Implementation of Policing Mechanisms for High Speed Networks

Célio Vinicius Neves de Albuquerque[1] *Marcio Faerman*[1]

Otto Carlos Muniz Bandeira Duarte

COPPE / EE - Programa de Engenharia Elétrica
Universidade Federal do Rio de Janeiro
CP 68504 - CEP 21945-970 Rio de Janeiro, RJ, Brasil
E-mail: {celio, faerman, otto}@coe.ufrj.br

Abstract

This paper proposes implementations of the well-known traffic policing mechanisms for high speed networks. The implementations are based on a high performance development environment and are described through commented block diagrams. The considered mechanisms are Leaky Bucket with variants, Jumping Window, Triggered Jumping Window, Continuous Sliding Window, Discretized Sliding Window and Time Slotted Moving Window. These mechanisms were implemented in UNIX and DOS environments. A comparison of the results focusing on the implementation complexity is presented considering required memory, number of timers and its precision, mean processing time of the communication system and mean mechanism processing time. The results confirm the low implementation complexity of Leaky Bucket, Jumping Window and Time Slotted Moving Window mechanisms.

1 Introduction

Integrated services, composed by several media, such as voice, asynchronous data and video, have been strongly required within the last few years. The transfer mode adopted for the promising Broadband Integrated Services Digital Network (B-ISDN) must have the potential to fulfill the demands of flexibility, efficiency and robustness. Asynchronous Transfer Mode (ATM) has turned out to fulfill these requirements.

In order to increase the efficiency of B-ISDN, traffic sources must be statistically multiplexed, resulting on a resource allocating strategy that would reserve fewer buffers than required by the maximum transmission rate parameter. The complex challenge of preserving and controlling the traffic characteristics throughout the network is therefore issued. This traffic control must avoid congestion situations that might cause an increase on cell losses, undesired delays and jitters.

B-ISDN should accept efficiently the greatest possible number of simultaneous users and assure the transport of the contracted traffic. The network should control and provide guarantees of meeting the Quality of Service (QoS) parameters, such as cell loss rate, delay and jitter defined by the application.

---

[1] Graduate Students of the Department of Electrical Engineering of COPPE/UFRJ.

One of the most complex challenges and open issues to be solved by B-ISDN is its traffic control strategy. In order to assure the performance of B-ISDN, several traffic control mechanisms are required. Some of them have been already considered [1], such as connection admission control, priority control, congestion control and usage/network parameter control.

This work concerns with usage/network parameter control that is the function, performed by the network during the information transfer phase, of supervising and controlling the user traffic. The main goal of this control is to protect the network resources against, intentional or not, unwell behaved traffics. This traffics are the ones that violate the QoS parameters negotiated during the connection setup, being potentially harmful to the QoS parameters of other connections. This kind of control has the function of detecting possible violations of the contracted parameters. This control mechanism has been given a lot of names, such as traffic enforcement, flow enforcement and will be further referred in this paper as policing mechanism.

In addition to supervising and controlling the traffic sources efficiently, the policing mechanism must be feasible, cost effective and flexible, requiring a low implementation complexity. Several mechanisms have been proposed. Among them there are some outstanding mechanisms, such as the leaky bucket [2,3] and its variations [4] and the window-based mechanisms [2,5,6].

This paper proposes implementations for several policing mechanisms. The mechanisms are leaky bucket and its variation with output traffic smoothing, jumping window, triggered jumping window, continuous sliding window, discretized sliding window and time-slotted moving window [17]. The implementations are based on a high performance development environment [7,8,9] and are described by commented block diagrams. The mechanisms were implemented using C language on UNIX and DOS environments. A comparison of the results focusing on the implementation complexity is presented considering required memory, number of timers and its precision, mean processing time of the communication system and mean mechanism processing time.

In section 2 the policing mechanisms considered in this work are described. Section 3 presents the implementation architecture and its modules. Section 4 proposes implementations for the mechanisms. Section 5 presents the numerical results obtained through the implementation and compares the mechanisms. Finally the conclusions are presented in section 6.

2 Policing Mechanisms

Assuming that a connection is successfully admitted, the network protects its resources against abusive sources of traffic, using a traffic control mechanism called policing. This control assures that the traffic will satisfy its negotiated parameters, such as mean and maximum transmission rates, burstiness factor, maximum burst length, etc. Besides, service classes that are not delay sensitive may have its traffic shaped or smoothed in order to control the transmission rate.

Cells that compose the component of the traffic considered by the policing mechanism as violator are discarded or tagged. In this case, the tagged cells are transmitted throughout the network, unless they cause a situation of congestion, with possible additional charge for the user.

Among the policing mechanisms proposed in literature, the outlined ones are leaky bucket, jumping window and sliding window.

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 (figure 1). The credit store has a limited size C_{max}. If the credit store is full, the submitted cell is considered violator, being discarded or tagged. The credits are removed from the store or leaky from the bucket at the negotiated mean transmission rate. The store capacity can be considered as the maximum burst length (regardless the credits removed during the burst) since only C_{max} cells can be transmitted during an interval less than the credits' decrement interval.

