Implementations of Traffic Control Mechanisms for High Speed Networks[†]

Célio V. N. Albuquerque
Univ. of California, Irvine
celio@ics.uci.edu

Marcio Faerman
Univ. of California, San Diego
mfaerman@cs.ucsd.edu

Otto Carlos M. B. Duarte*
Univ. Federal do Rio de Janeiro
otto@gta.ufrj.br

COPPE/EE – Programa de Engenharia Elétrica
P.O. Box 68504 – CEP 21945-970 – Fax: +55-21-290-6626
Rio de Janeiro – RJ - Brasil

Abstract. This paper proposes a traffic policing mechanism for multimedia communication systems with quality of service guarantees, called Slotted Moving Window (SMW). Moreover, it also presents efficient implementations of SMW and several other well known traffic policing mechanisms, for comparison purposes. SMW main feature is the capability to accurately control the mean cell rate in a cost-effective approach. In addition, SMW is characterized by low implementation complexity, low memory requirements and use of a single low precision timer. Its key idea consists in aggregating arrival events in order to minimize processing and buffering requirements. The comparative analysis concentrates on the number of timers and their precision, the required memory, the processing time, the maximum burst duration and the violation probability. Finally, the results confirm the efficiency and low implementation complexity of both Leaky Bucket and the Slotted Moving Window mechanism.

1. INTRODUCTION

One of the main issues in high speed networks is to protect the network resources against abusive traffic sources that violate the quality of service (QoS) parameters negotiated during the connection setup, being potentially harmful to the QoS of other connections. The control mechanism responsible for preventing possible violations of the QoS parameters is known as policing mechanism [1]. Policing mechanisms must be efficient, feasible, cost-effective and flexible, requiring a low implementation complexity. Several mechanisms have been proposed. Among the most mentioned mechanisms in literature are leaky bucket [2,3], its variants [4], and several window-based mechanisms. Among these last ones it can be pointed out jumping window, sliding window and exponentially weighted moving average [2,5,6].

Among policing mechanisms, one that presents simultaneously advantages referent to the traffic memorization, the mean bit rate control in addition to the restriction of the burst length, is the sliding window mechanism. However, this mechanism has been criticized because of its implementation complexity [2,5]. Dittmann *et al* [6] proposes an implementation scheme that requires a complexity level not so critic.

This work proposes and analyzes an adjustable and cost-effective traffic control mechanism called Slotted Moving Window (SMW). SMW is a window-based policing mechanism that requires an implementation complexity inferior to the sliding window mechanism complexity and that is as restrictive in relation to burst duration. This mechanism also has the advantage of being extremely flexible. According to user and network contract, the behavior of slotted moving window can be configured to range from the simplicity of jumping window to the wide set of advantages offered by sliding window.

Implementations of several policing mechanisms are also presented. The implemented mechanisms are leaky bucket, jumping window, continuous sliding window, discrete sliding window and Slotted Moving Window. The implementations are developed over a high performance implementation architecture [8]. A comparison of the results focusing on the implementation complexity is presented considering required memory, number of timers and their precision and mean processing time.

Section 2 presents a high performance development environment used in the implementation of the above mentioned policing mechanisms. Section 3 describes well-known policing mechanisms and proposes their implementation. Section 4 proposes the slotted moving window mechanism and identifies its worst-case traffic. Section 5 presents implementation results comparing the behavior and the complexity of all mechanisms. The conclusions are presented in Section 6.

2. IMPLEMENTATION ARCHITECTURE

This section presents a high performance development environment for communication systems with quality of service guarantees (Figure 1). It is designed to help developers implement and evaluate with efficiency, flexibility and very low cost, a wide variety of network systems, communication protocols and distributed applications. It consists of three basic modules used by all communication systems and some specific modules responsible for the specific characteristics of the

---

[†] This research is supported in part by grants from UFRJ, CNPq, PROTEM, CAPES and FUJB.

