Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks *

Debanjan Saha[†] *Sarit Mukherjee*[‡] *Satish K. Tripathi*[§]

Abstract

We propose a simple cell scheduling mechanism for ATM networks. The proposed mechanism, named Carry-Over Round Robin (CORR), is an extension of weighted round robin scheduling. We show that albeit its simplicity, CORR achieves tight bounds on end-to-end delay and near perfect fairness. Using a variety of video traffic traces we show that CORR often outperforms some of the more complex scheduling disciplines such as Packet-by-Packet Generalized Processor Sharing (PGPS).

*This work is supported in part by NSF under Grant No. CCR-9318933 and Army Research Laboratory under Cooperative Agreement No. DAAL01-96-2-0002. A short version of the paper appeared in the proceedings of IEEE Infocom'96.

[†]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598. Email:debanjan@watson.ibm.com.

[‡]Dept. of Computer Science & Engg. University of Nebraska, Lincoln, NE 68588. Email:sarit@cse.unl.edu.

[§]Dept. of Computer Science, University of Maryland, College Park, MD 20742. Email:tripathi@cs.umd.edu.

1 Introduction

This paper presents a simple yet effective cell multiplexing mechanism for ATM networks. The proposed mechanism, named Carry-Over Round Robin (CORR), is a simple extension of weighted round robin scheduling. It provides each connection a minimum guaranteed rate of service at the time of connection setup. The excess capacity is fairly shared among active connections. CORR overcomes a common shortcoming of most round robin and frame based scheduler, that is, coupling delay performance and bandwidth allocation granularity. We show that despite its simplicity, CORR often outperforms some of the more sophisticated schemes, such as Packet-by-Packet Generalized Processor Sharing (PGPS) in terms of delay performance and fairness.

Rate based service disciplines for packet switched network is a well studied area of research [1, 2, 4, 7, 11, 8]. Based on bandwidth sharing strategies, most of the proposed schemes can be classified into one of the two categories – (1) fair queuing mechanisms, and (2) frame-based or weighted round robin policies. Virtual clock [11], packet-by-packet generalized processor sharing (PGPS) [3, 7], self clocked fair queueing (SFQ) [6], are the most popular examples of schemes that use fair queueing strategies to guarantee a certain share of bandwidth to a specific connection. The most popular frame-based schemes are Stop-and-Go (SG) [4, 5] and Hierarchical-Round-Robin (HRR) [2]. While fair queueing policies are extremely flexible in terms of allocating bandwidth in very fine granularity and fair distribution of bandwidth among active connections, they are expensive in terms of implementation. Frame-based mechanisms on the other hand are inexpensive in terms of their implementation. However, they suffer from many shortcomings, such as inefficient utilization of bandwidth, coupling between delay performance and bandwidth allocation granularity, unfair allocation of bandwidth.

CORR strives to integrate the flexibility and fairness of the fair queueing strategies with the simplicity of frame-based/round robin mechanisms. The starting point of our algorithm is a simple variation of round robin scheduling. Like round robin, CORR divides the time-line into allocation cycles, and each connection is allocated a fraction of the available bandwidth in each cycle. However, unlike slotted implementations of round robin schemes where bandwidth is allocated as a multiple of a fixed quantum, in our scheme bandwidth allocation granularity can be arbitrarily small. This helps CORR to break the coupling between framing delay and granularity of bandwidth allocation. Another important difference between CORR and frame based schemes, such as SG and HRR is that CORR is a work conserving service discipline. It does not waste the spare capacity of the system, rather share it fairly among active connections. A recent paper [10] proposed a similar idea for efficient implementation of fair queuing. However, the algorithm proposed in [10] has not been analyzed for delay and other related performance metrics.

We have presented detailed analysis of CORR and derived tight bounds on end-to-end delay. Our derivation of delay bounds does not assume a specific traffic arrival pattern. Hence, unlike PGPS (for which delay bound is available only for leaky bucket controlled sources) we can derive end-to-end delay bounds for CORR for a variety of traffic sources. Using traffic traces from real life video sources we have shown that CORR often performs better than PGPS in terms of the size of the admissible region. We have also analyzed the fairness properties of CORR under the most general scenarios and have shown that it achieves nearly perfect fairness.

The rest of this paper is organized as follows. In section 2 we present the intuition behind CORR and its algorithmic description. We discuss the properties of the algorithm in section 3. Section 4 is devoted to the analysis of the algorithm and its evaluation in terms delay performance and fairness. In section 5 we compare the end-to-end performance of CORR with PGPS and SG using a variety of traffic traces. We conclude the paper in section 6.

2 Scheduling Algorithm

Like round robin scheduling, CORR divides the time line into allocation cycles. The maximum length of an allocation cycle is T. Let us assume that the cell transmission time is the basic unit of time. Hence, the maximum number of cells (or slots) transmitted during one cycle is T. At the time of admission, each connection C_i is allocated a rate R_i expressed in cells per cycle. Unlike simple round robin schemes, where R_is have to be integers, CORR allows R_is to be real. Since R_is can take real values, the granularity of bandwidth allocation can be arbitrarily small, irrespective of the length of the allocation cycle. The goal of the scheduling algorithm is to allocate each connection C_i close to R_i slots in each cycle and exactly R_i slots per cycle over a longer time frame. It also distributes the excess bandwidth among the active connections C_is in the proportion of their respective R_is.