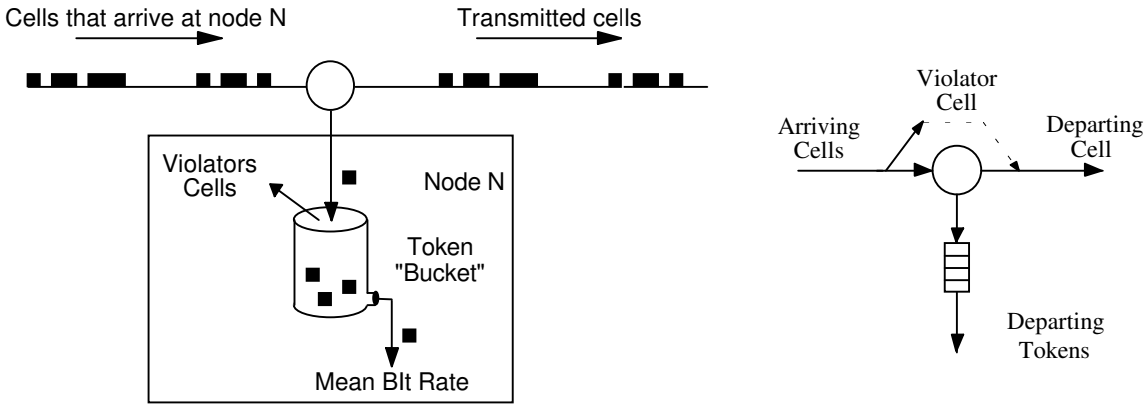


Figure 1: Representations of leaky bucket mechanism.

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 C_{max} cells can be submitted to the network. In the case of the C_{max} cells have already been transmitted during the current window, the following cells will be considered violator and will be therefore discarded or tagged (figure 2).

Triggered Jumping Window

The triggered jumping window mechanism (TJW) is similar to the jumping window mechanism, however, the start of each window is not automatic, is synchronized with the arrival of the first cell submitted after the conclusion of the previous window. This procedure

avoids large periods of silence during a window resulting in more accuracy on the traffic control.

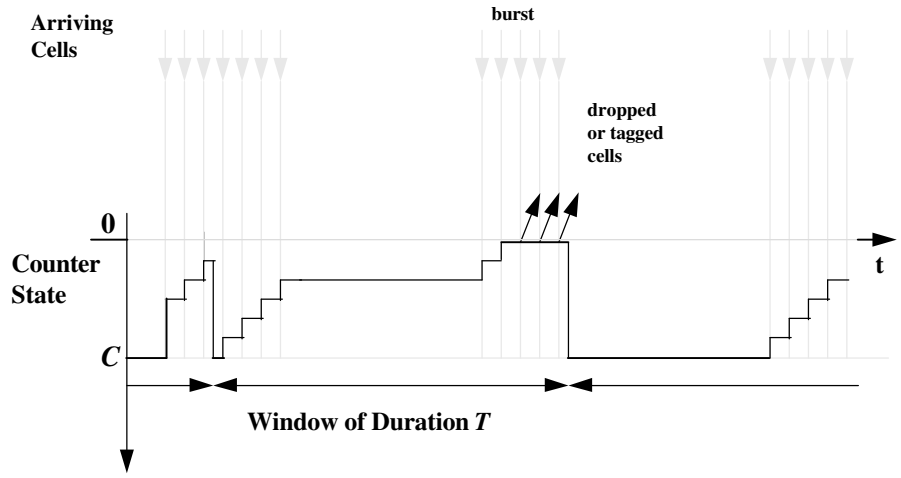


Figure 2: The jumping window mechanism.

Continuous Sliding Window

The continuous sliding window (CSW) mechanism also limits the maximum number of cells C_{max} 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 after. 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 C 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 C, 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. Figure 3 outlines the behavior of this mechanism.

Discretized Sliding Window

The discretized 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 is related to the number of timers required to its implementation. This feature will be further exploited in section 4. 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.