* Corresponding author

implemented communication system. The basic modules are the Memory Manager (MM), the Task Scheduler (TS) and the Timer Manager (TM). The specific modules are the System Manager Module, the High Performance Communication Protocol, the Synchronization Module and the Quality of Service Module.

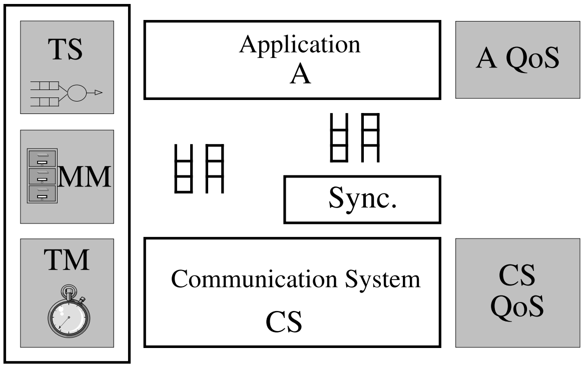


Figure 1: Implementation Architecture.

The memory manager module, in order to avoid excess of fragmentation, divides the memory into transmission and reception regions. This model allows a more orderly memory utilization resulting on faster and more efficient management. These regions are then subdivided in four parts. Two of them are reserved for fixed size data structures used by standard protocol layers interfaces. One of the regions is reserved for variable size structures used by the primitive parameters exchanged between adjacent protocol layers. The fourth region is dynamically allocated according to each established flow and contains fixed size data structures called blocks, used by the data that is effectively transmitted. Different from other general managers, this manager stores the allocation control information separated from the data, allowing an efficient implementation, without data copies, of several functions such as segmenting and reassembling, concatenation and separation, partial release of data, etc.

The task scheduler module inserts tasks in queues and determines the execution sequence of them. Besides, it has functions to provide priority levels for real-time flows. The behavior of the task queue is modeled by two queues where the low priority queue is served only if the high priority queue is empty. In order to distinguish the service of functions of very high priority such as timer time-outs, these functions are inserted on the first position of the task queue. This is also the output position of all tasks. A second priority level is given to tasks of real-time flows. Other tasks of lower priority are inserted at the end of the task queue. The two-queue model with priority levels was implemented by one single queue with three input references and only one output reference. In this manner, logical tests are avoided previously to the beginning of the execution of a new task to verify if the highest priority queue is empty or not.

The timer manager module proved to be very efficient. It is based on the Varghese & Lauck proposal [15], where an analogy with a real-time clock is simulated by a cyclic counter. This implementation privileges the performance, avoiding unnecessary logical checks at each hardware interruption, allowing the creation of a large number of timers “on the fly”. It provides functions to create one-way or auto-firing timers, restart, interrupt, and obviously flag the time-out of timers. In this implementation architecture, a time-out is considered as a service primitive, the time-out indication event is put into the appropriate interface queue, and the corresponding task - time-out processing function - into the maximum priority queue of the task scheduler.

The objective of the specific modules is to provide a set of functionalities for the system implementation. For instance, the synchronization module organizes the temporal order of the multimedia information (voice, video, graphs, data, etc.). It is based on Temporal Petri Nets [9,13] and implements inter and intra-media ways of synchronization. A more detailed description of this implementation architecture can be found in [7,8,11,12].

3. POLICING MECHANISMS

This section presents proposals for implementations of several policing mechanisms. The execution procedure is described through commented block diagrams, based on a presented high performance implementation architecture.

Leaky Bucket. One of the most mentioned policing mechanism in literature is known as leaky bucket (LB). According to this mechanism, each cell is able to be transmitted if it can insert a credit in a credit store called leaky bucket. The credit store has a limited size Cmax. If the credit store is full, the submitted cell is considered a violator, being discarded or tagged. The credits are removed from the store or “leaks from the bucket” at the negotiated mean transmission rate. The store capacity can be considered as the maximum burst length (regardless of the credits removed during the burst) since only Cmax cells can be transmitted during an interval less than the credits' decrement interval.

