To obtain the most cost-effective implementation, the Queue Lookup Table (QLT) in the Queue Selector is implemented using a Set-Associative Memory or SAM. SAM's can be implemented with conventional SRAM and some auxiliary logic, making them a good deal cheaper than CAMs. Given a VPI and a VCI, the set-associative memory in the Queue Selector returns a set of entries, any one of which could be used for storing information relating to that VPI and VCI combination. A set of entries is selected using a subset of the bits of the VPI and VCI. Each entry contains a valid bit, a tag and a queue identifier. The tag is formed from the bits of the VPI and VCI not used to select the set.

When a cell is received by the DQM, its VPI and VCI are passed to the Queue Selector, which retrieves a set of entries from the set-associative memory. The tag of the incoming cell is compared to the tag fields of all entries of the set in parallel. If the tag field of some valid entry in the set matches the tag of the incoming cell, the queue identifier stored in that entry identifies the queue that the cell should be appended to. If there is no tag match, it means that no cell belonging to this combination of VPI/VCI is currently stored in the DQM's memory, and so an unused queue identifier should be assigned. To perform this assignment, the Queue Selector picks an unused entry from the set returned by the set-associative memory and obtains an unused queue identifier from the list of free queue identifiers that it maintains. It then copies the tag of the cell and the number of the allocated queue into the selected entry, sets its valid bit and writes it back to the set associative memory.

When the DQM transmits the last cell from some queue, the number of that queue is passed to the Queue Selector which returns the queue to its list of available queues and clears the valid bit in the corresponding entry of the set-associative memory. Thus, both queues and entries in the set-associative memory are used only for those connections for which the DQM is storing cells.

4.2 Overflow Handling

When a cell is received by the DQM, it is possible that the set-associative lookup will yield a set of entries, all of which are in use (have the valid bit set) and none of which have matching tags. In this case, the cell must be discarded. This is referred to as overflow. We can make overflow less frequent by augmenting the set associative memory with a small Content Addressable Memory (CAM). The entire VPI and VCI are used as the key field for the CAM. In the value field, we store the queue identifier that is assigned to the virtual circuit. Figure 5 shows the set-associative lookup with CAM.

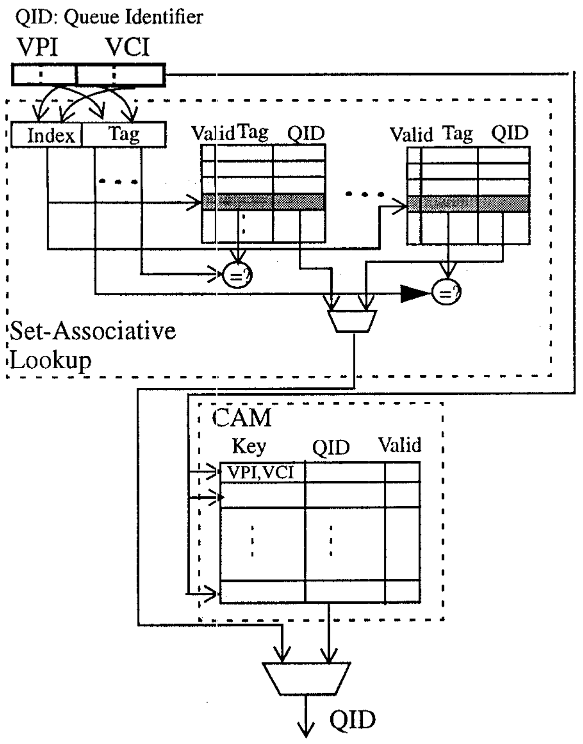

Figure 5 Set-Associative Lookup with CAM

When a cell enters the DQM, a CAM lookup is performed in parallel with the set-associative lookup described earlier. There are several cases that can then arise: If the set returned by the set-associative memory has a matching entry, it is used as previously described. If the set has no matching entry, but the CAM contains a matching entry, then the queue identified in the matching CAM entry is used for the arriving cell. In this case, if the set returned from the set-associative memory has an unused entry, the information in the CAM entry is transferred from the CAM entry to the entry in the set-associative memory, freeing up the CAM entry. If neither the set returned by the set-associative memory, nor the CAM has a matching entry, then a new queue is allocated. If the set has one or more free entries, one of them is allocated. Otherwise an entry in the CAM is allocated.

The inclusion of the CAM makes overflow less likely, since it can occur only if there is no room in the selected set nor in the CAM. In the next subsection, we study the overflow process and determine how to configure the SAM and CAM to make the probability of overflow acceptably small.