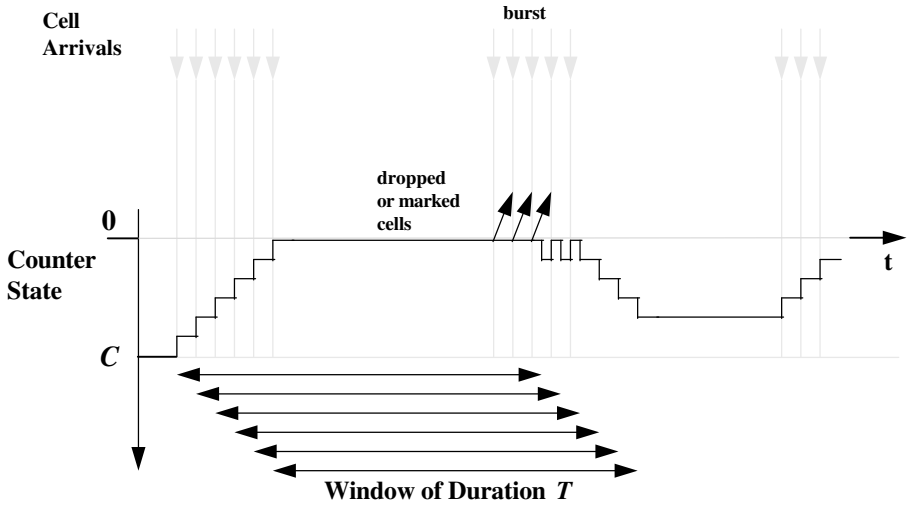


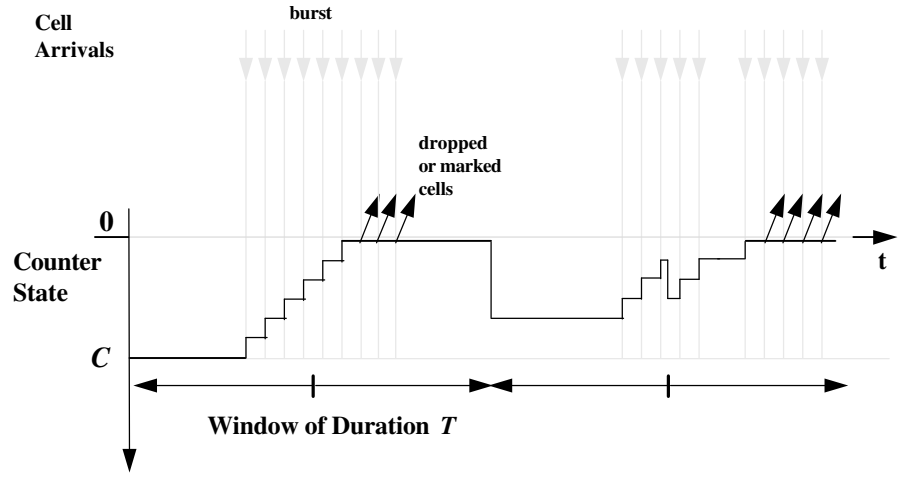
Figure 3: The continuous sliding window mechanism.

Time Slotted Moving Window

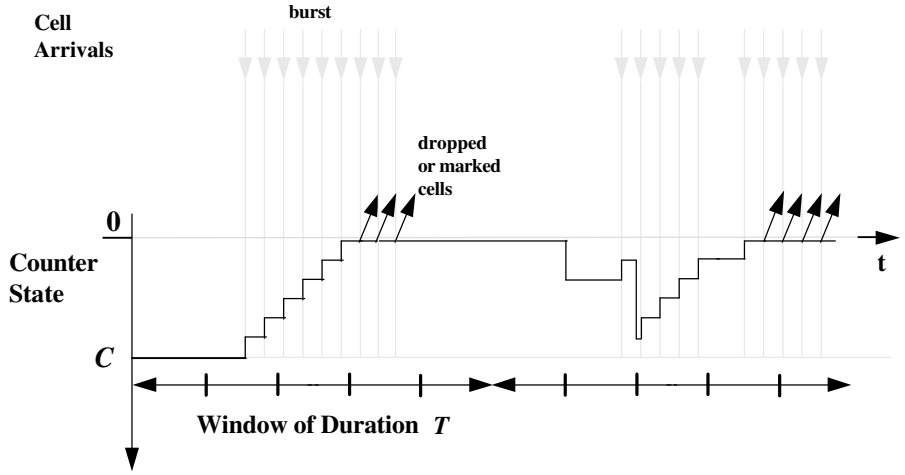
The mechanism time slotted moving window (TSMW) [17] considers that each window with length T is divided into P smaller parts called time slots. Similar to other window-based mechanisms, for each cell arrival, if the credit counter is positive it is decremented. The counter is incremented with the number of cells arrived during the time slot p_i, exactly T time units after the beginning of this time slot (figure 4).

It can be observed that with a number of time slots P equal to one, this mechanism is equivalent to the jumping window mechanism. When P rises tending towards infinite, the length of each time slot tends towards zero and this mechanism comes close to the sliding windows mechanisms.

The TSMW searches for a trade-off between the simplicity of jumping window mechanisms and the efficiency of sliding window ones, considering the features of traffic memorization, accurate policing of the mean transmission rate and the maximum burst length restrain.



(a)



(b)

Figure 4: The time slotted moving window mechanism with a) 2 time-slots and b) 5 time-slots.