Leaky bucket is implemented by a token counter and an auto-firing timer. At the end of the connection setup, its parameters (Cmax, mean) are determined and the auto-firing timer is created. This timer is responsible for incrementing the tokens every (1/mean) time units, where mean is the contracted mean bit rate. The token credit is incremented up to its maximum limit Cmax. For each cell submitted to the mechanism, if the token counter C is positive, it is decremented and the cell is transmitted.

Other way of implementing this mechanism, proposed by Dittmann et al [6], is based on time stamps computed in order to update the token counter through a subtraction performed at the arrival of each cell. The value to be subtracted is calculated according to the elapsed time since the arrival of the previous cell. This scheme has the disadvantages of requiring time markers of extremely high precision in addition to executing inefficient non-unitary subtraction operations at each cell arrival.

Jumping Window. The mechanism jumping window (JW) considers windows with a fixed length T side by side through the time. According to this scheme a window starts immediately after the conclusion of the previous window. During a window, only Cmax cells can be submitted to the network. In the case of the Cmax cells have already been transmitted during the current window, the following cells will be considered violators and will be therefore discarded or tagged.

Jumping window is implemented with a single auto-firing timer and a token counter, according to the following procedure. At the end of the connection setup, the mechanism starts the auto-firing timer responsible for the token counter updating at every T time units. Concurrently, for each cell submitted to the jumping window, if the token credit is positive, it is decremented and the cell is transmitted. If there are no available tokens the cell is considered a violator.

Continuous Sliding Window. The continuous sliding window (CSW) mechanism also limits the maximum number of cells Cmax submitted to the network during a window of length T. The difference from jumping window mechanisms consists on how the windows are dynamically defined. The start of a window is synchronized with the arrival of each cell and its conclusion occurs exactly T time units later. In this manner, several superposed windows are opened and closed through the time and all of them restrain the traffic source to submit the maximum of Cmax cells during any time interval of length T. This mechanism has the advantage of storing the traffic characteristics during every window. However, its implementation complexity is relatively high due to the need of storing the arrival time stamps of all cells during each window. Besides computing the number of arrivals, limited to Cmax, this mechanism has to activate a timer of length T for each cell that is successfully submitted to the network. This feature can be extremely expensive specially for long length windows.

Continuous sliding window is implemented by several timers, one for each cell, and one token counter. For each cell submitted to the mechanism, if the credit is positive it is decremented and a timer with duration T is created. The token credit is incremented at each timer time-out. The demand for a large and variable number of timers implies on larger memory requirement and higher CPU utilization. Besides, there is the disadvantage of the inefficient resource allocation for each connection.

Discrete Sliding Window. The discrete sliding window mechanism (DSW) is similar to the continuous mechanism with a larger granularity. The basic difference consists on the fact that the arrival time is now measured according to time intervals, with length equal to the time spent in the transmission of a cell. In this way, the arrival time stamps are considered through the discrete time. One of the advantages of this scheme the requirement of one single timer for its implementation. The storage of the arrival time stamps is performed by associating one bit to each position of the window where a cell can arrive. Again, like the previous continuous version, this mechanism can be relatively expensive for long duration windows.

The discrete sliding window mechanism is implemented by a token counter, a single timer, in addition to requiring a shift bit array. Each bit of the array is responsible for registering the arrival of a cell on its corresponding time unit. Each time unit has a length that restrain the arrival of only one cell per time unit, for example, for transmission rates of 155 Mb/s, the time unit tcel is 2.7 µs. This scheme, compared to the continuous one, allows a better utilization of memory and CPU.

4. SLOTTED MOVING WINDOW MECHANISM

The Slotted Moving Window (SMW) mechanism proposed in this section seeks a trade-off between jumping window simplicity and sliding window efficiency regarding traffic memorization, accurate policing of mean rate and restriction to burst length.