4.3 Design Analysis

4.3.1 Overflow Probability

To complete the design of the queue selector, we need to understand how the configuration of the SAM and CAM affects the probability of cell loss due to overflows. Let n be the number of queues supported by the DQM. Let β be the ratio of n to the number of storage locations in the SAM. This quantity bounds the fraction of the SAM entries that can be in use at one time, and is called the *load factor*. Let s be the number of entries in each set of the SAM, let $r = \frac{n}{\beta s}$ be the number of sets in the SAM and let c be the number of entries in the overflow CAM.

We define the overflow probability to be the probability that when a cell A arrives on a "new connection" (one for which no queue is currently allocated), there is no available entry in either the set-associative memory or the CAM.

To calculate the probability of overflow, we must make some assumption about the number and distribution of "in-use" entries in the set-associative memory and the CAM, at the time cell A arrives. We will assume that the "in-use" entries are randomly distributed in the following way. Let the set-associative memory and the CAM be empty initially. Now suppose that n cells arrive on n different virtual circuits. Assume that each of the arriving cells is equally likely to be mapped to any of the sets in the set-associative memory and that all arrivals are independent. Now, define x_i to be the number of virtual circuits (out of the original n) that are mapped to set i in the set associative memory but spill over into the CAM.

$$Pr\{x_i = 0\} = \sum_{i=0}^{s} \binom{n}{i} \cdot \left(\frac{1}{r}\right)^{i} \cdot \left(1 - \frac{1}{r}\right)^{(n-i)}$$

and for $1 \le h \le n - s$,

$$Pr\{x_i = h\} = \binom{n}{s+h} \cdot \left(\frac{1}{r}\right)^{s+h} \cdot \left(1 - \frac{1}{r}\right)^{n-(s+h)}$$

Let y be the total number of virtual circuits that spill over from their sets in the set-associative memory to the CAM. Clearly $y = x_1 + \ldots + x_r$. We can get a conservative upper bound on y by treating the x_i as independent random variables. In particular, if we let z be the random variable whose distribution is obtained by taking the convolution of the distributions for $x_1, \ldots, x_r$, then $Pr\{z \geq c\} \geq Pr\{y \geq c\}$.

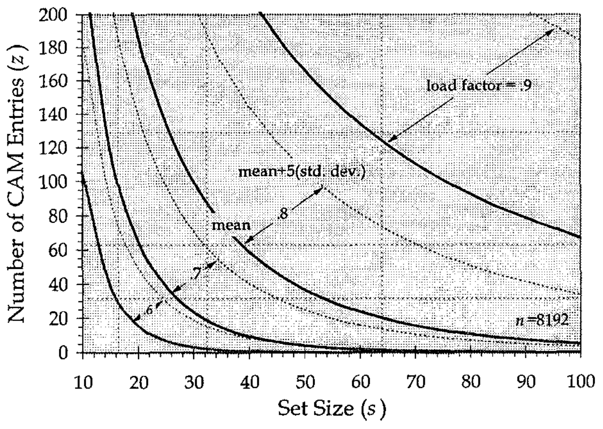

Figure 6 Recommended CAM Size

Figure 6 shows how z varies with s and the load factor β. The plot includes curves showing the mean value of z and the mean plus five standard deviations. These were calculated numerically using the probability distribution of x_i to obtain its mean and standard deviation, then multiplying these by r and $\sqrt{r}$ to obtain the mean and standard deviation of z. These curves allow us to determine the values of c, s and β that will lead to good performance. For example, we can see that if we want to operate with a load factor of 0.8, then with a set size of s, we need a CAM size of at least 25 and more realistically, about 75 to keep overflows acceptably rare. If we want to operate with a higher load factor, we must increase s, c or both. With a lower load factor, we can reduce s and c, at the cost of more memory.

The overflow probability is no more than

$$\begin{aligned} Pr\{x_i > 0, z > c\} &= Pr\{x_i > 0\} Pr\{z > c | x_i > 0\} \\ &= Pr\{z > c\} Pr\{x_i > 0 | z > c\} \\ &\leq min\left(Pr\{x_i > 0\}, Pr\{z > c\} \right) \end{aligned}$$

By the central limit theorem, we can estimate z using a normal distribution. Let μ and σ denote the mean and standard deviation of x_i. For any positive number γ,

$$\begin{aligned} Pr\{z > r\mu + \gamma \cdot \sigma\sqrt{r}\} &\approx \frac{1}{\sqrt{2\pi}} \int_{\gamma}^{\infty} e^{-\gamma^2/2} dx \\ &< \frac{1}{\gamma\sqrt{2\pi}} \cdot e^{-\gamma^2/2} \end{aligned}$$

Figure 7 plots the value of this last quantity, as a function of load factor, for several different choices of s and c. With a set size and CAM size of 32, the estimated overflow probability is less than one in a million when the number of storage locations in the SAM is $(1/0.65)n$. If both are increased to 64, a load factor of nearly 0.8 yields the same overflow probability. For $n = 8192$, a load factor of 0.8 implies 10,240 SAM entries. With $s = 64$, this implies $r = 160$.