Maximum and Mean Transmission Rates Control and Smoothing

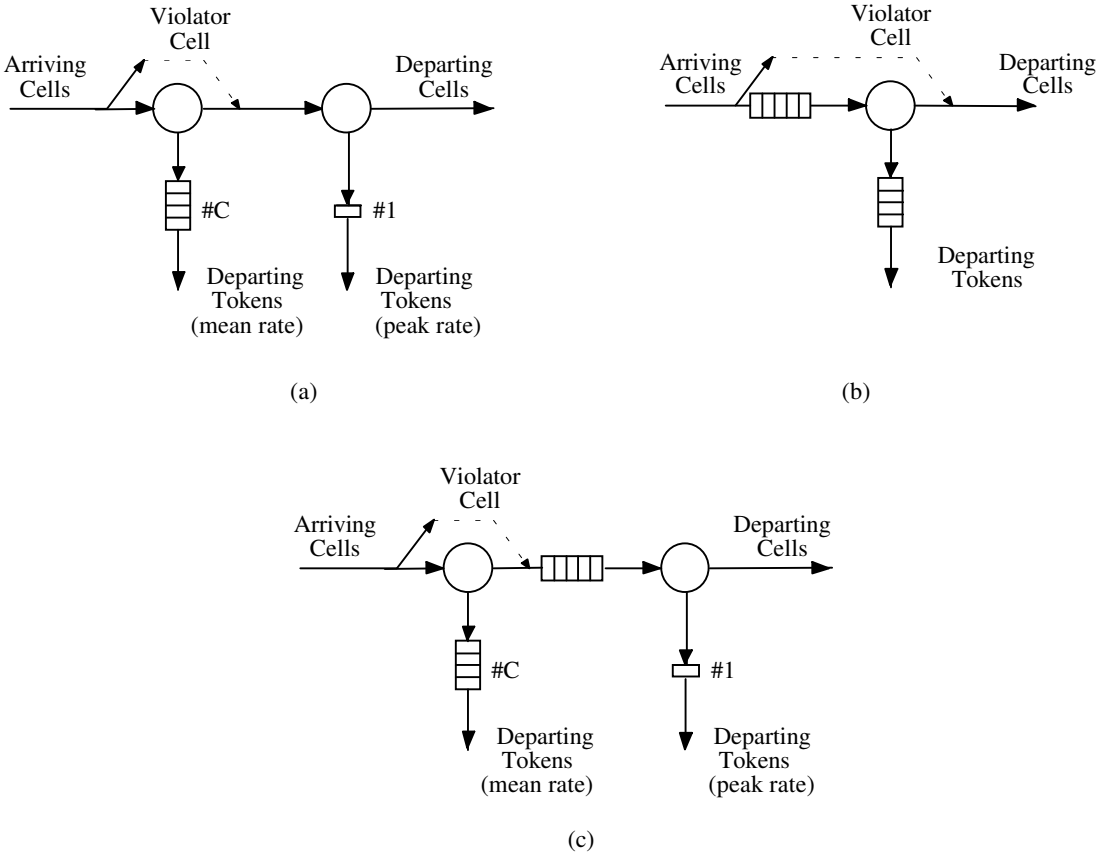


Figure 5: Leaky bucket control traffic variations.

In general, the policing mechanisms control only one transmission rate. The control of both the mean and the maximum transmission rates simultaneously requires the interconnection of two policing mechanism modules. It can be obtained through the cascade of a conventional

mechanism controlling the mean rate with another mechanism functioning as a spacer of cells according to the maximum rate. Figure 5(a) shows this scheme for the leaky bucket mechanism. In this case, the spacer consists of a leaky bucket with a credit store limited to $C_{max} = 1$ and with credit up-dating rate equal to the maximum transmission rate.

Other type of traffic control is called traffic shaping. In this kind of control, the traffic is "shaped" in order to respect and obey its negotiated parameters. When the controlled parameter is the transmission rate, the control is called traffic smoothing. The smoothing is performed through the insertion of a buffer responsible for the storage of the exceeding, violator cells, instead of discarding them. Although these cells are not discarded, the storage of them may introduce undesirable delays and jitters. Figure 5(b) presents the leaky bucket version of input traffic smoothing and figure 5(b) of output traffic smoothing. The violator cells are stored until they are allowed to be submitted without being policed.

3 Implementation Architecture

The implementations of the policing mechanisms were based on a high performance development environment for communication systems with quality of service guarantees (figure 6). The implementation architecture consists of three basic modules used by all developed communication system and some specific modules that are responsible for the definition of the characteristics of the implemented communication system. The basic modules are the Memory Manager Module (MM), the task Scheduler Module (TS) and the Timer Manager Module (TM). The specific modules are the System Manager Module, the High Performance Communication Protocol, the Synchronization Module and the Quality of Service Module that includes the Control Traffic and Policing Module.

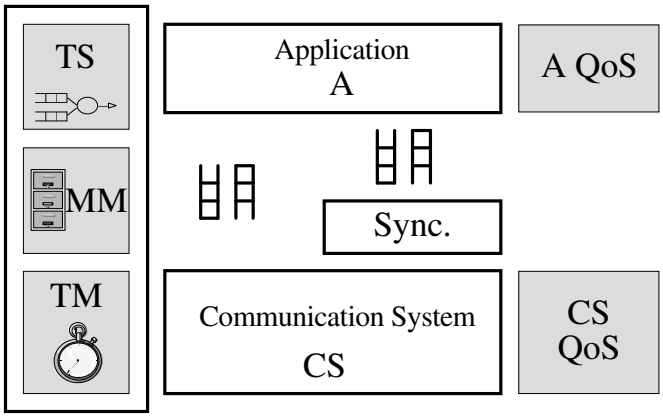


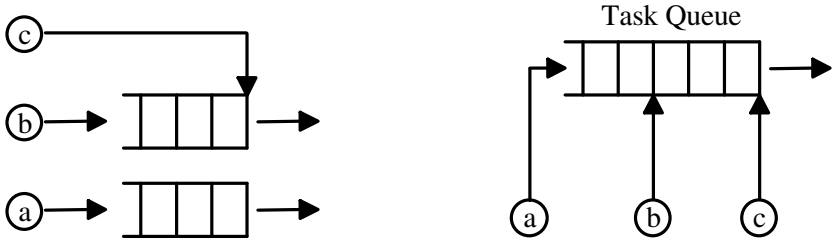
Figure 6: The implementation architecture.

The objective of the specific modules is to provide a set of functionalities for the system implementation. The system manager module configures the others according to the characteristics of the desired system. The communication protocol [10] offers a set of existing functions available in the main high performance protocols. These protocol facilities are configurable either through service primitives or through the system manager module. The synchronization module organizes the temporal order of the multimedia information (voice, video, graphs, data, etc.). Based on Temporal Petri Nets [12], this module implements the

inter and intra-media ways of synchronization. The quality of service module implements the mechanisms Virtual Clock [12], Stop-and-Go [13], Jitter-Earliest-Due-Date [14] and Hierarchical Round Robin [15]. This module also includes the implementation the policing mechanisms leaky bucket and its version with output traffic smoothing, jumping window, triggered jumping window, continuous sliding window and time-slotted moving window.

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 turning the management faster and therefore more efficient. It is possible because transmission and reception flows are independents. These regions are subdivided in four parts. Two of them are reserved for fixed size data structures used in the protocol layers interfaces and in the basic modules access points. One of the regions is reserved for variable size structures used by the primitive parameters exchanged between adjacent protocol layers. The forth 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 the efficient implementation, without copies, of several functions such as segmenting/reassembling, concatenation/separation, partial release of data, etc.