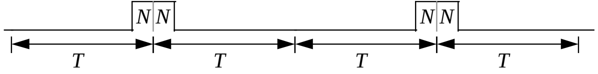
One of the main factors that leads sliding window to require a high implementation complexity is the storage of cell's arrival times. On the other hand, the juxtaposed windows of jumping window mechanism can be simply implemented with an auto-firing timer.

In this mechanism the window of duration T is subdivided in S periods of equal time named slots. Similarly to JW and SW, at each cell arrival a counter, if positive, is decremented. The counter is incremented by the number of cells that arrived during a slot Si, T time units after the beginning of this slot Si. Hence, the Slotted Moving Window is based on the idea of "superposed windows that jump, from slot to slot" instead of sliding.

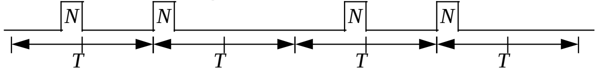
It can be observed that for the number of slots S equal to one this mechanism is equivalent to the jumping window. When S tends towards infinite, the duration of each slot tends towards zero and this mechanism approximates to the sliding window mechanism.

The worst case traffic is generated by bursts of length N equal to Cmax cells. The bursts are synchronized in an alternate way with the slots. The end of a burst synchronizes to the end of a slot, while the next burst starts at the instant of the beginning of the slot that it occupies. This pattern is repeated periodically. Figure 2 shows a comparison among the approached mechanisms. When the burst duration is shorter than slot duration, the minimum silence period between bursts has a duration of S-1 slots. In this situation, as each slot lasts T/S time units, the duration of the minimum silence period can be expressed as: $\frac{S-1\times T}{S}$.

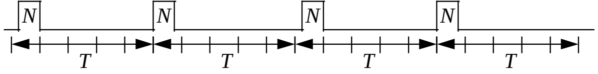
As well as sliding window, this mechanism limits the burst duration to N cells while the maximum burst duration of jumping window is 2N cells. However, the mechanism does not succeed to maintain sliding window characteristic where in every time interval equal to T, only N cells can be submitted to the network. As in the jumping window the source can transfer up to 2N cells in a T time period but they are spaced by a period of time that is at least equal to one slot, in the worst case this silence period is T/2 for two slots.



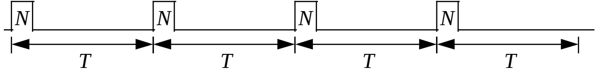
(a) jumping window



(b) Slotted Moving Window with 2 slots



(c) Slotted Moving Window with 5 slots



(d) sliding window

Figure 2: Comparison of worst case traffic for SMW.

For an increase in the number of slots, the minimum interval between bursts also increases. However, when S tends towards infinite the slot duration goes to zero and each burst cell arrives in a different slot. In this situation, the worst case traffic pattern of Slotted Moving Window tends towards worst case traffic pattern of sliding window.

Implementation. Slotted Moving Window is implemented by a single auto-firing timer, a token counter and a FIFO array dimensioned according to the traffic and the mechanism parameters. Differently from other window-based mechanisms this mechanism defines the parameter S, in addition to the parameters T and Cmax. The parameter S is the number of time slots contained in a window of length T. This parameter is extremely important in determining the memory and CPU requirements. The greater the number of time slots S, the larger is the FIFO array, the higher is the CPU utilization and the closer is this mechanism to sliding window . Conversely, the lower the number of time slots, the closer is this mechanism to the jumping window.

When the connection establishment phase is completed, the array FIFO is allocated with size S x nb, where nb is the number of bits necessary to represent the maximum number of cells that can be transmitted during a slot. Analytically the amount of bits necessary to represent the array FIFO is expressed by:

$$M = S \times nb;$$

where: $nb = \left\lfloor \log_2 \left\lceil T \big/ \left(S \times t_{cel} \right) \right\rceil \right\rfloor + 1 \cdot$

