Multi-rate Traffic Shaping and End-to-End Performance Guarantees in ATM

Debanjan Saha*
Department of Computer Science
University of Maryland
College Park, MD 20742
E-mail: debanjan@cs.umd.edu

Sarit Mukherjee†
Dept. of Computer Science & Engg.
University of Nebraska
Lincoln, NE 68588-0115
E-mail: sarit@cse.unl.edu

Satish K. Tripathi*
Department of Computer Science
University of Maryland
College Park, MD 20742
E-mail: tripathi@cs.umd.edu

Abstract

This paper proposes a traffic control scheme for integrated services ATM networks. The control strategy comprises of two components: a shaping mechanism at the network entry point and a frame based service discipline at the switches. The shaper enforces a short trem peak rate, and a long term average rate. The multiplexing scheme at a switch allocates a guaranteed bandwidth to a connection. A connection may get more than the guaranteed amount, up to a connection specific maximum, if slack bandwidth is available. By imposing an upper bound on the allocated bandwidth, we secure a better handle on the delay jitter. Unlike most frame-based schemes, our scheme allows allocation of bandwidth at any arbitrary granularity. We suggest a simple admission control policy and derive deterministic bounds on end-to-end delay and jitter. An outline of a hardware realization of the scheme is also presented.

*Supported in part by ARPA and Philips Labs under contract DASG60-92-0055 and NSF under contract CCR-9002351.

†Supported in part by the Office of Research Council, UNL, under the grant LWT/05 -142-93801.

1 Introduction

An integrated services ATM network is expected to offer services to a wide range of real-time applications. The integration of real-time traffic in ATM networks places stringent requirements on the worst-case end-to-end delay and jitter[1] bounds associated with such traffic. At the same time, to accommodate diverse traffic from these applications, network services need to be very flexible. In this paper we develop a framework for traffic control that can provide deterministic guarantees on end-to-end performance, and can support diverse throughput requirements of a wide mix of applications.

The traffic control schemes [10, 3, 4, 7, 5, 9, 6, 2] proposed for integrated services packet switched networks comprise of two components: traffic shaping at the network entry point and a packet service discipline at the switches. Besides smoothening, the shaper helps characterize the traffic entering the network. A succinct characterization is extremely important in obtaining tight performance bounds. However, traffic shaping does not necessarily guarantee these bounds since the characteristics of the shaped traffic may get modified at the intermediate switches. In order to provide predictable end-to-end performance, we need to characterize the traffic entering every switch on the path of a connection. Hence, it is important to use a suitable service discipline at switches so that traffic from a connection exiting a switch and entering the next switch on its path can be easily and precisely characterized.

In most of the proposed schemes, the shaper enforces a specific rate of packet arrival, typically a declared peak or average rate. However, most applications generate inherently bursty traffic. Hence, enforcing an average rate results in higher delay, and on the other hand, a peak rate allocation leads to lower network utilization. To strike a balance between network delay/jitter and bandwidth utilization, it is desirable that the network enforces multiple rates — a peak rate to allow bursty traffic over short intervals, and an average rate (much lower than the peak rate) at all other time.

Packet service discipline at switching nodes can be broadly classified as either *work-conserving* or *non-work-conserving*. In a work-conserving service discipline an output link is never idle so long as there are packets to send. A non-work-conserving discipline can keep a link idle even if there are packets waiting. Although non-work-conserving policies offer higher average delay than work-conserving polices, typically the former enforces lower delay jitter (hence smaller buffer space). For guaranteed worst case performance, a non-work-conserving service discipline at switches is a preferred choice.

In our scheme, the shaper enforces a maximum allowed "average" rate over a long interval while giving the connections an option to violate this constraint by choosing a "peak" rate over shorter

[1]The maximum difference over all packets between departure and arrival times.

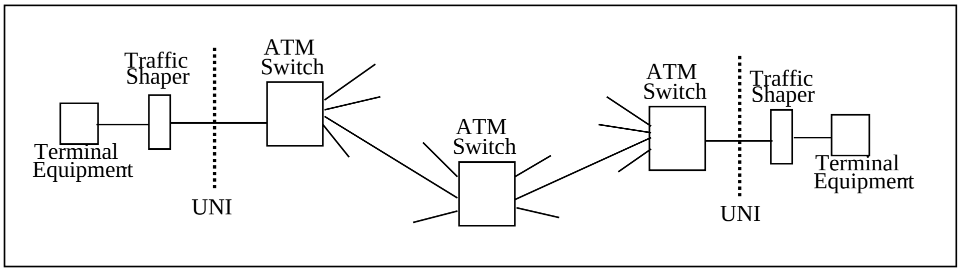

Figure 1: ATM network architecture.

intervals. We use a frame based non-work-conserving packet scheduling scheme at the switches. The scheme guarantees all connections a minimum bandwidth (atleast the average rate) negotiated at the connection set up time. A connection may use more than the allocated minimum, up to a connection specific maximum, if slack bandwidth is available. By imposing an upper bound on the allocated bandwidth we secure a better handle on delay jitter. Our scheme also overcomes a common shortcoming of most frame based service disciplines, i.e., allocation of bandwidth only in multiples of a fixed quantum. In a high-speed network supporting a wide mix of applications, this inflexibility can lead to severe under-utilization of resources. We solve this problem by multiplexing several connections on a single slot of a frame (to be discussed in detail in section 2).

