A Generic Architecture for Programmable Traffic Shaper for High Speed Networks

Krishnan K. Kailas[†] Ashok K. Agrawala[‡]
{krish, agrawala}@cs.umd.edu
[†]Department of Electrical Engineering
[‡]Department of Computer Science
University of Maryland
College Park, MD 20742, USA

S. V. Raghavan
svr@iitm.ernet.in
Department of Computer Science & Engineering
Indian Institute of Technology, Madras, India.

Abstract

Traffic shapers by preventing congestion and smoothing the traffic, play an important role in realizing the traffic control schemes employed in high speed networks to ensure the Quality of Service (QoS) requirements of the application. In this report, we present a generic architecture for programmable traffic shaper for high speed networks. The programmability of the proposed architecture is illustrated by implementing some of the existing traffic shaping schemes. The architectural design issues of the proposed scheme are described and discussed.

1 Introduction

Network applications with real-time constraints and high bandwidth requirements such as distributed multimedia applications are made feasible with the advent of Asynchronous Transfer Mode (ATM) networks [10]. The call admission control (CAC) functions of ATM network and the associated protocols allow the applications to specify the traffic characteristics as well as quality of service (QoS) requirements at the call setup time. In order to guarantee these negotiated QoS requirements, several traffic policing and congestion avoidance schemes are employed. One such scheme is the *traffic shaper*, which is usually located at the source end to regulate the cells admitted into the network, there by preventing congestion and providing smooth traffic.

In this report, we present a generic architecture of a programmable traffic shaper. The generic architecture discussed here may be used for implementing any of the currently existing traffic shaping techniques. Some of the hardware architectures considered and the implementation issues of the proposed architecture are described in section 5. Section 6 gives final conclusion and some pointers to future research in this topic.

2 Traffic Shaping

Providing QoS guarantees and maximizing the bandwidth utilization are two conflicting requirements, especially when bursty traffic is to be supported by the network. Several bandwidth management schemes such as peak-rate allocation, minimum throughput allocation and bursty traffic specification, and a number of congestion control schemes such as preventive, reactive and adaptive congestion control schemes were proposed to address this issue [9, 1]. Hence, traffic shaping at the source is essential to guarantee QoS and to avoid congestion.

The basic idea behind traffic shaping is to control the input traffic so that it is amenable to the scheduling mechanism at the switches for providing the required QoS guarantees. Several mechanisms have been proposed for traffic shaping. These may be broadly classified into two groups viz., Leaky bucket type mechanisms and window based mechanisms. Performance comparison of the above mentioned traffic shaping schemes can be found in [8]. A flexible shift register based traffic shaping (SRTS) scheme [4, 5, 6, 7] proposed recently claims to have better the performance figures in comparison with leaky bucket scheme. SRTS scheme is based on the temporal profile of the packet stream admitted by the shaper over a fixed time frame. We further generalize this idea in our proposed generic traffic shaping scheme which can implement any of the existing schemes separately or as a combination of them.

3 Towards A Generic Traffic Shaping Scheme

3.1 Motivation

Our motivation for a generic traffic shaping scheme is based on the following general properties and requirements for good source traffic shaping:

1. The shaping scheme should be easily programmable.
2. The traffic shaper should be independent of any specific policing mechanism.
3. The shaping scheme should be able to describe a wide range of traffic behaviors [3].
4. The shaping rules should make it easy to describe traffic patterns to the network [3].
5. The shaping scheme should be easy to police [8].
6. The scheme should be fast, simple and cost effective to implement in hardware so that traffic shaping can be done in real-time [8].
7. The dynamic reaction time of the mechanism should be short to avoid flooding of the buffers in the network [8].

The first two requirements are for making the traffic shaper generic so that the best traffic shaping mechanism can be chosen based on the traffic characteristics and QoS requirements. The items 3–5 are more applicable to the technique used for policing the traffic. The last two requirements are meant to cope with the volume and burstiness of the variable bit rate traffic generated by video/audio sources and to maintain their isochronous traffic characteristics.

3.2 Traffic shaping based on temporal profile

We make use of the temporal history profile of the input as well as output streams to monitor and shape the traffic flow. The block schematic of the traffic shaper is shown in Fig. 1. The temporal profiles of the input and output streams are stored in the traffic shaper. A programmable windowing mechanism is provided to access this history profile information. The binary decision to admit or hold a packet is computed in real-time based on the information from multiple user-defined observation-windows defined on the temporal profile. A fast programmable ALU is provided for computing the user-defined admission function f_a in real-time. The temporal history profile is updated at each time slot τ determined by the maximum permissible rate of the network. Hence, peak rate limiting is implicit in our scheme by design.