The CORR scheduler (see figure 1) consists of three asynchronous events — *Initialize*, *Enqueue*, and *Dispatch*. The event *Initialize* is invoked when a new connection is admitted. If a connection is admissible [1], it simply adds the connection to the connection-list $\{C\}$. The connection-list is ordered in the decreasing order of $R_i - \lfloor R_i \rfloor$, that is, the fractional part of R_i. The event *Enqueue* is activated at the arrival of a packet. It puts the packet in the appropriate connection queue and updates the cell count of the connection. The most important event in the scheduler is *Dispatch*. The event *Dispatch* is invoked at the beginning of a busy period. Before explaining the task performed by *Dispatch*, let us introduce the variables and constants used in the algorithm and the basic intuition behind it.

The scheduler maintains separate queues for each connection. For each connection C_i, n_i keeps the count of the waiting cells, and r_i holds the number of slots currently credited to it. Note that r_is can be real as well as negative fractions. A negative value of r_i signifies that the connection has been allocated more slots than it deserves. A positive value of r_i reflects the current legitimate requirement of the connection. In order to allocate slots to meet the requirement of the connection as closely as possible, CORR divides each allocation cycle into two sub-cycles — a *major cycle* and a *minor cycle*. In the major cycle, integral requirement of each connection is satisfied first. Slots left over from major cycle are allocated in minor cycle to connections with still unfulfilled fractional requirements. Obviously, a fraction of a slot cannot be allocated. Hence, eligible connections are allocated a full slot each in the minor cycle whenever slots are available. However, all the connections

[1] We discuss admission control later.

Constants

T: Cycle length.

R_i: Slots allocated to C_i.

Variables

$\{C\}$: Set of all connections.

t: Slots left in current cycle.

n_i: Number of cells in C_i.

r_i: Current slot allocation of C_i.

Events

Initialize(C_i) /* Invoked at connection setup time. */

  add C_i to $\{C\}$; /* $\{C\}$ is ordered in decreasing order of $R_i - \lfloor R_i \rfloor$. */

  $n_i \leftarrow 0$; $r_i \leftarrow 0$;

Enqueue() /* Invoked at cell arrival time. */

  $n_i = n_i + 1$

  add cell to connection queue;

Dispatch() /* Invoked at the beginning of a busy period. */

  $\forall C_i$:: $r_i \leftarrow 0$;

  while not end-of-busy-period **do**

   $t \leftarrow T$;

   1. Major Cycle:

    for all $C_i \in \{C\}$ **do** /* From head to tail. */

     $r_i \leftarrow min(n_i, r_i + R_i)$; $x_i \leftarrow min(t, \lfloor r_i \rfloor)$;

     $t \leftarrow t - x_i$; $r_i \leftarrow r_i - x_i$; $n_i \leftarrow n_i - x_i$;

     dispatch x_i cells from connection queue C_i;

    end for

   2. Minor Cycle:

    for all $C_i \in \{C\}$ **do** /* From head to tail. */

     $x_i \leftarrow min(t, \lceil r_i \rceil)$;

     $t \leftarrow t - x_i$; $r_i \leftarrow r_i - x_i$; $n_i \leftarrow n_i - x_i$;

     dispatch x_i cells from connection queue C_i;

    end for

  end while

Figure 1: Carry-Over Round Robin Scheduling.

with fractional requirements may not be allocated a slot in the minor cycle. The connections that get a slot in the minor cycle over-satisfy their requirements and carry a debit to the next cycle. The eligible connections that do not get a slot in the minor cycle carry a credit to the next cycle. The allocations for the next cycle are adjusted to reflect this debit and credit carried over from the last cycle. Following is a detailed description of the steps taken in the *Dispatch* event.

At the beginning of a busy period, all r_is are set to 0 and a new cycle is initiated. The cycles continue until the end of the busy period. At the beginning of each cycle, the current number of unallocated slots t is initialized to T, and the major cycle is initiated. In the major cycle, the *dispatcher* cycles through connection-list and, for each connection C_i, updates r_i to $r_i + R_i$. If the number of cells queued in the connection queue, n_i, is less than the updated value of r_i, r_i is set to n_i. This is to make sure that a connection cannot accumulate credits. The minimum of t and $\lfloor r_i \rfloor$ cells are dispatched from the connection queue of C_i. The variables are appropriately adjusted after dispatching the cells. A minor cycle starts with the slots left over from preceding major cycle. Again, the *dispatcher* walks through the connection-list. As long as there are slots left, a connection is deemed eligible for dispatching iff 1) it has queued packets, and 2) its r_i is greater than zero. If there is no eligible connection or if t reaches zero, the cycle ends. Note that the length of the major and minor cycles may be different in different allocation cycles.

Example: Let us consider a CORR scheduler with cycle length $T = 4$ and serving three connections C_1, C_2, and C_3 with $R_1 = 2$, $R_2 = 1.5$, and $R_3 = 0.5$, respectively. In an ideal system where fractional slots can be allocated, slots can be allocated to the connections in a fashion shown in figure 2, resulting in full utilization of the system. CORR also achieves full utilization, but with a different allocation of slots.