The rest of the paper is organized as follows. Section 2 describes traffic shaping and scheduling. Section 3 is devoted to the properties of the control scheme and section 4 to the impact of the control parameters. In section 5 we present a hardware realization of the scheme. We briefly discuss related work in section 6 and conclude in section 7.

2 Traffic Shaping and Scheduling

We consider a network architecture as shown in figure 1. Traffic from all sources are regulated at the user network interface (UNI), which is the network entry/access point. Inside the network, traffic generated by a source/stream is constrained to follow the same path — called a connection. From now onwards, we will use C_i interchangeably for a source, a stream as well as a connection.

We divide the time axis into successive periods of equal length, called a frame. Every internal or access link of the network can use an arbitrary time origin to define frames corresponding to that link. Every link can also choose an arbitrary frame size. However, to keep the discussion simple we will assume a global time origin and a uniform frame size F (in time units) for all links. Since the links may be of different speeds, the number of slots[2] a frame can accommodate could differ

[2]Slots are of fixed size, expressed in number of bits. A slot can hold a packet/cell. From now onwards, we will use

on different links. Over each link we view the corresponding frames of the link as traveling with fixed size slots from one end of the link to the other end (refer to figure 2). We denote by P_l the number of slots per frame on link l. We assume that propagation and transmission delays are fixed and known. In the rest of the discussion we focus on queuing delays only.

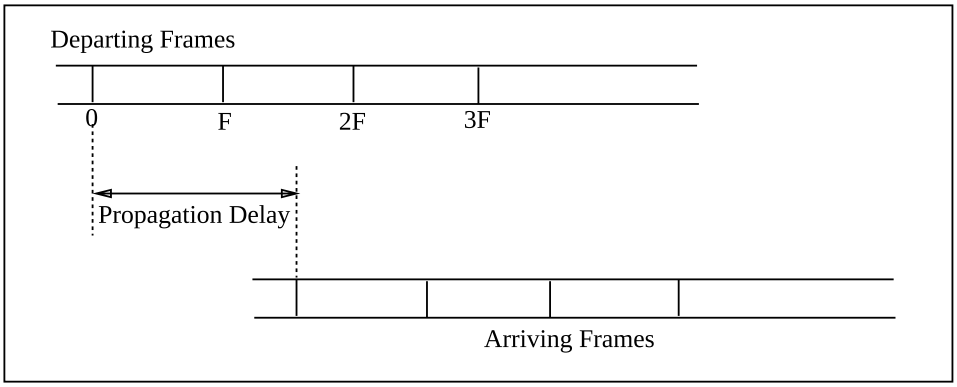

Figure 2: Departing and arriving frames on a link.

2.1 Multi-rate Shaping

A traffic shaper (figure 1) shapes the traffic from a source entering the network. The advantage of using a shaper is that it allows decomposition of packet delay into two components — delay in the shaper and delay in the network. The first of these components can be estimated from the statistical characterization of the source. The shaper can shape the traffic in a manner that can be succinctly characterized. This makes it easier for the network to bound the second component. From this point onwards, we will restrict our attention to the network component of the delay.

Traffic is shaped according to a smoothness criterion that limits the number of packet entries into the network over an interval (consisting of an integral number of frames). This essentially enforces a certain packet arrival rate over that interval. A multi-rate shaper allows two rates of packet arrival. A peak rate is enforced over short intervals. However, over a long interval, number of packet entry is regulated in such a fashion so that an average rate (much lower than the peak rate) is obtained. To formalize the traffic shaping we define below the "smoothness" of a traffic stream.

Definition 2.1 *A traffic stream C_i is said to be (W, M)* smooth *if,*

---

slot, packet and cell interchangeably.

```
INITIALIZE(C_i) /* Initialization for connection C_i */
    n_i ← 0 /* Current queue length of C_i */
    w_i^p ← 0; w_i^a ← 0 /* Current peak and average windows */
    m_i^p ← M_i^p; m_i^a ← M_i^a /* Current peak and average number of releasable packets */

TIMER(C_i) /* Operates on all active connections. */
    At the beginning of a frame:
        if (w_i^a ← (w_i^a + 1) mod W_i^a) = 0 then m_i^a ← M_i^a end if
        if (w_i^p ← (w_i^p + 1) mod W_i^p) = 0 then m_i^p ← M_i^p end if

SHAPER()
    1. On arrival of a packet from C_i:
        n_i ← n_i + 1;
        hold the packet in the buffer
    2. Release packets from C_i:
        while m_i^a > 0 ∧ m_i^p > 0 ∧ n_i > 0 do
            release the next packet in the network
            m_i^a ← m_i^a − 1; m_i^p ← m_i^p − 1; n_i ← n_i − 1
        end while
```

Table 1: Functions performed at the shaper

$$\sum_{k=jW}^{(j+1)W-1} m_k \leq M,$$

where $j = 1, 2, \ldots$, *and* m_k *is the number of packets from stream* C_i *that travel in the* k^{th} *frame.*

Informally, a traffic stream is (W, M) smooth if it carries no more than M packets in a time window of W frames.