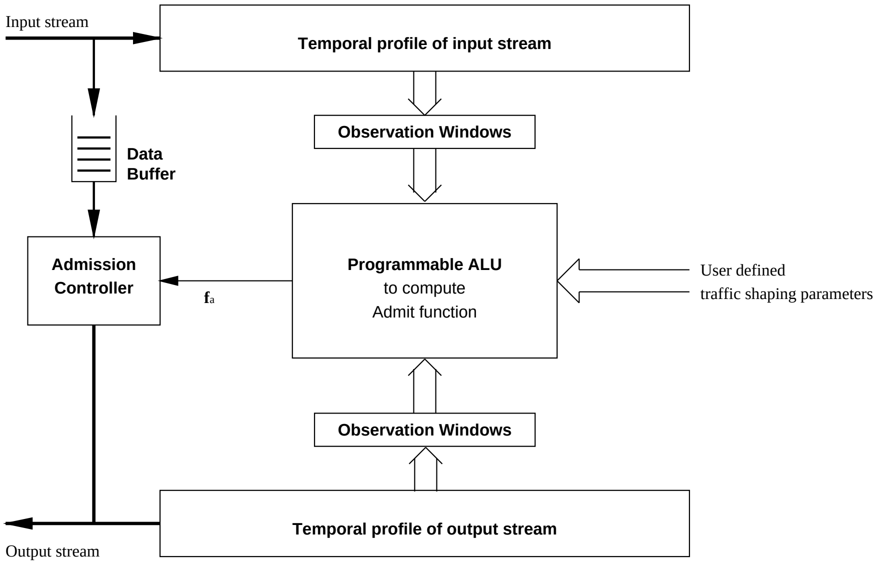

Figure 1: Block schematic of a generic programmable traffic shaper

In the following three sections we illustrate the programmability of the proposed scheme by implementing three common admission control schemes viz., the leaky bucket (LB) scheme, window-based (Jumping window) schemes and SRTS scheme.

3.3 Implementation of Leaky Bucket Scheme

Leaky bucket (LB) scheme and its variants basically emphasize bandwidth enforcement [2] by limiting the cell admission rate to a negotiated rate. The cells are admitted in this scheme only when it can draw a token from a token pool of finite capacity, say σ. The token pool is replenished with new tokens at a fixed rate. This scheme can be implemented very easily by making use of the temporal profile of the output stream with slot time $\tau = 1/r$, where r is the peak rate. The admission function for LB scheme may be defined as:

$$f_a = \begin{cases} 1 & \text{if } n(W_\sigma) < \sigma \\ 0 & \text{otherwise.} \end{cases}$$

where $n(W_\sigma)$ is the number of cells admitted in a window of σ time slots.

3.4 Implementation of Window-based Traffic shaping Schemes

Window-based schemes can be directly implemented on our proposed generic traffic shaper. For example, a Jumping Window mechanism can be implemented with an admit function defined by:

$$f_a = \begin{cases} 1 & \text{if } n(W_{(0,i)}) < N \text{ for } 0 \leq i \leq \delta \\ 0 & \text{otherwise.} \end{cases}$$

where $n(W_{(0,i)})$ is the number of cells admitted in the window $(0, i)$ and δ is the window width. i is incremented at every time slot and reset to 0 when it reaches δ for "jumping" to the new window.

3.5 Implementation of Shift Register Traffic Shaper

The Shift Register Traffic Shaper (SRTS) is a new window based traffic shaping scheme. SRTS can adjust the burstiness of the input traffic to obtain reasonable bandwidth utilization while maintaining statistical service guarantees [4, 5, 6, 7]. The SRTS admit control function makes use of the temporal history (maintained by a shift register) of the cells admitted. The admit function is defined by:

$$f_a = \begin{cases} 1 & \text{if } (n(W_1) < n_1) \wedge (n(W_2) < n_2) \wedge (n(W_3) < n_3) \wedge \cdots \\ 0 & \text{otherwise.} \end{cases}$$

where $n(W_i)$ is the number of cells admitted in i th window. This scheme permits controlled burstiness by choosing appropriate values for window widths and maximum number of cells admitted (n_i) in each window.

4 Generic Architecture for Programmable Traffic Shapers

In this section we describe the final architecture and two intermediate architectures which lead to the final architecture of the programmable traffic shaper. The hardware complexity and performance issues of each architecture are compared. The hardware architecture required for storing the temporal profile and windowing is identical for input as well as output streams. Hence, in the rest of this report we will be describing the architecture of only one stream for simplicity.