For ease of exposition, let us assume that all three connections are backlogged starting from the beginning of the busy period. In the major cycle of the first cycle, CORR allocates C_1, C_2, and C_3, $\lfloor R_1 \rfloor = 2$, $\lfloor R_2 \rfloor = 1$, and $\lfloor R_3 \rfloor = 0$ slots, respectively. Hence, at the beginning of the first minor cycle, $t = 1$, $r_1 = 0.0$, $r_2 = 0.5$, and $r_3 = 0.5$. The only slot left over for the minor cycle goes to C_2. Consequently, at the end of the first cycle, $r_1 = 0.0$, $r_2 = -0.5$, and $r_3 = 0.5$, and the adjusted requirements for the second cycle are

$$r_1 = r_1 + R_1 = 0.0 + 2.0 = 2.0$$

$$r_2 = r_2 + R_2 = -0.5 + 1.5 = 1.0$$

$$r_3 = r_3 + R_3 = 0.5 + 0.5 = 1.0$$

Since all the r_is are integral, they are all satisfied in the major cycle.

The main attraction of CORR is its simplicity. In terms of complexity, CORR is comparable to round robin and frame based mechanisms. However, CORR does not suffer from the shortcomings of round robin and frame based schedulers. By allowing the number of slots allocated to a connection in an allocation cycle to be a real number instead of an integer, we break the coupling between the service delay and bandwidth allocation granularity. Also, unlike frame based mechanisms, such as SG and HRR, CORR is a work conserving discipline capable of exploiting the multiplexing gains of packet switching. In the following section we discuss some of its basic properties.

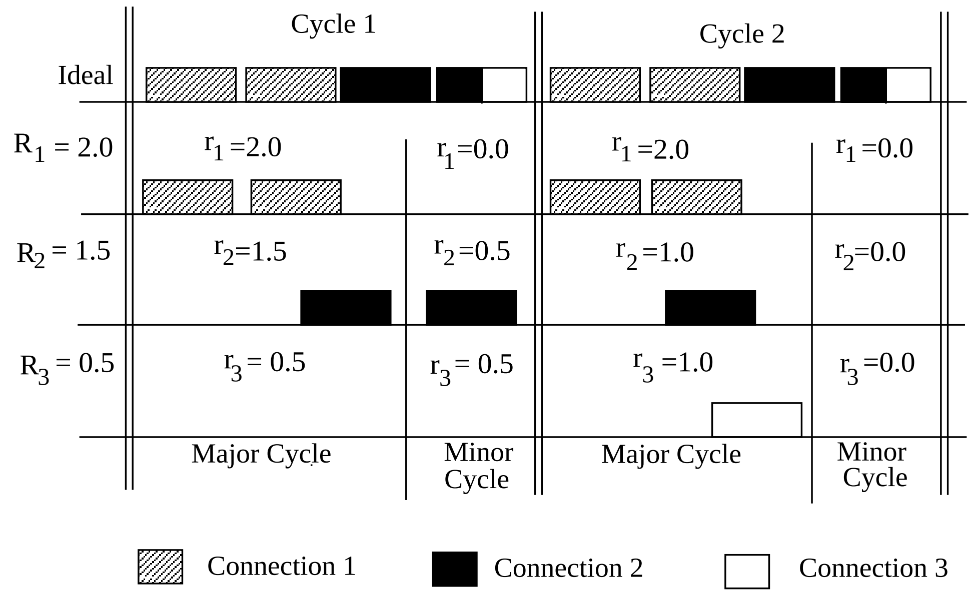

Figure 2: An Example Allocation.

3 Basic Properties

In this section, we discuss some of the basic properties of the scheduling algorithm. Lemma 3.1 defines an upper bound on the aggregate requirements of all streams inherited from the last cycle. This result is used in lemma 3.2 to determine the upper and lower bounds on the individual requirements carried over from the last cycle by each connection.

Lemma 3.1 *If* $\sum_{\forall C_i \in \{C\}} R_i \leq T$ *then at the beginning of each cycle* $\sum_{\forall C_i \in \{C\}} r_i \leq 0$.

Proof: We will prove this by induction. We will first show that it holds at the beginning of a busy period, and then we will show that if it hold in the kth cycle, it also holds in the $(k+1)^{th}$ cycle.

Base Case: From the allocation algorithm we observe that at the beginning of a busy period $r_i = 0$ for all connection C_i. Hence,

$$\sum_{\forall C_i \in \{C\}} r_i = 0$$

Thus, the assertion holds in the base case.

Inductive Hypothesis: Assume that the premise holds in the kth cycle. We need to prove that it also holds in the $(k+1)^{th}$ cycle. We use superscript for cycles in the following proof.

$$\sum_{\forall C_i \in \{C\}} r_i^{k+1} = \sum_{\forall C_i \in \{C\}} r_i^k + \sum_{\forall C_i \in \{C\}} R_i - T \leq 0 + T - T \leq 0.$$

This completes the proof. $\square$

Henceforth we assume that the admission control mechanism makes sure that $\sum_{\forall C_i \in \{C\}} R_i \leq T$ at all nodes. This simple admission control test is one of the attractions of the CORR scheduling.