We assume that the shaper smooths the traffic from the stream C_i into a (W_i^a, M_i^a) and (W_i^p, M_i^p) smooth flow, where $W_i^a \geq W_i^p$ and $M_i^a/W_i^a \leq M_i^p/W_i^p$. Essentially, the shaper enforces a maximum average rate of M_i^a cells over an interval (or window) of W_i^a frames and a maximum peak rate of M_i^p cells over an interval (or window) of W_i^p frames.

The shaper functions in the following way. It has a buffer space to hold the incoming packets. We assume it to be of infinite size. When a packet arrives from connection C_i, it is stored in the buffer. If releasing the packet into the network does not violate the imposed rate constraints (both peak and average), the packet is admitted. In table 1 we briefly outline a functional description of the shaper. It uses several counters. The counter n_i keeps track of the number of buffered packets belonging to connection C_i. The counters m_i^p and m_i^a represent the number of packets that can be admitted into the network in the current peak and average windows, respectively. If both of them are greater than zero, a packet is released. The counters w_i^p and w_i^a keep running counts of the number of frames expired in the current peak and average windows, respectively.

The `INITIALIZE()` function is invoked when a new stream is admitted. It initializes the counters for the connection. The `TIMER()` function is invoked at the beginning of every frame. It increments w_i^p and w_i^a at every invocation, and checks if the peak and/or average window(s) expires in this frame. It sets m_i^p and m_i^a to M_i^p and M_i^a at the beginning of a new peak and average window, respectively. The `SHAPER()` function determines if releasing incoming packets in the network violates the smoothness criteria of the corresponding connection. If they are not violated, packets are admitted to the network. Otherwise, they are held in the shaper buffer.

In order to compute end-to-end performance metrics, shaping traffic at the network entry point is not sufficient. The same traffic characteristics may not hold at intermediate switches unless a suitable multiplexing scheme is used there. In the next section we present a service discipline that maintains the original shape of the traffic throughout the network.

2.2 Service Discipline

From the modeling perspective, a switch can be viewed as a queue with rate of service equal to the output link speed. If a number of connections are multiplexed over the same output link, the effective rate of service received by individual connections depends on the service discipline used. We use a frame based non-work-conserving service discipline to be discussed shortly. The justification behind using such a scheme is to maintain traffic smoothness.

In our scheme, each connection C_i is guaranteed a minimum rate of R_i (higher than the average rate, and preferably less than the peak rate) slots per frame, negotiated at the time of connection set up. In a frame based allocation scheme, bandwidth is always allocated [6, 5] in multiples of a fixed quanta[3]. However, this allocation strategy tends to under-utilize the link capacity. To alleviate the problem, in our scheme, we allow R_i to be *non-integral*. The integral part of R_i can be easily allocated by scheduling $\lfloor R_i \rfloor$ slots per frame. However, fraction of a slot cannot be allocated to a connection. In order to meet the fractional requirement, the easiest approach is to allocate one slot to that connection. This is equivalent to allocation of $\lceil R_i \rceil$ slots per frame. However, this

[3]This quanta is usually one slot per frame.

scheme under-utilizes the link capacity, and to be precise, reduces to the aforementioned allocation by quanta scheme. We adopt the following approach to achieve the fractional allocation without reducing the link utilization:

We call a frame *busy* if all of its slots are used to fulfill the guaranteed rate of all the connections being served by it. If the frame has any slack slot after it has met the total requirement, it is called *non-busy*. A *(non-)busy period* refers to the interval (consisting of frames) where all the consecutive frames are (non-)busy. The allocation algorithm is different for a busy and a non-busy period.

In a non-busy period, each connection C_i is allocated at least $\lceil R_i \rceil$ slots in a frame. If there are still free slots left, they are distributed randomly making sure that C_i gets no more than M_i^p slots in a window of W_i^p frames[4]. In a busy period, a more complex scheduling is required. Unlike in a non-busy period, in a busy period there are not enough slots to over-allocate all connections with the next higher integral number of slots. In order to be as fair as possible, the scheduler first satisfies the integral portion of the slot requirements of all connections. Of the remaining slots, some connections with fractional requirements are allocated one slot each, while the fractional requirements of others are left unsatisfied. Slot requirements for the next frame are suitably adjusted to compensate for this unfair allocation. The rationale behind this allocation scheme is the following — by suitable admission control, in every frame the integral slot requirements of all connections are always satisfied. Of the connections with fractional requirements, some may get an extra slot while others may not. However, if a connection is deprived of its fractional requirements in consecutive frames, they accumulate and exceed 1. At that stage a slot must be allocated to it. Once again, the scheduler limits the maximum number of slots used by C_i to M_i^p in a time window of W_i^p frames. Table 2 gives a functional description of the scheduler.

In the description of the scheduler, r_i denotes remaining slot requirement of C_i in the current frame, w_i^p is the number of frames expired in the current peak window and m_i^p counts the number of slots the connection can get in the remainder of the peak window (i.e. next $W_i^p - w_i^p$ frames, including the current one). At the time of connection set up the `INITIALIZE()` function sets r_i to R_i. It also sets m_i^p to M_i^p and w_i^p to 0, following the same reason of the shaper. The `TIMER()` function resets m_i^p back to M_i^p every W_i^p frames. The `SCHEDULE()` function allocates the departing slots of link l. It first satisfies integral requirements of all the connections. Depending on the number of slots remaining, it arbitrarily selects some connections with fractional requirement and allocates one slot to each of them. If there are still slots left[5], connections may be given additional bandwidth to clear backlogs of packets, if any. Since the requirement of a connection gets fully satisfied in a non-busy period, r_i for the next frame is re-initialized to the original requirement R_i.