The task scheduler module inserts the tasks in queues and determines the execution sequence of them. Besides, it has functions to perform priority levels for real-time flows. The behavior of the task queue is modeled by two queues where the lower priority queue is served only if the higher priority queue is empty. In order to distinguish the service of functions of very high priority such as timers' time-outs, these functions are inserted on the first position of the task queue (figure 7 - label **c**). This is also the output position of all tasks. A second priority level is given to tasks of real-time flows inserted in the label **b**. Other tasks of lower priority are inserted in the end of the task queue (label **a**). 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, 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.



Model Implementation

a - entry point of lower priority tasks.
b - entry point of higher priority tasks.
c - entry point of maximum priority tasks
and output of all the tasks.

Figure 7: Task scheduler module.

The timer manager module proved to be very efficient. It is based on the Varghese & Lauck proposal [16], 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 interrupt, allowing the creation of a great number of run-time timers dynamically. It provides functions to create simple or auto-firing timers, restart, interrupt, and obviously sign 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.

4 Policing Mechanisms Implementation

This section presents actual proposals for implementations of the considered policing mechanisms. The execution procedure is described through commented block diagrams, based on the presented high performance implementation architecture. The implementation complexity is commented upon.

Leaky bucket

The leaky bucket mechanism is implemented by a token counter and an auto-firing timer. At the end of the connection setup, the mechanism parameters (C, λ_{mean}) are determined and the auto-firing timer is created. This timer is responsible for incrementing the tokens every ($1/\lambda_{mean}$) time units, where λ_{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. This procedure is illustrated in the flow graph of figure 8.

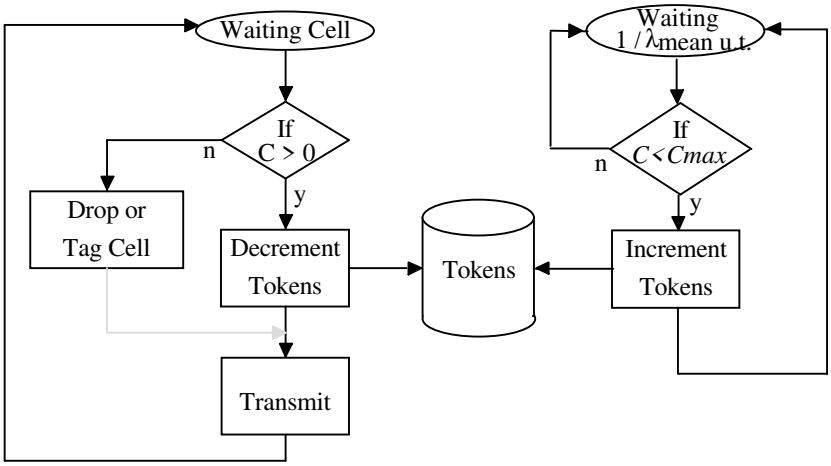


Figure 8: Block diagram for leaky bucket implementation.

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 is described by the following pseudocode, and has the disadvantages of requiring time markers of extremely high precision in addition to executing inefficient non-unitary subtraction operations at each cell arrival.

For each cell arrival do:

1. Δt := last_arrival_time - current_time;
2. last_arrival_time := current_time;
3. if $C < \Delta t \times \lambda_{mean}$
   then $C := 0$;
   else $C := C - \Delta t \times \lambda_{mean}$;
4. if $C > C_{max} - 1$
   then discard the cell;
   else $C := C + 1$;

Leaky bucket with Output Traffic Smoothing

The leaky bucket variant with maximum bit rate control or with output traffic smoothing is also implemented. This control mechanism is implemented by adding to the original mechanism another auto-firing timer responsible for the control of the maximum transmission rate, behaving as a cell spacer. A FIFO queue is also required to store the cells up to the moment they are eligible to be transmitted. In order to achieve more efficiency on logical tests, the implementation of the token counter obeys a logic inverse in relation to the described on section 2, thus considering that the condition of violation is detected when the counter is zero. In this scheme, shown in figure 9, for each submitted cell, if the counter is positive, it is decremented and the cell is sent to the output transmission queue. The auto-firing smoothing timer is responsible for removing the cells from the FIFO queue and to transmit them, at every $1/\lambda_{max}$ time units, where λ_{max} is the maximum output transmission rate.