Lemma 3.2 *If $\sum_{\forall C_i \in \{C\}} R_i \leq T$ then at the beginning of each cycle*

$$-1 < -\delta_i \leq r_i \leq \delta_i < 1,$$

$$where \;\; \delta_i = max_k\{kR_i - \lfloor kR_i \rfloor\}, k = 1, 2, \ldots$$

Proof: To derive the lower bound on r_i, observe that in each cycle no more than $\lceil r_i \rceil$ slots are allocated to connection C_i. Also, note that r_i is incremented in steps of R_i. Hence, the lowest r_i can get to is,

$$\begin{aligned} -\delta_i &= max_k\{kR_i - \lceil kR_i \rceil\}, k = 1, 2, \ldots \infty \\ &= -max_k\{kR_i - \lfloor kR_i \rfloor\}, k = 1, 2, \ldots \infty \end{aligned}$$

Derivation of the upper bound is little more complex. Let us assume that there are n connections C_i, $i = 1, 2, \ldots, n$. With no loss of generality we can renumber them, such that $R_i \geq R_j$, when $i < j$. For the sake of simplicity, let us also assume that all the R_is are fractional. We show later that this assumption is not restrictive. To prove the upper bound we first prove that r_i never exceeds 1, for all connections C_i. Now, since R_n is the lowest of all R_is, $i = 1, 2, \ldots, n$, C_n is the last connection in the connection list. Consequently, C_n is the last connection considered for a possible cell dispatch in both major and minor cycles. Hence, if we can prove that r_n never exceeds 1, so is true for all other r_is. We will prove this by contradiction.

Let us assume that C_n enters a busy period in allocation cycle 1. Observe, that C_n experiences the worst case allocation when all other connections also enter their busy period in the same cycle. Let us assume that r_n exceeds 1. This would happen in the allocation cycle $\lceil 1/R_n \rceil$. Since r_n exceeds 1, C_n is considered for a possible dispatch in the major cycle. Now, C_n is not scheduled during the major cycle of the allocation cycle $\lceil 1/R_n \rceil$ if and only if the following is true at the beginning of the allocation cycle,

$$\sum_{i=1}^{n-1} \lfloor r_i + R_i \rfloor \geq T$$

From lemma 3.1 we know that, $\sum_{i=1}^{n} r_i \leq 0$ at the beginning of each cycle. Since $r_n > 0$, $\sum_{i=1}^{n-1} r_i < 0$ at the beginning of the allocation cycle $\lceil 1/R_n \rceil$. But,

$$\sum_{i=1}^{n-1} \lfloor r_i + R_i \rfloor \leq \sum_{i=1}^{n-1} (r_i + R_i) < 0 + \sum_{i=1}^{n-1} R_i < T$$

This contradicts our original premise. Hence, r_n cannot exceed 1. Noting that r_n is incremented in steps of R_n the bound follows.

We have proved the bounds under the assumption that all R_is are fractional. If we relax that assumption, the result still holds. This is due to the fact that the integral part of R_i is guaranteed to be allocated in each allocation cycle. Hence, even when R_is are not all fractional, we can reduce the problem to an equivalent one with fractional R_is using the transformation

$$\hat{R}_i = R_i - \lfloor R_i \rfloor \quad \mathit{and} \quad \hat{T} = T - \sum_{i=1}^{n} \lfloor R_i \rfloor.$$

This completes the proof. □

4 Quality of Sevice Envelope

In this section we analyze the worst-case end-to-end delay performance of CORR. Other measures of performance such as delay jitter and buffer size at each node can also be found from the results derived in this section. In order to find the end-to-end delay, we have first derived delay bounds for a single node system. We then show that the end-to-end delay for a multi-node system can be reduced to delay encountered in an equivalent single node system. Hence, the delay bounds derived for single node system can be substituted to find the end-to-end delay. We have also presented a comprehensive analysis of CORR's fairness. We define a fairness index to quantify how fairly bandwidth is allocated among active connections, and show that the fairness index of CORR is within a constant factor of any scheduling discipline.

4.1 Delay Analysis

In this section we derive the worst case delay bounds of a connection spanning single or multiple nodes, each employing CORR scheduling to multiplex traffic from different connections. We assume that each connection has an associated traffic envelope that describes the characteristics of the traffic it is carrying, and a minimum guaranteed rate of service at each multiplexing node. Our goal is to determine the maximum delay suffered by any cell belonging to the connection. We start with a simple system consisting of a single multiplexing node, and find the worst case delay for different traffic envelopes.

Single-node Case

Let us consider a single server employing CORR scheduling to service traffic from different connections. Since we are interested in the worst case delay behavior, and each connection is guaranteed a minimum rate of service, we can consider each connection in isolation. Our problem then is to find the maximum difference between the arrival and the departure times of any cell, assuming that the cells are serviced using CORR scheduling with a minimum guaranteed rate of service. The arrival time of a cell can be obtained from the traffic envelope defined by the traffic envelope associated with a connection. Traffic envelope associated with a connection depends on the shaping mechanism (see Appendix) used at the network entry point. In this paper we have considered leaky bucket and moving window shapers.

Following, we derive the worst case departure time a cell in terms of the service rate allocated to the connection and the length of the allocation cycle. Knowing both the arrival and the departure functions, we can compute the worst case delay bound. Before presenting the results let us first formally introduce the definition of a connection busy period and a system busy period.