[4] This is done to ensure traffic smoothness.

[5] This is a non-busy frame.

Otherwise, the leftover requirement is carried over to the next frame along with R_i.

In all frame based allocation schemes known to us, bandwidth is allocated in incremental steps of one slot. This means that if we choose a frame size of 1ms, granularity of bandwidth allocation is approximately 425 Kb/s, assuming a packet size of 53-bytes. This coarse granularity of bandwidth allocation can lead to severe under-utilization of network resources. Our scheme overcomes this problem by allocating "non-integral" number of slots to a connection and there by allowing allocation of bandwidth at any arbitrary granularity.

3 Computing the QoS Envelope

In this section we prove some important properties of the traffic control scheme and derive worst case end-to-end bounds on delay and jitter. We first prove that under certain admission control criteria, the multiplexing at switches does not destroy the original smoothness of the traffic. That is to say that the output stream from a switch conforms to the same shaping envelope as the input stream to the switch. Consequently, end-to-end delay (jitter) in a multi-hop connection reduces to the product of the number of hops and the delay (jitter) suffered at a switch in isolation.

The first step towards proving the smoothness property is to determine the admission control criteria under which a connection C_i is guaranteed a bandwidth of R_i slots per frame. In the following lemma, and in the subsequent theorem we investigate the behavior of the algorithm in a busy period. Bandwidth guarantee is trivially satisfied in a non-busy period.

Lemma 3.1 *Let N_l be the number of connections being multiplexed on link l, r_i^k be the slot requirement of connection C_i at the beginning of the k^{th} frame and $\sum_{i=1}^{N_l} R_i \leq P_l$. Then $\sum_{i=1}^{N_l} r_i^k \leq P_l$ is satisfied for all frames in a busy period.*

Proof: Let us consider a busy period consisting of frames $K, K+1, \ldots, K+m$. We need to show $\forall k \in [K, K+m], \sum_{i=1}^{N_l} r_i^k \leq P_l$. We prove the lemma by induction.
Base Case: From the allocation algorithm we observe that $r_i^K = R_i$. Clearly,

$$\sum_{i=1}^{N_l} r_i^K = \sum_{i=1}^{N_l} R_i \leq P_l,$$

and hence, the assertion holds in the base case.
Inductive Hypothesis: Assume that the premise holds for all $k \leq K + j$. To prove that it holds for all k, we need to show that it holds for $k = K + j + 1$. From the allocation algorithm,

$$\sum_{i=1}^{N_l} r_i^{K+j+1} = \sum_{i=1}^{N_l} r_i^{K+j} + \sum_{i=1}^{N_l} R_i - P_l \leq P_l.$$

This completes the proof. $\square$

From now on, we would assume that a connection is admitted only if $\sum_{i=1}^{N_l} R_i \leq P_l$ is satisfied for all links l it passes through. This condition can be used as a simple *admission control* test for a new connection. The next theorem proves the minimum number of slots that C_i gets in the worst case.

Theorem 3.1 *If N_l be the number of connections being multiplexed on link l and $\sum_{i=1}^{N_l} R_i \leq P_l$, all C_i's get at least $(\lfloor mR_i \rfloor - 1)$ slots in m consecutive frames starting at the beginning of a busy period.*

Proof: Let us assume that a busy period starts from frame K. From the allocation algorithm we observe that, in each busy period C_i gets atleast $\lfloor R_i \rfloor$ slots. Therefore, it gets atleast $m\lfloor R_i \rfloor$ slots in m consecutive frames. Let us denote the fractional part of R_i by $\{R_i\}(= R_i - \lfloor R_i \rfloor)$. In the worst case C_i will not get any extra slot for its fractional requirement. Therefore, the fractional requirement builds up to $m\{R_i\}$. When $m\{R_i\} \geq 1$, C_i is given one extra slot, and C_i gets $\lfloor mR_i \rfloor$ slots. If $m\{R_i\} < 1$, C_i does not get any extra slot in the worst case, and the number of slots allocated to it becomes $(\lfloor mR_i \rfloor - 1)$. $\square$

Theorem 3.2 *If a connection C_i is (W_i^a, M_i^a) and (W_i^p, M_i^p) smooth before entering a switching node and $M_i^a \leq (\lfloor W_i^a R_i \rfloor - 1)$, then the output stream from the switch is also (W_i^a, M_i^a) and (W_i^p, M_i^p) smooth.*

Proof: The allocation algorithm enforces an upper bound of M_i^p packets in a period of W_i^p frames. Hence the output stream is (W_i^p, M_i^p) smooth. Now, since C_i is (W_i^a, M_i^a) smooth, the traffic that enters a switching node in a period of W_i^a frames is at most M_i^a. Since $M_i^a \leq \lfloor W_i^a R_i \rfloor - 1$, the traffic that enters the switching node in a period of W_i^a frames is served within that time, leaving no backlog. Hence, the output is also (W_i^a, M_i^a) smooth. $\square$