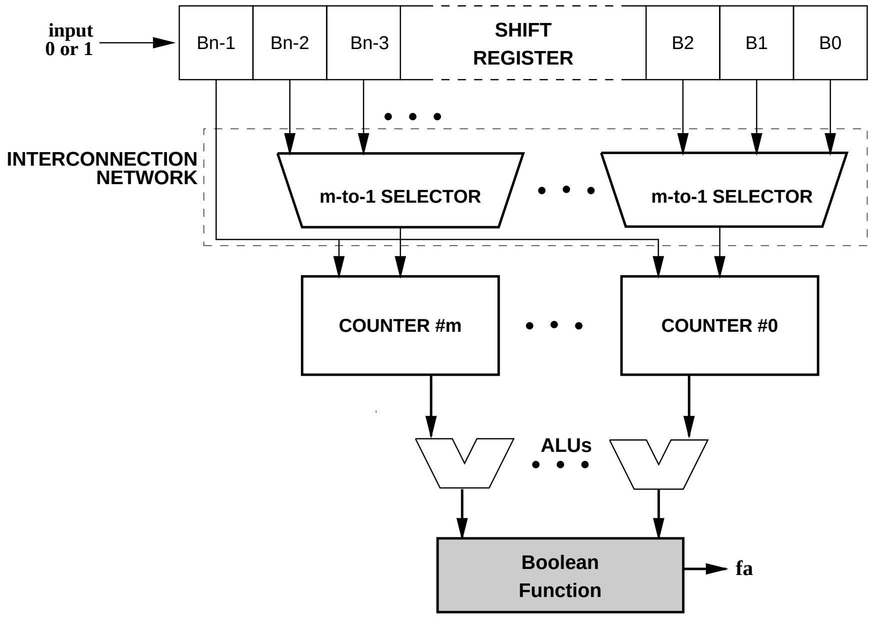

Figure 2: Traffic shaper architecture using shift register

4.1 Shift register and interconnection network

Fig. 2 shows a preliminary architecture based on a shift register to keep track of the temporal profile as in the SRTS scheme proposed in [4]. A n-bit long shift register is used to store the temporal history of the cells admitted in the last n time slots, where each time slot $\tau = 1/N$ and N is the current peak rate. A 0 in the shift register indicates absence and a 1 indicates presence of a cell arrival or admission in the referred time slot. The observation windows are realized by m number of m-to-1 selectors, each one has an address input to specify the shift register stage to be selected. A set of m counters are used to count the number of cells admitted or arrived within each window by incrementing the count when a 1 enters and decrementing the count when a 1 leaves out of the window associated with the counter.

The ALU can access all the counter values as registers and compute the user-defined admit function f_a.

For example, in the SRTS scheme the ALU needs to evaluate the admit function $f_a = (n(W_1) < n_1) \wedge (n(W_2) < n_2) \wedge (n(W_3) < n_3) \wedge \cdots$ at each time slot where $n(W_i)$ is the number of cells admitted in i^{th} window and n_i is the threshold value of the number of cells in i^{th} window for traffic shaping.

The admit function is calculated at the end of each time slot. This may cause a cell arriving during an idle slot but otherwise eligible to be admitted to be delayed by τ in the worst case. In order to minimize this admission delay at the traffic shaper, the "soft" discretization technique proposed in [4] may be used. The idea is to sample the input at a higher speed within each time slot τ using a main clock (main-CLK). As shown in Fig. 3 the τ clock for the shift register can be derived using 2 counters viz., the slot counter (SC) and elapsed time counter (ETC). The ETC will start counting with every idle slot and will be disabled by a cell arrival if f_a evaluates to admit the cell. The main clock speed is limited by the speed of the ALU module to compute f_a.

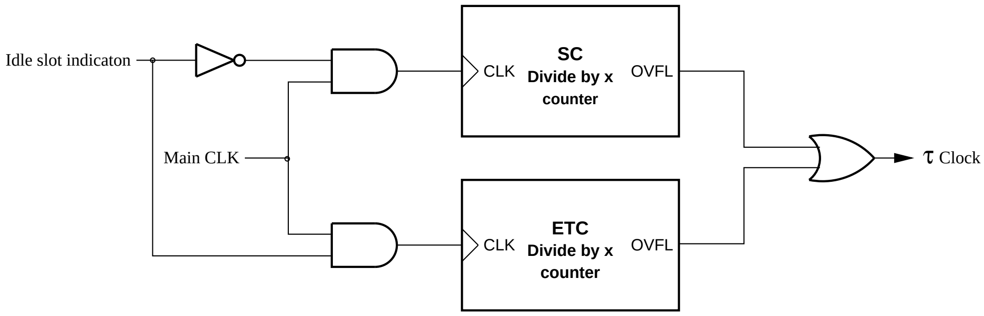

Figure 3: τ clock generation from main-CLK for "soft" discretization

The basic problem with this architecture is that all the windows should start from one common point. A solution to this problem is to have an interconnection network (say, a crossbar switch) to connect any shift register stage to any counter so that the window boundaries can be anywhere within the shift register. This will result in a very complex circuit to make it fully programmable.