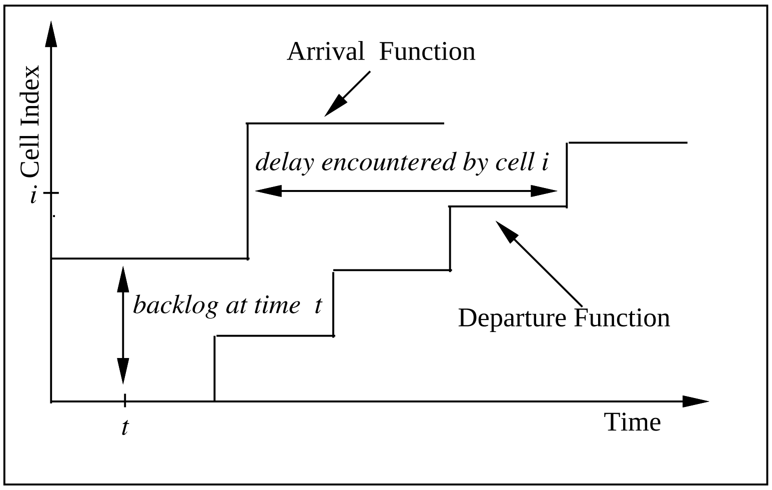

Figure 3: Computing delay and backlog from the arrival and departure functions.

Definition 4.1 *A connection is said to be in the busy period if connection queue is non-empty. The system is said to be in the busy period if at least one of the active connection is in the its busy period.*

Note, that a particular connection can switch between busy and idle periods even when the system is in the same busy period. The following theorem determines the departure time of a specific cell belonging to a particular connection.

Theorem 4.1 *Assume that a connection enters a busy period at time* 0*. Let* $d(i)$ *be the latest time by which the* i^{th} *cell, starting from the beginning of the current busy period, departs the system. Then* $d(i)$ *can be expressed as,*

$$d(i) = \left\lceil \frac{i+1+\delta}{R} \right\rceil T, \quad i = 0, 1, \ldots, \infty.$$

where R *is the rate allocated to the connection and* T *is the maximum length of the allocation cycle.*

Proof: Since a cell may leave the system any time during an allocation cycle, to capture the worst case we assume that all the cells served during an allocation cycle leave at the end of the cycle. Now, when a connection enters a busy period, in the worst case, $r = -\delta$. If cell i departs at the end of the allocation cycle L, the number of slots allocated by the scheduler is $L \times R + \delta$ and the number of slots consumed is $i+1$ (since packet number starts from 0). In the worst case,

$$1 > L \times R - \delta - (i+1) \geq 0.$$

This implies that,

$$\frac{i+1+\delta+1}{R} > L \geq \frac{i+1+\delta}{R}.$$

From the above inequality and noting that L is an integer and $d(i) = L \times T$, we get

$$d(i) = \left\lceil \frac{i+1+\delta}{R} \right\rceil T.$$

□

Theorem 4.1 precisely characterizes the departure function $d(\cdot)$ associated with a connection. As mentioned earlier, the arrival function $a(\cdot)$ associated with a connection is determined by the traffic envelope and has been characterized for different composite shapers in the Appendix. Knowing both $a(i)$ and $d(i)$, the arrival and departure time of the i^{th} cell in a busy period, delay encountered by the i^{th} cell can be computed as $d(i) - a(i)$ (see figure 3). Note that this is really the horizontal distance between the arrival and departure functions at i. Hence, the maximum delay encountered by any cell is really the maximum horizontal distance between the arrival and the departure functions. Similarly, the vertical distance between these functions represents the backlog in the system. Unfortunately, finding the maximum delay, that is the maximum horizontal difference between the arrival and the departure functions, is a difficult task. Hence, instead of finding the maximum delay directly by measuring the horizontal distance between these functions, we first determine the point at which the maximum backlog occurs and the index i of the cell which is at the end of the queue at that point. The worst case delay is then computed by evaluating $d(i) - a(i)$. Following we carry out this procedures for $a(i)$ defined by composite leaky bucket, and moving window shapers.

Lemma 4.1 *Consider a connection shaped using an n-component moving window shaper and passing through a single multiplexing node employing CORR scheduling with an allocation cycle of length T* [2]*. If the connection is allocated a service rate of R, then the worst case delay encountered by any cell belonging to the connection is upper bounded by,*

$$D^{CORR/MW} \leq \begin{cases} \left\lceil \dfrac{m_j+\delta}{R} \right\rceil T - \displaystyle\sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l, & \left\lceil \dfrac{R+\delta}{w_j\,(R/T - m_j/w_j)} \right\rceil = 1 \\ \left\lceil \dfrac{2m_j+\delta}{R} \right\rceil T - w_j - \displaystyle\sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l, & \left\lceil \dfrac{R+\delta}{w_j\,(R/T - m_j/w_j)} \right\rceil > 1 \end{cases}$$

$$when \qquad \frac{m_j}{w_j} < \frac{R}{T} < \frac{m_{j+1}}{w_{j+1}}, \qquad j = 1, 2, \ldots, n-1.$$

[2] We assume that the cycle length is smaller than the smallest window period.