Theorem 3.2 is extremely important in deriving end-to-end performance. Since a traffic stream conforms to its original shape after passing through a switch, performance bounds for a multi-hop connection can be computed by simply adding the corresponding bounds at the individual switches on the path. In the subsequent discussion we will assume that $M_i^a \leq \lfloor W_i^a R_i \rfloor - 1$ for all streams at all nodes. This condition defines a lower bound on the service rate (R_i) required by a connection. The choice of R_i depends on other factors such as maximum tolerable delay, delay jitter etc. Next, we derive the worst case bound on the delay and jitter suffered at a switch in isolation. In order to compute delay and jitter we first derive the worst case queue length of a connection at a switch.

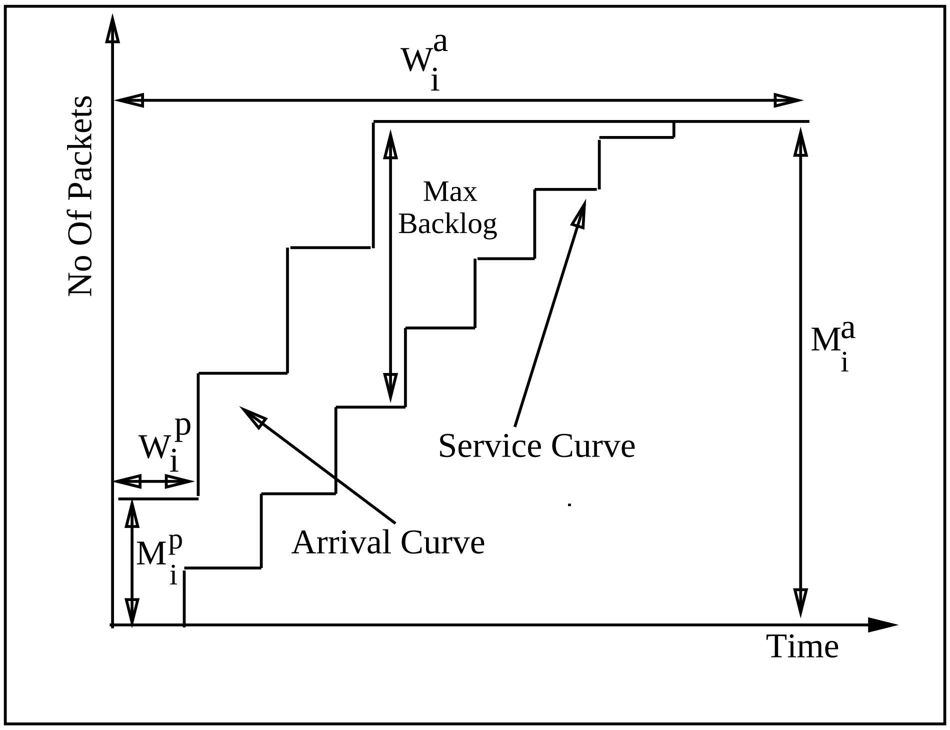

Figure 3: Delay bound computation.

Lemma 3.2 *The worst case queue length (backlog) of a connection C_i at a switch cannot exceed,*

$$B_i^{max} \leq max(M_i^p, M_i^a - \lfloor((M_i^a/M_i^p - 1)W_i^p - 1)R_i\rfloor + 1).$$

Proof: Consider a busy period. The worst case arises when a burst of size M_i^p cells comes every W_i^p frames (refer to figure 3). If the server can clear the burst before the arrival of the next one, the backlog can be at most M_i^p cells. If the next burst comes before the last one is cleared, backlog accumulates and reaches the maximum at the point shown in figure 3. For simplicity, we assume that M_i^p divides M_i^a. The worst case backlog can be computed by subtracting the number of packets transmitted from the number of packet that entered the switching node from the beginning of a busy period to the point of maximum backlog. Clearly, traffic input is at most M_i^a packets. To find the number of packets transmitted, we observe that number of frames expired from the beginning of the busy period to the point of maximum backlog is $(M_i^a/M_i^p - 1)W_i^p$. Out of these, at most one frame time may be wasted because of asynchrony between the arriving and departing frames. Hence, at least $(\lfloor((M_i^a/M_i^p - 1)W_i^p - 1)R_i\rfloor - 1)$ packets are transmitted. This completes the proof. $\square$

Theorem 3.3 *The maximum queuing delay (expressed in number of frames) suffered by a packet from connection C_i at a switch is,*

$$D_i^{max} \leq \lceil \frac{max(M_i^p, M_i^a - \lfloor((M_i^a/M_i^p - 1)W_i^p - 1)R_i\rfloor + 1)}{R_i} \rceil + 1$$

Proof: We derive the bounds for worst case delay suffered by a packet from a connection C_i by first computing the maximum backlog and then finding the time it takes to clear it. From lemma 3.2 we get the worst case backlog. In the worst case, the time it takes to clear this backlog is $\lceil B_i^{max}/R_i \rceil$ frames. Adding to that the worst case delay due to asynchrony in arriving and departing frames, we get the expression on the right hand side. □