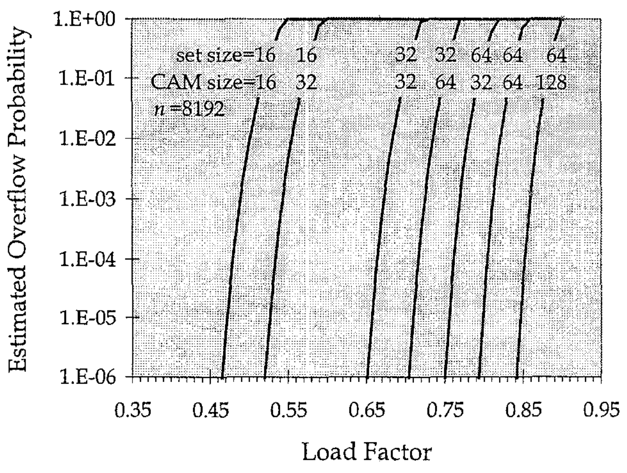

Figure 7 Overflow Probability

4.3.2 Memory Estimation

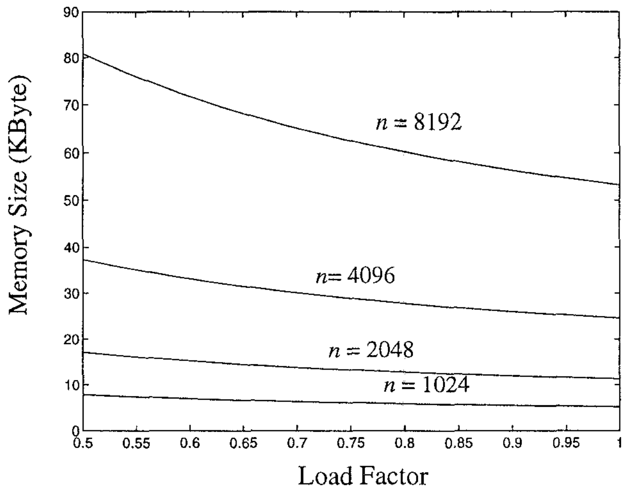

Figure 8 Memory Size for Dynamic Queue Assignment

The chip complexity is mainly driven by the on-chip memory. The memory required to implement dynamic queue assignment is shown in Figure 8. The calculation includes the SAM, the Free Queue List and the Address Map. For $n = 8192$, a load factor of 0.8 gives a memory requirement of about 60 Kbytes, while a load factor of 0.5 gives a memory requirement of 80 Kbytes. Note that the load factor affects only the SAM, but not the Free Queue List or the Address Map. When the load factor is 0.5, the memory required for the SAM is 1/3 of the total.

Figure 7 shows the memory area estimation for a 0.35 micron CMOS process. Combining the results in Figure 8 and Figure 7, with 1024 queues and a load factor of 0.8, the chip area consumed by dynamic queue assignment is less than 2 mm^2. For 8192 queues, the area is approximately 17 mm^2, or less than 20% of the area of a 100 mm^2 chip. The analysis indicates that the number of queues could be increased to 16K without consuming an excessive fraction of the chip area. 64K queues can be supported in a 0.18 micron process.

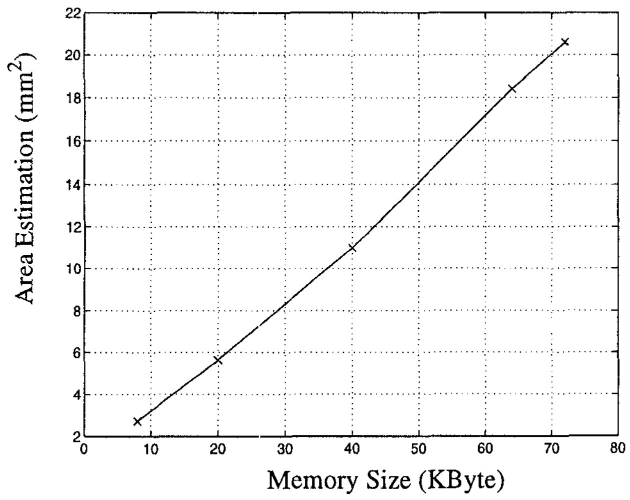

Figure 9 Memory Area Estimation (0.35 Micron)