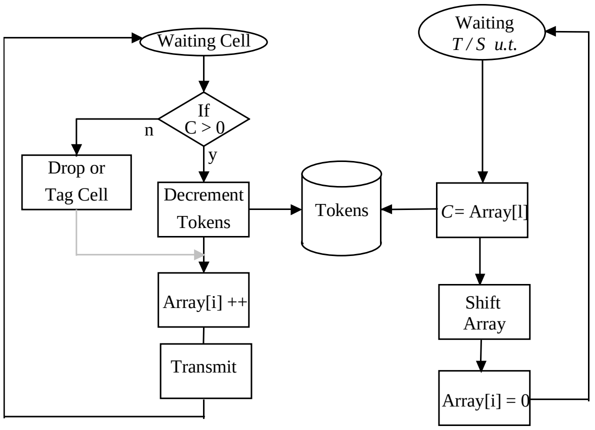


Figure 3: Slotted Moving Window implementation.

The auto-firing timer is responsible for updating the token counter at each T/S time units. This updating corresponds to increment the token by the number of cells that arrived during the time slot Si started T time units before this operation. This value is stored on the last position of the FIFO array.

5. RESULTS

This section presents a quantitative and qualitative analysis of the presented mechanisms. Section 5.1 evaluates the implementation results of the behavior of SMW. Section 5.2 concentrates on the implementation complexity of all presented mechanisms.

5.1. Evaluating Slotted Moving Window

This section evaluates the behavior of the proposed SMW mechanism. It presents quantitative and qualitative results obtained through implementation. The comparison concentrates on the violation probability and the maximum burst duration.

In order to compute violation probability (p_{viol}) of the mechanisms, two traffic patterns are used: a geometric and a bursty traffic. In both cases the mean bit rate is equal to 1/5 cells/time unit. The burst/silence traffic has a burst length (N) given by a geometric distribution with mean 10 time units and a burst bit rate of 1 cell per time unit. The silence period was given by a geometric distribution with a mean of 40 time units. Therefore, this traffic has a burstiness factor (maximum cell rate divided by mean cell rate) of 5.

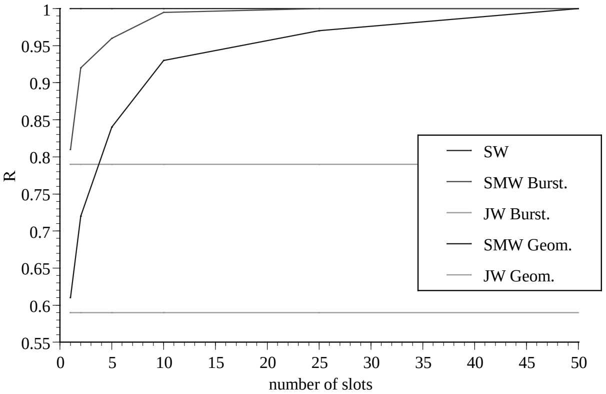


Figure 4: Influence of the number of slots on the behavior of SMW for geometric and bursty/silence traffics.

Figure 4 shows the influence of the configured number of slots on the violation probability behavior of SMW when submitted to geometric and burst/silence sources of traffic. The vertical axis shows the ratio *R* that is defined as the ratio of SMW violation probability to SW violation probability. The lower bound of the graphics is the ratio of JW violation probability (SMW with $S = 1$) to SW violation probability, while the upper bound is equal to one, representing the ratio *R* of SW violation probability to itself. It can be observed that, for a number of slots *S* equal to one, the violation probability of the slotted moving window mechanism is the same of the jumping window mechanism and as long as number of slots increases, the violation probability tends quickly towards violation probability of sliding window mechanism ($R = 1$). In the case of burst/silence traffic the p_{viol} tends more quickly towards its final value than in geometric traffic case.

5.2. Implementation Complexity

This section presents comparative data focusing on the implementation complexity of the considered mechanisms. The required amount of memory, the number of timers and their precision, the mean total processing time of each cell and the mean mechanism processing time of each cell are analyzed.