Corollary 3.1 *The buffer space required at each switch to guarantee no packet loss for connection C_i is B_i^{max}.*

Proof: Keeping a buffer of the size of maximum possible buffer size is sufficient to guarantee no pack loss. The proof follows directly from theorem 3.2 □

Corollary 3.2 *The maximum jitter suffered by packets from a connection C_i at a switch is,*

$$J_i^{max} \leq D_i^{max}$$

Proof: The delay jitter is defined to be the difference between the maximum and minimum delay. Since the minimum queuing delay suffered by a packet is zero the result is obvious. □

Theorem 3.4 *The maximum end-to-end queuing delay and jitter suffered by a packet from an H-hop connection C_i may not exceed, HD_i^{max}.*

Proof: From theorem 3.3 and corollary 3.2 we get, maximum delay and delay jitter suffered by a packet from a connection C_i is D_i^{max}. From theorem 3.2 we get, traffic from connection C_i maintains its smoothness all through the network. Hence, the delay and the jitter bound for an H-hop connection is the sum of the bounds at individual switches. This concludes the proof. □

The results in this section are derived under the assumption of a uniform frame size all over the network. They can be extended when different switches/links use different frame sizes. In that case the shape of the traffic would not be the same all through the network. However, knowing the input characterization of a stream entering a switch, the output can be characterized. Hence, to compute the end-to-end performance bounds, bounds at each switch on the path has to be determined individually.

4 Choosing the Control Parameters

In the following discussion we study the impact of different control parameters on worst case delay using numerical examples. Note that the averaging interval W_i^a does not appear in the expression for delay bound. However, $R_i^{mean} = M_i^a/W_i^a$ defines the average rate of arrival and determines the lower bound on the allocated bandwidth. On the other hand $R_i^{peak} = M_i^p/W_i^p$ defines the peak arrival rate. To get a quantitative feeling, in figure 4 we plot the maximum delay D_i^{max} against R_i^{peak}/R_i^{mean}. As expected, we observe longer worst case delay for higher mean to peak ratio. We also observe a sharp improvement in delay bound with an increase in rate of service.

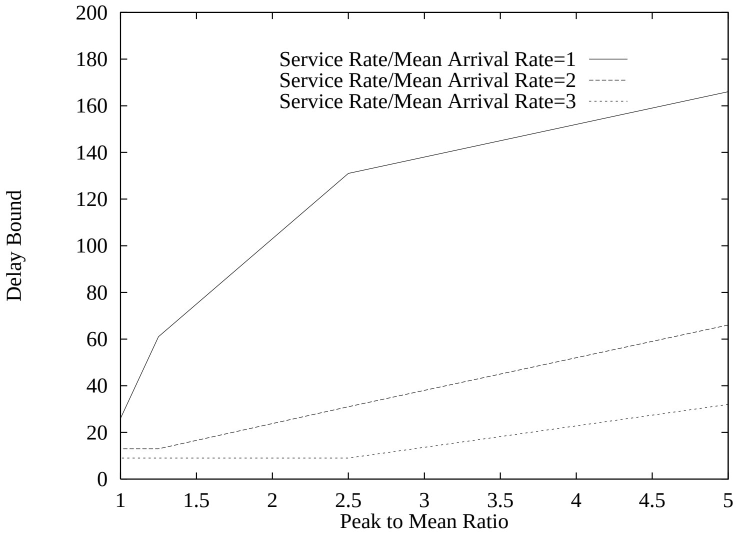

Figure 4: Effect of burstiness on delay bound. In this example $W^a = 200$, $M^a = 1000$ and $W^p = 125$.

The worst case delay not only depends on the peak and mean arrival rates and rate of service, it also depends on the averaging window size. The following plot examines the impact of M_i^p and W_i^p for the same value of $R_i^{peak} = M_i^p/W_i^p$. It is apparent that for the same peak rate of arrival, longer averaging window size and hence larger bursts lead to longer delays.

As mentioned earlier, traffic smoothing at the network entry point allows one to separate delay into two components — delay in the shaper buffer and delay in the network. The first component of the delay depends on the statistical characteristics of the source and the shaping envelope chosen by the user. The second component is independent of the actual characteristics of the source and depends on the shaping parameters and network load and can be bounded by the network. The user, depending on the cost and performance tradeoffs, can choose the most appropriate parameters.

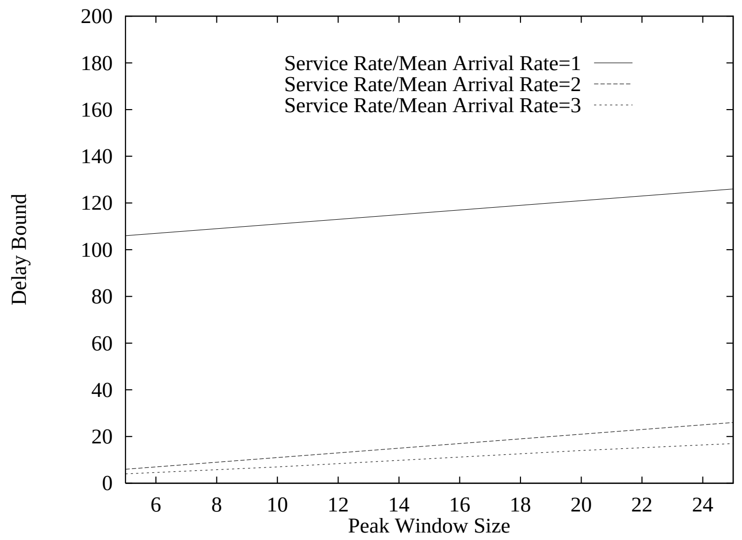

Figure 5: Effect of M^p on delay bound. In this example $W^a = 200$,$M^a = 1000$ and $R^{peak}/R^{mean} = 2$.