Proof: First we will show that under the conditions stated above the system is stable. That is, the length of the busy period is finite. To prove that, it is sufficient to show that there exists a positive integer k such that, the number of cells serviced in kw_j time is more than equal to km_j. In other words, we have to show that there exists a k such that the following holds,

$$\begin{aligned}
kw_j &\geq d(km_j - 1) \\
kw_j &\geq \left\lceil \frac{(km_j - 1) + 1 + \delta}{R} \right\rceil T \\
kw_j &\geq \left[\frac{(km_j - 1) + 1 + \delta}{R} + 1 \right] T \\
k &\geq \frac{R + \delta}{w_j \left(R/T - m_j/w_j \right)}
\end{aligned}$$

Clearly, for there to exists a positive integer k so that the above equality is satisfied, the following condition needs to hold.

$$R/T - m_j/w_j > 0 \quad or \quad R/T > m_j/w_j$$

By our assumption, $R/T > m_j/w_j$. Hence, the system is stable. Now, we need to determine the point at which the maximum backlog occurs. Depending on the value of k, the maximum backlog can occur at one of the two places.

Case 1: $k = 1$. If $k = 1$, that is, when traffic coming in during a time window of length w_j departs the system in the same window, the maximum backlog occurs at the arrival instant of the $(m_j - 1)^{th}$ cell [3]. Clearly, the index of the cell at the end of the queue at that instant is $(m_j - 1)^{th}$. Hence, the maximum delay encountered by any cell under this scenario is the same as the delay suffered by the $(m_j - 1)^{th}$ cell, and can be enumerated by computing $d(m_j - 1) - a(m_j - 1)$. We can evaluate $a(m_j - 1)$ as following,

$$\begin{aligned}
a(m_j - 1) &= \sum_{l=1}^{n} \left(\left\lfloor \frac{m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\
&= \sum_{l=1}^{j} \left(\left\lfloor \frac{m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\
&+ \sum_{l=j+1}^{n} \left(\left\lfloor \frac{m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\
&= \sum_{l=j+1}^{n} \left(\frac{m_j}{m_l} - 1 - \left(\frac{m_j}{m_{l-1}} - 1 \right) \frac{m_{l-1}}{m_l} \right) w_l \qquad (\text{since } m_{l-1} > m_l)
\end{aligned}$$

[3] Note that cells are numbered from 0. Since R can be non-integral, we can choose T arbitrarily small without affecting the granularity of bandwidth allocation

$$= \sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l.$$

Therefore, the worst case delay is bounded by,

$$\begin{aligned} D^{CORR/MW} &\leq d(m_j - 1) - a(m_j - 1) \\ &\leq \left\lceil \frac{m_j + \delta}{R} \right\rceil T - \sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l. \end{aligned}$$

Case 2: $k > 1$. When k is greater than 1, the connection busy period continues beyond the first window of length w_j. Since $R/T > m_j/w_j$, the rate of traffic arrival is lower than the rate of departure. Still, in this case, not all cells that arrive during the first window of length w_j are served during that period and the left over cells are carried over into the next window. This is due to the fact that unlike the arrival curve, the departure function does not start at time 0 but at time $\lceil (1+\delta)/R \rceil$. This is the case when $k = 1$ also. However, in that case, the rate of service is high enough to serve all the cells before the end of the first window. When $k > 1$ the backlog carried over from the first window is cleared in portions over the next $k - 1$ windows. Clearly, the backlogs carried over into subsequent windows diminish in size and is cleared completely by the end of the k^{th} window. Hence, second window is the one where the backlog inherited from the last window is the maximum. Consequently, absolute backlog in the system reaches its maximum at the arrival of the $2m_j - 1$ cell. Hence, the maximum delay encountered by any cell under this scenario is the same as the delay suffered by the $(2m_j - 1)^{th}$ cell, and can be enumerated by computing $d(2m_j - 1) - a(2m_j - 1)$. We can evaluate $a(2m_j - 1)$ as following,

$$\begin{aligned} a(2m_j - 1) &= \sum_{l=1}^{n} \left(\left\lfloor \frac{2m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{2m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\ &= \sum_{l=1}^{j-1} \left(\left\lfloor \frac{2m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{2m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\ &\quad + \left(\left\lfloor \frac{2m_j - 1}{m_j} \right\rfloor - \left\lfloor \frac{2m_j - 1}{m_{j-1}} \right\rfloor \frac{m_{j-1}}{m_j} \right) w_j \\ &\quad + \sum_{l=j+1}^{n} \left(\left\lfloor \frac{2m_j - 1}{m_l} \right\rfloor - \left\lfloor \frac{2m_j - 1}{m_{l-1}} \right\rfloor \frac{m_{l-1}}{m_l} \right) w_l \\ &= 0 + w_j + \sum_{l=j+1}^{n} \left(\frac{2m_j}{m_l} - 1 - \left(\frac{2m_j}{m_{l-1}} - 1 \right) \frac{m_{l-1}}{m_l} \right) w_l \qquad (\text{since } m_{l-1} \geq 2m_l) \\ &= w_j + \sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l. \end{aligned}$$