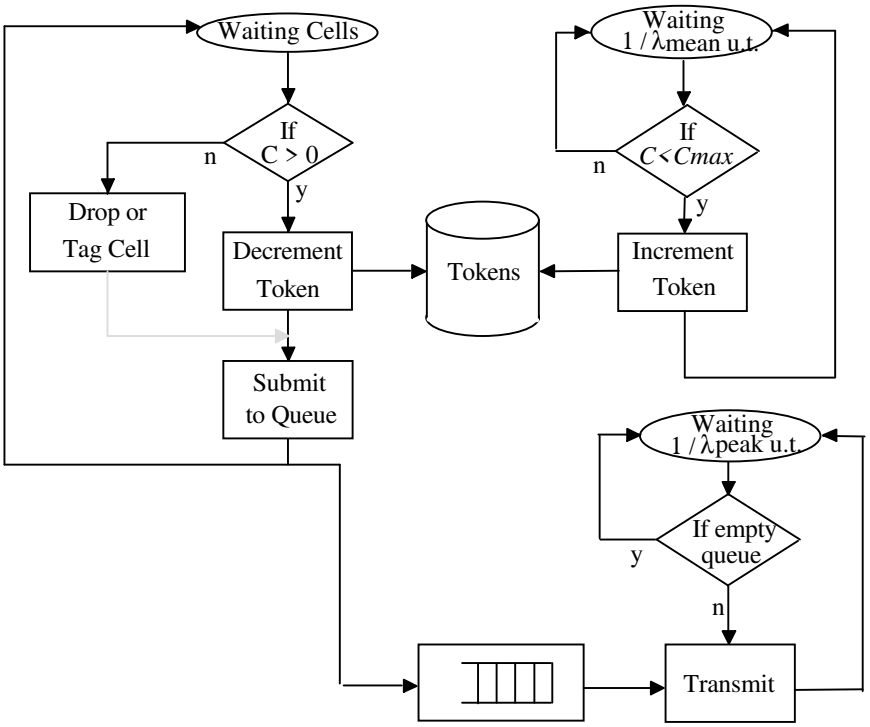


Figure 9: Block diagram for leaky bucket with output traffic smoothing.

Jumping Window

The jumping window mechanism 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 transited multiplexed with other connections. If there is no available tokens the cell is considered violator and in this case it was chosen to discard the cell.

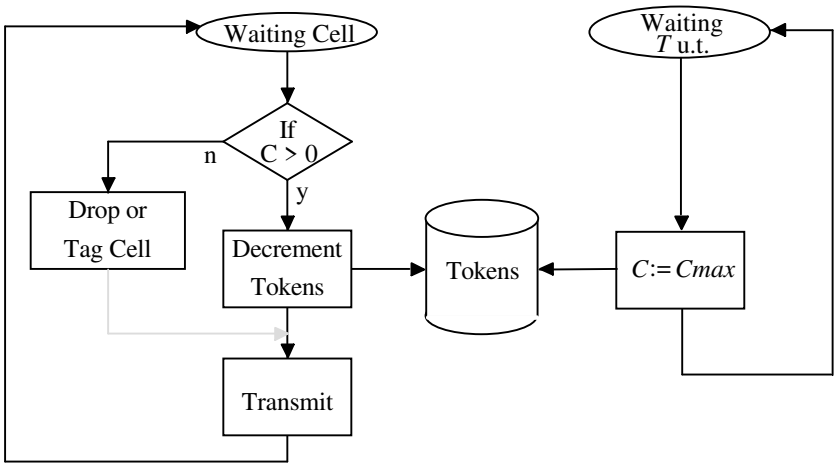


Figure 10: Block diagram for jumping window implementation.

Triggered Jumping Window

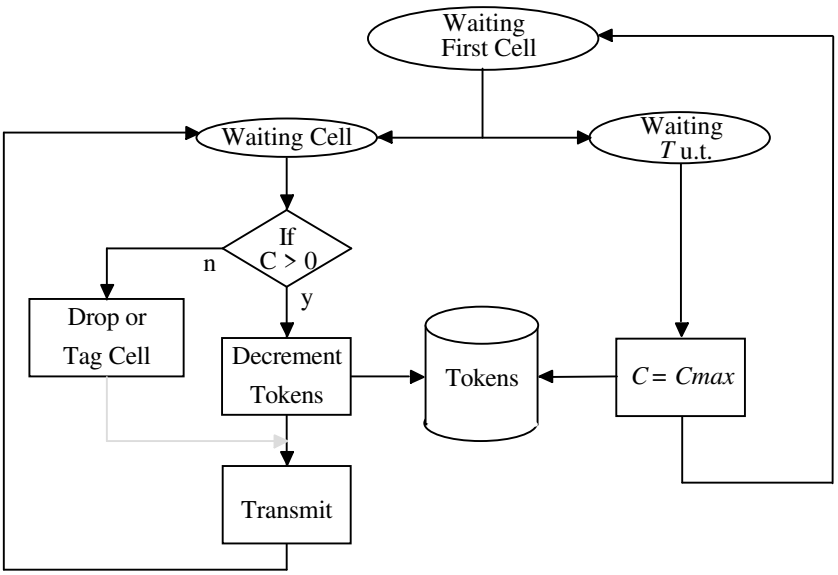


Figure 11: Block diagram for triggered jumping window implementation.

This mechanism is implemented in a similar way when compared to the original jumping window. However, the timer responsible for the token counter updating is now not auto-firing

and is initialized at the arrival of the first cell of each window and long exactly *T* time units. This feature is implemented by a little change on the mechanism state machine. After the connection setup and after each timer time-out, the triggered jumping window remains on the state "Wait for First Cell". At the moment of the first cell arrival, the timer if starts to clock, the token counter is decremented and the mechanism goes to the normal transmission state.

Continuous Sliding Window

The mechanism 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* time units is created. At each timer time-out, the token credit is incremented. The demand for a large and variable number of timers implies on a higher required memory and CPU utilization. Besides, there is the disadvantage of the inefficient resource allocation for each connection.

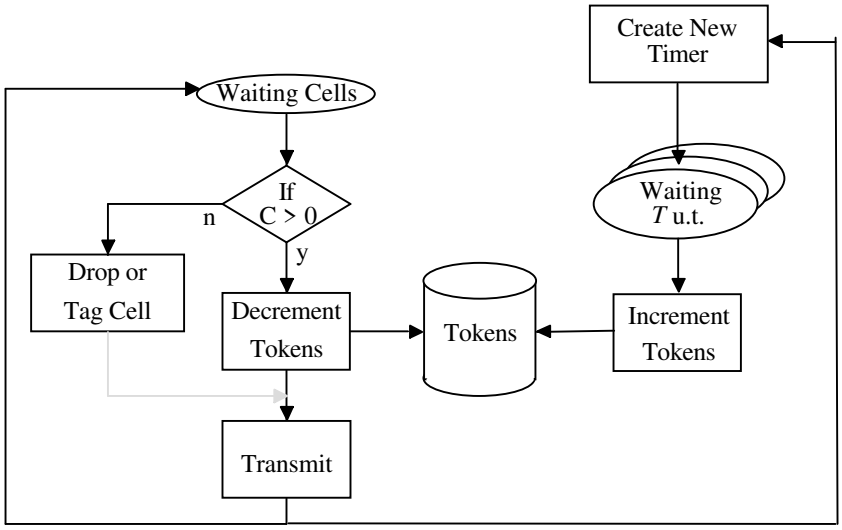


Figure 12: Block Diagram for the continuous sliding window implementation.