Table 1 contains comparative information about the mechanisms. The notation $\{x\}$ represents the number of bits necessary to represent the value *x*. The processing time of each mechanism was measured in SUN SPARC 2 workstations. All measurements were obtained through a mean of 1000 samples, all of them were achieved in the very same conditions.

Regarding the obtained results, it can be observed that Leaky Bucket, Jumping Window and Slotted Moving Window are the most efficient, with processing times of magnitude of 140 µs. The Continuous Sliding Window is one of the most expensive mechanisms in relation to processing time - 167 µs - because of the creation of a timer for each cell arrival. Leaky Bucket with output smoothing presented a high processing cost because of the updating of the output queue. Slotted Moving Window presented a mean processing time relatively independent of the number of slots used and of a constant value in between JW and CSW.

Table 1: Comparison of the mechanisms.

Mechanism	LB	JW	CSW	DSW	SMW
Proc. Time	143 µs	141 µs	167 µs	150 µs	150 µs
Timers	1	1	*N*	1	1
Timer Precision	Variable	low	high	high	flexible
Timers Memory	{Timer}	{Timer}	*N* x {Timer}	{Timer}	{Timer}
Counters and Parameters Memory	$\{N\}$ + $\{N_{max}\}$	$\{N\}$ + $\{N_{max}\}$	$\{N\}$ + $\{N_{max}\}$	$\{N\}$ + $\{N_{max}\}$	$\{N\}$ + $\{N_{max}\}$ + $\{S\}$

An important factor to be considered is the number of timers and their precision. It can be observed that the continuous sliding window mechanism needs the greatest number of timers - *N* timers. It may be emphasized that the greater the number of timers, the greater the CPU utilization. Besides, these timers demand an extremely high precision, because they must be fired exactly at the arrival of each cell. Discrete Sliding Window is a little more feasible, however it still needs a very high precision, for example, 2.7 µs for a transmission rate of 155Mb/s. Leaky Bucket needs timers of the inverse of the mean policed transmission rate and of the inverse of the peak transmission rate magnitude. For transmission rates of for example 100 cells/second, the timer precision is 10 ms. Jumping Window uses a timer of *T* time units of magnitude. Slotted Moving Window allows the use of a timer adjustable in all precision range, from the one used in JW to the one used in CSW, according to the number of slots used.

Another very important issue in implementation complexity analysis is the amount of memory necessary for each mechanism. The required memory can be divided in the memory used by the timers, the memory used by the counters and parameters of the mechanisms and the auxiliary control memory. This last one is necessary only for the sliding window and the slotted moving window mechanisms that respectively require:

$$\left\lceil \frac{T}{t_{cel}} \right\rceil \text{ and } S\times\left\lceil T/\left(S\times t_{cel}\right)\right\rceil \text{ bits}$$

for traffic memorization. It can be noticed that continuous sliding window needs *N* x {Timer} bits for timers, while slotted moving window requires {S} bits to store the number of slots.

The two variants of sliding window are comparable in relation to the necessary amount of memory. Jumping window is the one that requires the smallest amount of memory. Slotted moving window requires a variable amount of memory according to the number of slots used, yet inferior to the amount required by sliding window. For example, the case of discrete moving window with $T = 60 \text{ x } t_{cel}$ and $N = 30$. In this case it requires 60 bits to store the traffic of a window interval. If slotted moving window is used with 20 slots then at most 3 cells can arrive during each slot. These cells are represented by 2 bits. Hence, the total number of bits to represent the array FIFO is 40 bits. For a SMW with 10 slots, 30 bits are required and with 5 slots, 20 bits. It can be observed from Figure 3 that with 5 slots, the behavior of slotted moving window is very close to sliding window performance and a memory savings of 66% is obtained.

6. CONCLUSIONS

This paper proposed, implemented and analyzed the Slotted Moving Window mechanism. Furthermore, several traffic policing mechanisms were described, implemented and compared.