Therefore the worst case delay is bounded by,

$$D^{CORR/MW} \leq d(2m_j - 1) - a(2m_j - 1)$$

$$\leq \left\lceil \frac{2m_j + \delta}{R} \right\rceil T - w_j - \sum_{l=j+1}^{n} \left(\frac{m_{l-1}}{m_l} - 1 \right) w_l.$$

□

Lemma 4.2 *Consider a connection shaped by an n-component leaky bucket shaper and passing through a single multiplexing node employing CORR scheduling with an allocation cycle of maximum length T. If the connection is allocated a service rate of R, then the worst case delay suffered by any cell belonging to the connection is upper bounded by,*

$$D_{max}^{CORR/LB} \leq \left\lceil \frac{B_j + 1 + \delta}{R} \right\rceil T - (B_j - b_j + 1)\, t_j,$$

$$\textit{when} \quad \frac{1}{t_j} < \frac{R}{T} < \frac{1}{t_{j+1}}, \qquad j = 1, 2, \ldots, n.$$

Proof: In order to identify the point where maximum the backlog occurs, observe that the rate of arrivals is more than the rate of service until the slope of the traffic envelope changes from $\frac{1}{t_{j+1}}$ to $\frac{1}{t_j}$. This change in the slope occurs at the arrival of the ${B_j}^{th}$ cell in the worst case. Hence, the maximum delay encountered by any cell is at most as large as the delay suffered by ${B_j}^{th}$ cell. We can compute $a(B_j)$ as following,

$$\begin{aligned}
a(B_j) &= \sum_{l=1}^{n+1} (B_j - b_l + 1)\, t_l\, [U(B_j - B_l) - U(B_j - B_{l-1})] \\
&= \sum_{l=1}^{j-1} (B_j - b_l + 1)\, t_l\, [U(B_j - B_l) - U(B_j - B_{l-1})] \\
&\quad + (B_j - b_j + 1)\, t_j\, [U(B_j - B_j) - U(B_j - B_{j-1})] \\
&\quad + \sum_{l=j+1}^{n+1} (B_j - b_l + 1)\, t_l\, [U(B_j - B_l) - U(B_j - B_{l-1})] \\
&= (B_j - b_j + 1)\, t_j.
\end{aligned}$$

Now $d(B_j) - a(B_j)$ yields the result. □

The results derived in this section define tight upper bounds for delay encountered in a CORR scheduler under different traffic arrival patterns. The compact closed form expressions make the task of computing the numerical bounds for a specific set of parameters very simple. We would also like to mention that compared to other published works, we consider a much larger and general set of traffic envelopes in our analysis. Although simple closed form bounds under very general arrival patterns is an important contribution of this work, bounds for a single node system is not very useful in a real life scenario. In most real systems, a connection spans multiple nodes and the end-to-end delay bound is what is of interest. In the following section we derive bounds on end-to-end delay using the results presented in this section.

Multiple-node Case

In the last section we derived worst case bounds on delay for different traffic envelopes for a single node system. In this section, we derive similar bounds for a multi-node system. We assume that there are n multiplexing nodes between the source and the destination, and at each node a minimum available rate of service is guaranteed.

We denote by $a_k(i)$ the arrival time of cell i at node k. The service time at node k for cell i is denoted by $s_k(i)$. We assume that the propagation delay between nodes is zero[4]. Hence, the departure time of cell i from node k is $a_{k+1}(i)$. Note, that $a_1(i)$ is the arrival time of the i^{th} cell in the system and $a_{n+1}(i)$ is the departure time of the i^{th} cell from the system.

Let us denote by $S_k(p,q) = \sum_{i=p}^{q} s_k(i)$. This is nothing but the aggregate service times of cells p through q at node k. In other words, $S_k(p,q)$ is the service time of the burst of cells p through q at node k.

The following theorem expresses the arrival time of a particular cell at a specific node in terms of the arrival times of the preceding cells at the system and their service times at different nodes. This is a very general result, and is independent of the particular scheduling discipline used at the multiplexing node and traffic envelope associated with the connection. We will use this result later to derive the worst case bound on end-to-end delay.

Theorem 4.2 *For any node k and for any cell the i, the following holds:*

$$a_k(i) = \max_{1 \le j \le i} \left\{ a_1(j) + \max_{j=l_1 \le l_2 \le \cdots \le l_k = i} \left(\sum_{h=1}^{k-1} S_h(l_h, l_{h+1}) \right) \right\}.$$

Proof: We will prove this theorem by induction on k and i.

Induction on k:

Base Case: When $k = 1$,

$$\begin{aligned} a_1(i) &= \max_{1 \le j \le i} \left\{ a_1(j) + \max_{j=l_1 \le l_2 \le \cdots \le l_k = i} \left(\sum_{h=1}^{0} S_h(l_h, l_{h+1}) \right) \right\} \\ &= a_1(i) \end{aligned}$$

Clearly, the assertion holds.