Discretized Sliding Window

The discretized 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 t_{cel} is 2,7 µs. This scheme, compared to the continuous one, allows a better utilization of memory and CPU.

Time Slotted Moving Window

The time slotted moving window is implemented by a single auto-firing timer, a token counter and a FIFO array dimensioned according to the traffic and mechanism parameters. Different from other window-based mechanisms, in addition to the parameters *T* e *C*, this mechanism defines the parameter *P*, that 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 *P*, the greater the FIFO array required memory, the greater the CPU utilization and the closer to sliding window is this mechanism. The lower the number of time slots, the closer to jumping window is this mechanism.

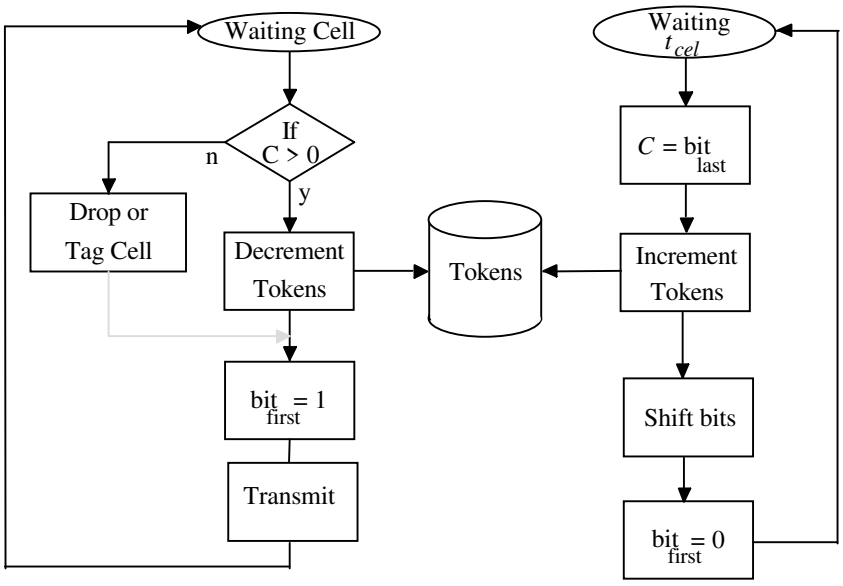


Figure 13: Block diagram for discretized sliding window implementation.

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

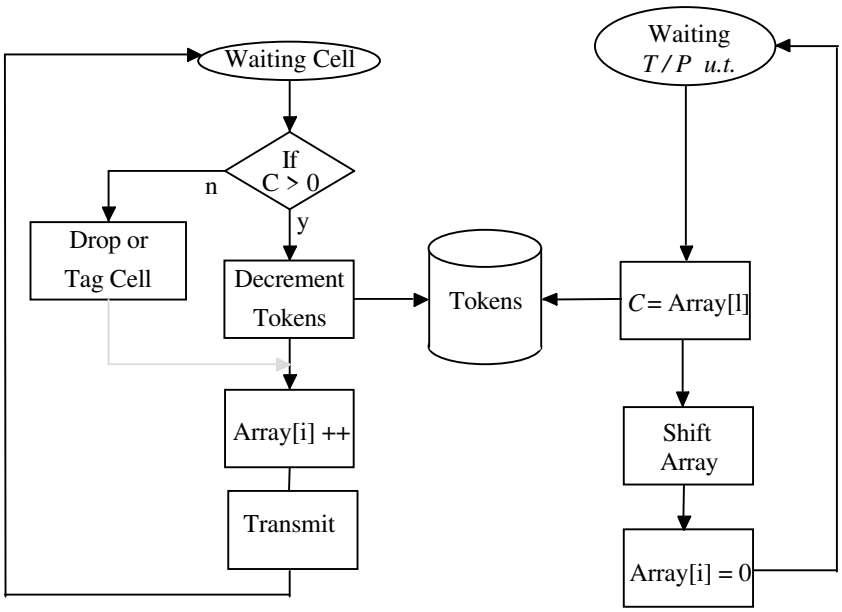


Figure 14: Block Diagram for time slotted moving window implementation.

5 Results

This section presents comparative data obtained from a specific implementation, 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 general and specific for each mechanism processing time of each cell was measured in a work station SUN SPARC 2. The times of all mechanisms were obtained through a mean of 1000 measurements, all of them were achieved in the same machine operation conditions. The station was dedicated for the execution of the mechanism tasks. The computation of the times was obtained through a virtual clock that computed separately the processing time of process. Regarding the obtained results, it can be observed that leaky bucket mechanism, jumping window and time slotted moving window are the most efficient, with general and specific processing times of magnitude of 160μs and 140μs. The continuous sliding window is one of the most expensive mechanisms in relation to processing time - 192μs e 167μs - because of the timers firing at each cell arrival. The leaky bucket with output smoothing presented a high processing cost because of the output queue updating. Concerning the time slotted moving window mechanism, it presented a mean processing time for each cell relatively independent of the used number of slots and of a value intermediary to JW and to CSW, of 176 μs and 150 μs magnitude.