5 Realization of the Scheme

In this section we briefly outline the design of a buffer manager (figure 6), the key component used to implement the scheduling mechanisms at the switch. The architecture of the buffer manager is based on the design suggested in [2]. However, our scheme does not require the sequencer, the most complex component used in [2].

The buffer manager keeps track of the requirements of each active connection C_i in the current frame. It also stores pointers to the Head and Tail of each connection queue. At any time a packet at the head of connection queue is eligible to be scheduled for transmission if it satisfies the eligibility criteria described in the scheduling algorithm (refer to section 2).

The buffer manager consists of a cell pool, an idle address FIFO, a table-look-up memory and a processor. The cells are stored in the cell pool. The table-look-up-memory, referred to as *connection table*, stores necessary control information. The idle address FIFO contains addresses of the current empty cell locations in the pool.

The buffer manager maintains a virtual queue for each connection by linking the packets through pointers. Cells stored in the cell buffer are tagged with the pointer to the next cell from the same connection, if any. When a cell arrives, it is stored in the cell pool at the address given by the idle address FIFO. While the cell is being written into the cell pool, its virtual channel identifier is extracted by the processor and the corresponding entry in the connection table is modified. If the connection queue is empty, the buffer address of the cell is written into both Head and Tail fields.

Otherwise, the buffer address is written into the Tail field only. Pointers are suitably adjusted to add the arriving cell at the end of the connection queue.

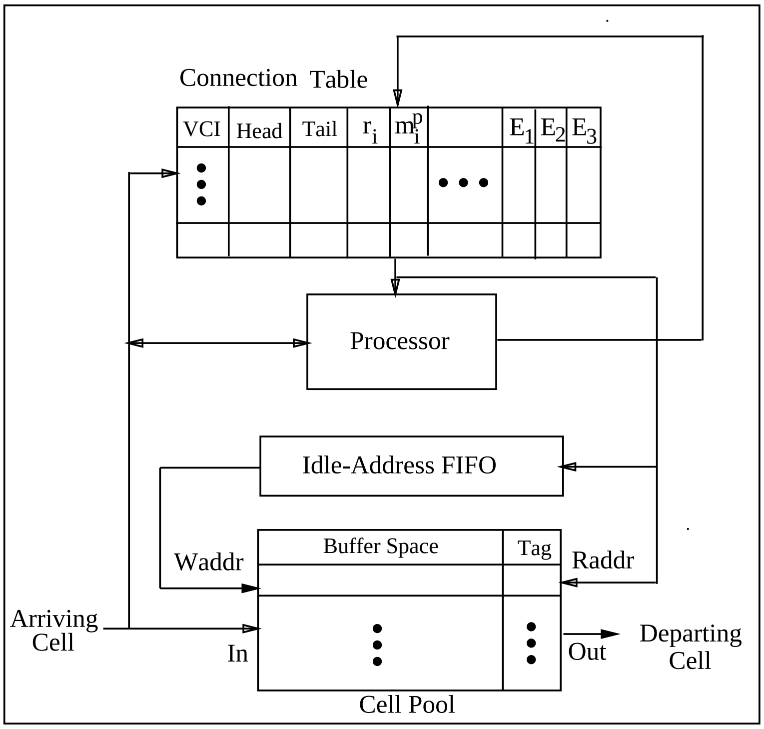

Figure 6: Architecture of the buffer manager.

The connection table stores three eligibility flags, $E_i, i = 1, 2, 3$, for all active connections. The flag E_1 is set when a stream has an outstanding integral requirement, the flag E_2 is set when it has an outstanding fractional requirement and flag E_3 is set if the stream has not exceeded its maximum allowed limit[6]. Flags are set only if a connection has a packet to send. This flag can be set/reset by a local logic and does not need processor cycles.

During transmission of a cell, an eligible connection is selected. The buffer address of the cell at the head of the connection is stored in the connection table and is extracted by the processor and fed to the Raddr lines of the cell pool. The buffer address of the next cell in the queue, if any, is written into the Head field of the corresponding entry in the connection table. The processor also makes necessary changes to the control information stored in the connection table.

[6] A connection C_i can send at most M_i^p cells in a time window of W_i^p frames.