4.2 Temporal history FIFO

In order to overcome the limitations of the preceeding architecture a novel architecture based on a hardware FIFO structure is proposed. Fig. 4 shows the new architecture. The FIFO contains the cumulative sum of all cells admitted till that stage in the temporal profile of the stream. The m-bit adder (or incrementer circuit) at the input will add the contents of last stage's value with the current binary input. Hence the ith element of the FIFO may be expressed as: $S_i = \sum_{i=0}^{n} S_i$. The sum S_i can overflow to 0 when S_{i-1} is 2^m. However, the number of cells admitted in any window $n(W_{(i,j)})$ can be computed as:

$$n(W_{(i,j)}) = \begin{cases} S_i - S_j & \text{if } S_i - S_j > 0 \\ S_i - S_j + 2^m & \text{if } S_i - S_j < 0 \end{cases}$$

where $m = \lceil \lg\ n \rceil$ is the width of the FIFO element in bits, and n is the length of the FIFO. The above computation can be implemented by a 2-stage adder and a simple logic circuit to compare the MSBs of S_i and S_j.

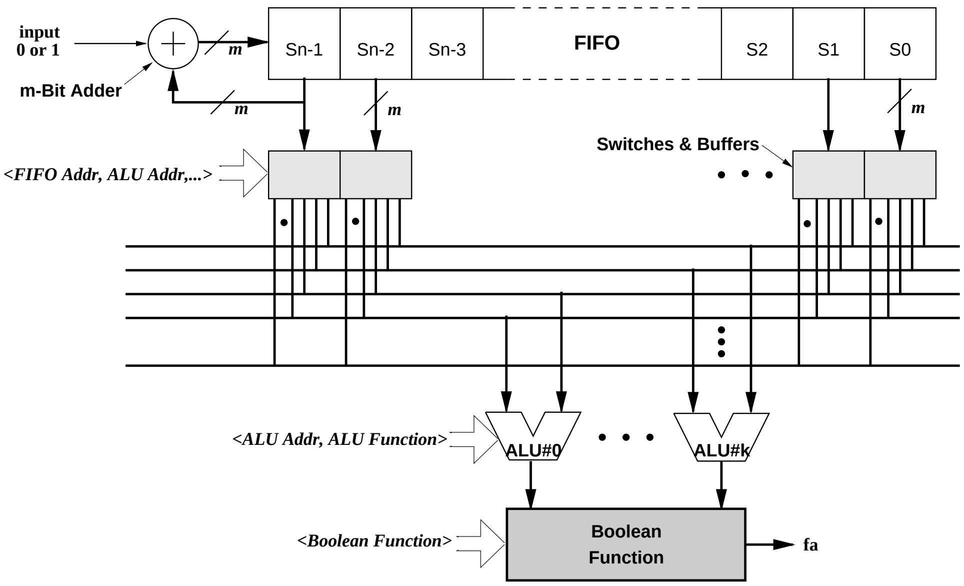

Figure 4: Generic Traffic Shaper Architecture using FIFO

The contents of the FIFO can be accessed by the k ALUs as a register bank through simple switches and buffers as shown in the Fig. 4. The admit function f_a is computed as a boolean function of the ALU outputs.

The FIFO may be implemented in two ways:

1. as a circular queue in RAM with one address register to keep track of the queue head. The address of any element in the FIFO can be computed from the queue head pointer by an adder.

2. as a multi-bit shift register similar to the architecture mentioned in the last section.

This generic architecture provides the programmability and flexibility needed to realize any traffic shaping function. However, the maximum length of the history profile is bounded by the length of the FIFO. One technique to over come this limitation is to make the architecture scalable so that multiple chips can be cascaded together to get a longer history profile. Another technique is to store the history profile at discrete intervals in contrast with the present scheme with continuous profile of every time slot. We have decided to choose the latter scheme primarily because it is difficult to implement fast bit-slice ALUs needed in the scalable architecture. The next section discusses the basic concepts which motivated our decision and hardware implementation of the scheme.