Table 1: Comparison of the mechanisms.

Mechanism	**LB**	**Smooth. LB**	**JW**	**TJW**	**CSW**	**DSW**	**TSMW**
Total Proc. Time	165 μs	194 μs	160 μs	165 μs	192 μs	176 μs	176 μs
Mechanism Proc. Time	143 μs	170 μs	141 μs	142 μs	167 μs	150 μs	150 μs
Number of Timers	1	2	1	1	C	1	1
Timers Precision	variable	variable	low	low	high	high	adjustable
Timers Memory	{Timer}	{Timer}	{Timer}	{Timer}	$C \times$ {Timer}	{Timer}	{Timer}
Counters and Parameters Memory	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$	$\{C\}$ + $\{C_{max}\}$ + $\{P\}$

An important factor to be considered is the precision required by the timers. It can be observed that the continuous sliding window mechanism needs the greatest value - C timers. Besides, these timers demand an extremely high precision, having to be fired at the arrival of each cell, and hence, at any instant of time. It may be emphasized that the greater the number of timers, the greater the CPU utilization. The discretized sliding window is not so critic,

however it still needs a very high precision, for example 2.7μs for a transmission rate of 155Mb/s. The 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 is of 10ms. As regards jumping window, it uses a timer of T time units of magnitude, and therefore its precision, in mean, is not so critic. The time 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 used number of slots.

The required amount of memory for each mechanism 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 time slotted moving window mechanisms that respectively require:

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

to traffic memorization. The additional memory regions are illustrated in table 1. It can be noticed that the continuous sliding window needs $C \times \{\text{Timer}\}$ bits for the timers, while the time slotted moving window requires {P} bits to store the number of slots.

6 Conclusion

This work presented the implementation of policing traffic mechanisms. The implementations were based on a high performance implementation architecture using optimized modules of memory management, task scheduling with priorities and an efficient timer management, in addition to optional modules for quality of service guarantees and media synchronization.

It was analyzed the implementation complexity of the mechanisms, according to the implementation results, in relation to the required amount of memory, the number of timers and its precision, the mean total processing time of each cell and the mean specific processing time of the mechanisms for each cell.

The obtained results demonstrate the highest complexity of sliding window mechanism and the implementation simplicity of leaky bucket and jumping window mechanisms. The time slotted moving window presents the advantage of being flexible, that according to its configuration, offers a functionality set that ranges from the functionality of jumping window, with its simplicity, to the one of continuous sliding window with its accurate control of the mean transmission rate. Furthermore, leaky bucket, jumping window and time 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.

It must be emphasized that the leaky bucket variant with output traffic smoothing needs a high implementation complexity, so that the large number of leaky bucket variants must be carefully analyzed in order to evaluate the trade-off between the advantages and the implementation cost of these variants.

7 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'94, 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. V. Albuquerque, M. L. Lanza, O. C. Duarte, "Sistemas de Comunicação Abertos em Redes Locais: Uma Implementação e Medidas de Desempenho em Ambientes de Computadores Pessoais", XIII Congresso Sociedade Brasileira de Computação (SEMISH), pp. 327-329, Florianópolis, Santa Catarina, Brazil, Sep. 1993.

[8] C. V. Albuquerque, M. D. Nunes, O. C. Duarte, "Performance Measurements in a Manufacturing Communication System", IEEE International Conference on Circuits and Systems ISCAS' 94, pp.3.69-3.72, London, England, May. 1994.

[9] L. F. Baginsky, O. C. Duarte, "Um Modelo de Implementação de Alto Desempenho para Protocolos de Comunicações Estruturados em Camadas", Revista da Sociedade Brasileira de Computação, vol. 6, pp. 46-56, Apr. 1991.

[10] M. D. Nunes, O. C. Duarte, "Análise de Mecanismos para Protocolos de Alto Desempenho", VI Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho, Caxambú, Minas Gerais, Brazil, Aug. 1994.

[11] M. Woo, N. U. Qazi, A. Ghafoor, "A Synchronization Framework for Communication of Pre-Orchestrated Multimedia Information", IEEE Network, pp.52-61, Jan. 1994.

[12] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks", ACM Trans. on Computer Systems, vol. 9, no. 2, May 1991.

[13] S. J. Golestani, "A Stop-and-Go Queueing Framework for Congestion Management", SIGCOMM Symposium, Communications Architecture & Protocols, pp. 8-18, Filadelphia Pensilvania, Sep. 1990.

[14] D.Verma, H. Zhang, D. Ferrari, "Delay Jitter Control for Real Time Communication in a Packet Switching Network", Proceedings of TriComm 91, pp. 35-46, Chapel Hill, Apr. 1991.

[15] H. Zhang, S. Keshav, "Comparison of Rate-Based Service Disciplines", Proceedings of ACM SIGCOMM, pp. 113-121, Sep. 1991.

[16] G. Varghese, T. Lauck, “Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility”, in Proc. 11th ACM SIGOPS Symp. on Operating Systems Principles, pp. 25-38, Austin, TX, 1987.

[17] M.. Faerman, C. V. Albuquerque, O. C. Duarte, “Janela Móvel Particionada: Proposta de um Novo Mecanismo de Policiamento de Tráfego”, Technical Report of Grupo de Teleinformática e Automação da COPPE/UFRJ, 1995.