6 Related Work

Traffic control in integrated services packet switched networks has received tremendous attention in the last few years. For a very good survey on this topic refer to [1]. In the rest of the section we briefly review some of the related literature on rate based traffic control. The potential benefit of traffic regulation at the switching nodes has been analyzed by Cruz in [3, 4]. Using a two parameter characterization of traffic he derives bounds on burstiness and end-to-end delay. For some cases they are shown to increase exponentially with the number of hops in a connection. Golestani [5] proposes a frame based scheme for peak rate constrained sources and stop and go queuing at the nodes. Kalmenak et. al. [6] propose a conceptually similar scheme using a hierarchy of frame sizes. Though these schemes are very simple, they require network wide standardization of frame sizes. Besides, allocating bandwidth at the peak rate can lead to considerable wastage of network resources. Another rate-based approach to traffic management is the virtual clock scheme proposed by Zhang [10]. The virtual clock scheme can be viewed as a generalization of the round robin service discipline. Its distinguishing feature is the provision of fairness and minimum throughput guarantees to competing connections. However, no bound on delay or jitter was obtained in [10]. Parekh et. al. [7] analyze the network behavior for leaky bucket constrained sources and head-of-the-line processor sharing service discipline at the switching nodes. Zhang et. al. [9] propose a static priority based rate controlled scheme. In their scheme, a connection is assigned a link deadline and a corresponding scheduling priority on each link along its path. This scheme can guarantee tight delay and jitter bounds. However, it is not clear how to choose the link deadlines to achieve a specific quality of service.

The schemes closest to ours are [5, 6, 9, 8]. Our work differs from [5, 6] in that we consider multi-rate constrained traffic as opposed to peak rate constrained traffic used in these schemes. Also, our scheme does not require network wide standardization of frame sizes and is more flexible in accommodating diverse throughput requirements of different applications. The difference between our approach and the one suggested in [9, 8] is in the service discipline at the switches. We use a frame based scheme in contrast to calendar based approach used in [9, 8]. Unlike [9, 8], we use a very simple bandwidth allocation algorithm to achieve a specific quality of service.

7 Concluding Remarks

We present a framework for traffic control in an integrated services ATM network. The proposed scheme comprises of two components: a shaping mechanism at the network entry point, and a service discipline at the switches. The shaper enforces dual rates, namely a mean and a peak rate.

At the switches, each connection is guaranteed a minimum bandwidth but, may receive a better service if there is unused capacity. We propose a frame based bandwidth allocation scheme. Unlike other frame based schemes, our scheme allows allocation of bandwidth at any arbitrary granularity. We outline a hardware realization of the control mechanism and derive worst case end-to-end delay and jitter bounds.

References

[1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne. Real-Time Communication in Packet-Switched Networks. *Proceedings of the IEEE*, 82(1), January 1994.

[2] H. J. Chao. Architecture Design for Regulating and Scheduling User's Traffic in ATM Networks. In *Proceedings, SIGCOMM*, August 1992.

[3] R. L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transsactions on Information Theory*, 37, 1991.

[4] R. L. Cruz. A Calculus for Network Delay, Part II: Network Analysis. *IEEE Transsactions on Information Theory*, 37, 1991.

[5] S. J. Golestani. A Framing Strategy for Congestion Management. *IEEE Journal on Selected Areas in Communication*, 9(7), September 1991.

[6] C. R. Kalmanek, H. Kanakia, and S. Keshav. Rate Controlled Servers for Very High Speed Networks. In *Proceedings, GLOBECOMM*, December 1990.

[7] A. K. Parekh and R. G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Network: The Single Node Case. *IEEE/ACM Transactions on Networking*, 1(3), June 1993.

[8] H. Zhang. Service Disciplines for Packet-Switched Integrated-Services Networks. *Ph.D. Thesis, University of California, Berkeley*, 1993.

[9] H. Zhang and D. Ferrari. Rate Controlled Static Priority Queuing. In *Proceedings, INFOCOM*, 1993.

[10] L. Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings, SIGCOMM*, 1990.

```
INITIALIZE(C_i) /* Initialization for connection C_i. */
    r_i ← R_i /* Slot requirement in current frame */
    w_i^p ← 0; m_i^p ← M_i^p /* To enforce peak rate constraint */

TIMER(C_i) /* Operates on all active connections */
    At the beginning of a frame:
        if (w_i^p ← (w_i^p + 1) mod W_i^p) = 0 then m_i^p ← M_i^p end if

SCHEDULE(l) /* Scheduling over link l */
    At the beginning of a frame:
        1. Initialize
            N ← P_l /* Number of slots in a frame */
        2. Satisfy integral requirements:
            while (∃C_i :: r_i ≥ 1 ∧ m_i^p > 0) ∧ N > 0 do
                allocate next slot to C_i
                r_i ← r_i − 1; m_i^p ← m_i^p − 1; N ← N − 1
            end while
        3. Satisfy fractional requirements:
            while (∃C_i :: r_i > 0 ∧ m_i^p > 0) ∧ N > 0 do
                allocate next slot to C_i
                r_i ← r_i − 1; m_i^p ← m_i^p − 1; N ← N − 1
            end while
            if N > 0 then EndofBusyPeriod ← true
        4. Distribute slack slots:
            while (∃C_i :: m_i^p > 0) ∧ N > 0 do
                allocate next slot to C_i
                m_i^p ← m_i^p − 1; N ← N − 1;
            end while
        5. allocate remaining slots to non-real-time traffic
        6. Compute requirements for next frame:
            if EndofBusyPeriod then
                    r_i ← R_i;
                    EndofBusyPeriod ← false
            else
                    r_i ← r_i + R_i
            end if
```

Table 2: Switch scheduling functions.