4.3 Time granularity of history profile

It can be argued that better traffic shaping and policing can be achieved if the admit function is computed based on a longer temporal history of the input and output stream than a shorter one. We introduce the notion of *time granularity* to address this problem. The idea is to increase the effective length of the temporal profile sacrificing the time granularity of measurement. As shown in Fig. 5, with the help of a counter the binary inputs are accumulated for a period of ΔT time slots and then moved into the FIFO. A l-bit programmable counter can vary the time granularity from $\Delta T = \tau$ to a maximum of $\tau \cdot 2^l$ corresponding to the history profile length increase from n to $n \cdot \Delta T$. With the above scheme, by choosing an appropriate value for ΔT based on the requirement of application, it is possible to achieve an optimal trade-off between time granularity and window width.

4.4 VLSI design issues

The architecture we have proposed can be directly implemented in VLSI. The maximum frequency of the main-CLK is limited by the speed of the

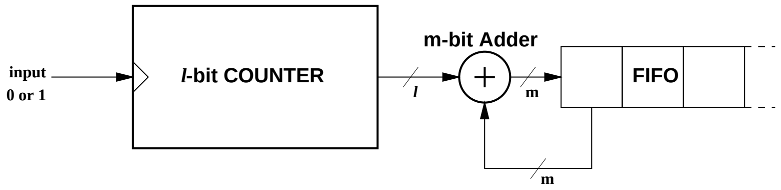

Figure 5: Circuit for programmable time granularity

ALU and the associated logic for computing the admit function. Hence the time critical path for our architecture is identified as FIFO accessing logic and the ALU. Since our computations are done in parallel by multiple ALUs with specialized hardware for computations, the main-CLK speed as high as 50MHz is feasible with the 1.2μ CMOS technology. Another factor which decides the main-CLK speed is the the size of the counters (see Fig. 3) used to derive τ from the main-CLK.

5 Comments and Conclusion

In this report we have proposed a generic an architecture for a programmable traffic shaper for high speed networks. The programmability of our architecture is illustrated by implementing various schemes. The hardware complexity and implementation feasibility issues of the proposed architecture are explored. The proposed architecture may be used for implementing adaptive congestion control schemes because of its capabilities to program the traffic shaping scheme or parameters on-line. The ability to characterize and control the burstiness of the traffic as a function of input and output stream makes this architecture a potential candidate in implementing congestion control schemes for multimedia type of real-time applications. The feasibility of implementing a scheme for selectively discarding cells making use of this architectural framework for supporting specific standards like MPEG video is yet to be explored. The hardware simulation and VLSI implementation of the proposed architecture need to be carried out.

6 Acknowledgment

We would like to thank S. Radhakrishnan of Indian Institute of Technology, Madras, India for suggesting a hardware implementation of “soft” discretization technique.

References

[1] Chinatsu Ikeda and Hiroshi Suzuki. Adaptive congestion control schemes for ATMLANs. In *Proceedings of IEEE Infocom'94*, pages 829–838, June 1994.

[2] Duke Hong and Tatsuya Suda. Congestion Control in ATM Networks. *IEEE Network Magazine*, pages 10–16, July 1991.

[3] Craig Partridge. *Gigabit Networking*, chapter 11. Professional computing series. Addison-Wesley, Reading, Mass., 1994.

[4] S. Radhakrishnan, S. V. Raghavan, and Ashok K. Agrawala. A Flexible Traffic Shaper for High Speed Networks: Design and Comparative Study with Leaky Bucket. "to appear in Computer Networks and ISDN Systems".

[5] S. Radhakrishnan, S. V. Raghavan, and Ashok K. Agrawala. A Flexible Traffic Shaper with Variable Burstiness for High Speed Networks. In *Proceedings of JENC6 International Conference* , Tel Aviv, Israel, May 1995.

[6] S. Radhakrishnan, S. V. Raghavan, and Ashok K. Agrawala. Design and Performance Study of a Flexible Traffic Shaper for High Speed Networks. September 1995.

[7] S. Radhakrishnan, S. V. Raghavan, and Ashok K. Agrawala. Performance Comparison of a New Traffic Shaper and Leaky Bucket for Bursty Real-Time Traffic. Communicated to MMM'95 International Conference on Multi-Media Modelling to be held in Singapore, November 1995.

[8] Erwin P. Rathgeb. Modelling and Performance Comparison Of Policing Mechanisms for ATM Networks. *IEEE Journal on Selected Areas in Communications*, 9(3):325–334, April 1991.

[9] J. Turner. Managing bandwidth in ATM networks with bursty traffic. *IEEE Network*, 6(5):50–58, September 1992.

[10] Ronald J. Vetter. ATM Concepts, Architectures, and Protocols. *Communications of ACM*, 38(2):30–38, 109, February 1995.