4.4 Overbooking of Queue Data Structures

Because the DQM assigns queues to virtual circuits dynamically, it is possible to support a larger number of virtual circuits than could be supported if queues were statically bound to specific virtual circuit identifiers. That is, we can overbook the DQM's data structures, taking a risk that on occasion we will not have an available queue to handle an arriving cell, forcing the cell to be discarded. In order to exploit the potential for overbooking, it is important to understand how many virtual circuits can be supported with a given number of queues. Here, we make some basic observations, leaving a detailed analysis of overbooking to a future study.

Note first that if a DQM supports n queues, there will always be an available queue if the number of queued cells is $\leq n$. For non-bursty traffic, the queue length rarely exceeds even 100 cells for traffic loads of 95% or less. Thus, for $n = 8192$, the probability is exceedingly small that an arriving cell will not find an available queue, even if the number of virtual circuits using the link is over one million.

For bursty traffic, it is also possible to overbook the queues extensively. Suppose we have m identical independent on-off bursty sources with $m > n$ and an average time of T between the start of successive bursts (from any single source). If the input traffic (averaged over periods longer than T) is less than the link rate, then the average rate from each individual source is the link rate divided by m, which is small if n is reasonably large. Typical virtual circuits have peak rates of perhaps 20 times the average rate. For $n = 8192$, this results in virtual circuit peak rates that are less than 0.25% of the link rate. For such traffic, the queue rarely accumulates a significant backlog of cells, so again, an arriving cell will generally find an available queue.

Suppose however, that we have sources with peak rates that are much larger than their average rates. In particular, assume that bursts arrive independently and instantaneously, with an exponentially distributed time between bursts from any specific source. Also, assume that burst lengths are exponentially distributed and that each burst is assigned a separate queue (even two bursts coming from the same source), and that all non-empty queues are drained at a rate that is inversely proportional to the number of non-empty queues (modeling a round-robin queue scheduler). This queueing system can be modeled by a birth-and-death process, in which the state index corresponds to the number of non-empty queues. If we let the number of sources go to infinity, while keeping the time between successive burst arrivals constant, this birth-and-death process becomes identical to that for the M/M/1 queue. This implies (among other things) that the probability that there are more than j non-empty queues is ρ^{j+1}, where ρ is the normalized traffic intensity for the queueing system. For $\rho = 0.95$, this probability is less than 10^{-6} for all $j > 268$. These results show that the DQM queues can be overbooked by a large factor, if n is sufficiently large. With smaller n, the potential for overbooking is reduced somewhat, but even with as few as 1024, we are unlikely to run out of queues under any realistic traffic conditions.

5 Summary

In this paper, we have described a dynamic queue manager for gigabit ATM networks and presented a detailed design and analysis of the dynamic queue selection component. The queue selection uses a set-associative memory and overflow CAM to enable flexible assignment of virtual path and virtual circuit connections to queues. This eliminates the need to explicitly configure which VCIs can be used with which VPIs and allows the DQM chip to make optimal use of its per channel data structures. This in turn, makes it feasible to overbook of the DQM's queue data structures.

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," INFOCOM'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] Maurizio Casoni, J. S. Turner, "Improved Analysis of Early Packet Discard," Proceedings of the International Teletraffic Congress,

[12] R. Chipalkatti, J. F. Kurose, and D. Towsley, "Scheduling Policies for Real-Time and Non-real-Time Traffic in a Statistical Multiplexer," Proc. 1989 GLOBECOM conf., 1989, pp 774-783.

[13] Massoud R. Hashemi, Alberto Leon-Garcia, "A General Purpose Cell Sequencer/Scheduler for ATM Switches," INFOCOM'97.

[14] Paul Landsberg, Charles Zukowski, "Generic Queue Scheduling: Concepts and VLSI," INFOCOM'94, pp. 1438-1445.

[15] Yoshihiro Ohba, "QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM networks," INFOCOM'97.

[16] A. A. Lazar, A. Temple, and R. Gidron, "A Metropolitan Area Network Based on Asynchronous Time Sharing," IEEE ICC, June 1989, pp. 630-634.

[17] Yuhua Chen, J. S. Turner, "Design of a Weighted Fair Queueing Cell Scheduler for ATM Networks," submitted to GLOBECOM '98, 1998.