The implementation complexity of the mechanisms was analyzed, focusing on the required amount of memory, the number of timers and their precision and the mean processing time. The obtained results demonstrate the highest complexity of sliding window mechanism and the implementation simplicity of leaky bucket and jumping window mechanisms. Slotted moving window presents the advantage of being flexible. The functionality and the complexity of this mechanism ranges from those of jumping window (proposed mechanism with $S = 1$) to those of sliding window, keeping its accurate control of the mean transmission rate. Furthermore, leaky bucket, jumping window and slotted moving window mechanisms have the advantage of permitting an implementation with just one auto-firing timer. This feature requires a lower complexity and a lower precision in relation to the timer management when compared to the triggered jumping window and continuous sliding window mechanisms.

Among important advantages of the mechanism it is possible to mention the traffic memorization, the accurate policing of mean bit rate and the restriction of the maximum burst length to N cells, in addition to the low complexity of implementation and flexibility in the precision of the timer used. The worst case traffic of this mechanism is characterized by the non-continuous transfer of up to $2N$ cells to the network during a T time period, but they are spaced by a silence period of at least equal to one slot duration, that in the worst case is $T/2$ for two slots.

Results obtained for geometric and burst/silence traffic patterns confirm that even with a small number of slots ($S = 5$) a functionality very close to sliding window performance can be obtained, with a low implementation complexity and with considerably saving of memory and CPU processing. Considering the example mentioned in Section 5, with only one timer of $T/5$ precision, SMW limits the burst length to 5 cells obtaining a memory saving of 66% in relation to discrete sliding window and regarding violation probability, the results for burst/silence traffic present a relative difference of just 4%.

REFERENCES

[1] ITU-T Recommendation I.311, B-ISDN General Networks Aspects, Genebra, 1991.

[2] E. Rathgeb, "Modeling and Performance Comparision of Policing Mechanisms for ATM Networks", IEEE JSAC, pp.325-334, vol. 9, no. 3, Apr. 1991.

[3] M. Butto, E. Cavallero, A. Tonietti, "Effectiveness of the Leaky Bucket Policing Mechanism in ATM Networks", IEEE JSAC, pp. 335-342, vol. 9, no. 3, Apr. 1991.

[4] A. Berger,"Performance Analysis of a Rate Control Throttle Where Tokens and Jobs Queue", IEEE INFOCOM, pp. 30-38, 1990.

[5] J. Junior, J. Monteiro, "On The Efficiency of Policing Mechanisms for ATM Networks", IEEE ITS, pp.470-474, Aug. 1994.

[6] L. Dittman, S. Jacobsen, K. Moth, "Flow Enforcement Algorithms for ATM Networks", IEEE JSAC, pp. 343-349, vol 9. no. 3, Apr. 1991.

[7] C. Albuquerque, M. D. Nunes, O. C. Duarte, "Performance Measurements in a Manufacturing Communication System", IEEE ISCAS, pp.3.69-3.72, London, England, May. 1994.

[8] C. Albuquerque, M. Nunes and O. Duarte, ‘‘An Efficient Implementation Architecture for Layered Communication Systems”, 38th IEEE MWSCAS, Brazil, August 1995.

[9] M. Woo, N.Qazi, A.Ghafoor, "A synchronization framework for communication of pre-orchestrated multimedia information", IEEE Network, pp.52-61, Jan. 1994.

[10] G. Varghese, T. Lauck, “Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility”, ACM SIGOPS, pp. 25-38, Texas, 1987.

[11] M. Faerman, C. Albuquerque and O. Duarte, ``Janela Móvel Particionada: Proposta de um Novo Mecanismo de Policiamento de Tráfego", SBT, Brazil, Sept. 1995.

[12] C. Albuquerque, M. Faerman and O. C. Duarte, “Implementação de Mecanismos de Policiamento em Redes de Alta Velocidade", SBRC, pp. 103-119, Brazil, May 1995.

[13] R. De Lima and O. Duarte, "Implementação de um Serviço de Sincronização Multimídia para Comunicação de Dados Pré-orquestrados", SBRC, pp. 19-35, Brazil, May 1997.