Inductive Hypothesis: Let us assume that the premise holds for all $m \le k$. In order to prove that the hypothesis is correct, we need to show that it holds for $m = k+1$.

$$a_{k+1}(i) = \max\{a_{k+1}(i-1) + s_k(i), a_k(i) + s_k(i)\}$$

[4] This assumption does not affect the generality of the results, since the propagation delay at each stage is constant and can be included in $s_k(i)$.

$$
\begin{aligned}
&= \max\left\{ \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_{k+1}=i-1}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right)\right] + s_k(i),\right. \\
&\quad \left.\max_{1\le j\le i}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k=i}\left(\sum_{h=1}^{k-1} S_h(l_h,l_{h+1})\right)\right] + s_k(i)\right\} \\
&= \max\left\{ \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_{k+1}=i-1}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right) + s_k(i)\right],\right. \\
&\quad \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k=i}\left(\sum_{h=1}^{k-1} S_h(l_h,l_{h+1})\right) + S_k(i,i)\right], \\
&\quad \left. a_1(i) + \sum_{h=1}^{i} S_h(i,i)\right\} \\
&= \max\left\{ \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k< l_{k+1}=i}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right)\right],\right. \\
&\quad \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k=l_{k+1}=i}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right)\right], \\
&\quad \left. a_1(i) + \sum_{h=1}^{k} S_h(i,i)\right\} \\
&= \max\left\{ \max_{1\le j\le i-1}\left[a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k\le l_{k+1}=i}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right)\right],\right. \\
&\quad \left. a_1(i) + \sum_{h=1}^{i} S_h(i,i)\right\} \\
&= \max_{1\le j\le i}\left\{a_k(j) + \max_{j=l_1\le l_2\le\cdots\le l_k\le l_{k+1}=i}\left(\sum_{h=1}^{k} S_h(l_h,l_{h+1})\right)\right\}
\end{aligned}
$$

Induction on i:

Base Case: When $i = 1$,

$$
\begin{aligned}
a_k(1) &= \max_{1\le j\le 1}\left\{a_1(j) + \max_{j=l_1\le l_2\le\cdots\le l_k=1}\left(\sum_{h=1}^{k-1} S_h(l_h,l_{h+1})\right)\right\} \\
&= a_1(1) + \sum_{h=1}^{k-1} S_h(1,1) \\
&= a_1(1) + \sum_{h=1}^{k-1} s_h(1)
\end{aligned}
$$

Hence, the assertion holds in the base case.

Inductive Hypothesis: Let us assume that the premise holds for all $n \le i$. In order to prove that the hypothesis is correct, we need to show that it holds for $n = i + 1$.

$$
\begin{aligned}
a_k(i+1) &= \max\{a_k(i)+s_{k-1}(i+1), a_{k-1}(i+1)+s_{k-1}(i+1)\} \\
&= \max\Bigg\{\max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_k=i}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)\right]+s_{k-1}(i+1), \\
&\quad \max_{1\le j\le i+1}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}=i+1}\left(\sum_{h=1}^{k-2}S_h(l_h,l_{h+1})\right)\right]+s_{k-1}(i+1)\Bigg\} \\
&= \max\Bigg\{\max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_k=i}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)+s_{k-1}(i+1)\right], \\
&\quad \max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}=i+1}\left(\sum_{h=1}^{k-2}S_h(l_h,l_{h+1})\right)+S_{k-1}(i+1,i+1)\right], \\
&\quad a_1(i+1)+\sum_{h=1}^{k-1}S_h(i+1,i+1)\Bigg\} \\
&= \max\Bigg\{\max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}<l_k=i+1}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)\right], \\
&\quad \max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}=l_k=i+1}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)\right], \\
&\quad a_1(i+1)+\sum_{h=1}^{i}S_h(i+1,i+1)\Bigg\} \\
&= \max\Bigg\{\max_{1\le j\le i}\left[a_1(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}\le l_k=i+1}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)\right], \\
&\quad a_1(i+1)+\sum_{h=1}^{k}S_h(i+1,i+1)\Bigg\} \\
&= \max_{1\le j\le i+1}\left\{a_k(j)+\max_{j=l_1\le l_2\le\cdots\le l_{k-1}\le l_k=i+1}\left(\sum_{h=1}^{k-1}S_h(l_h,l_{h+1})\right)\right\}
\end{aligned}
$$

□

The result stated in the above theorem determines the departure time of any cell from any node in the system in terms of the arrival times of the preceding cells and the service times of the cells at different nodes. This is the most general result known to us on the enumeration of end-to-end delay in terms of service times of cells at intermediate nodes. We believe, this result will prove to be a powerful tool in enumerating end-to-end delay for any rate based scheduling discipline and will be an effective alternative for the ad hoc techniques commonly used for end-to-end analysis.

Although the result stated in theorem 1 is very general, it is difficult to make use of it in its most general form. In order to find the exact departure time of any cell from any node we need to know both the arrival times of the cells and their service times at different nodes. Arrival times of different cells can be obtained from the arrival function, but computing service times for different cells at each node is a daunting task. Hence, computing precise departure time of a cell from any node in the system is often quite difficult. However, accurate departure time of a specific cell is rarely of critical interest. More often we are interested in other metrics, such as worst case delay encountered by a cell. Fortunately, computing the worst case bound on the departure time, and then the worst case delay is not that difficult. The following corollary expresses the worst case delay suffered by a cell in terms of the worst case service times at each node.

Corollary 4.1 *Consider a connection passing through n multiplexing nodes. Assume that there exists a S_w such that $S_w(p,q) \geq S_h(p,q)$ for all $q \geq p$ and $h = 1,2,\ldots n$. Then, in the worst case, delay $D(i)$ suffered by the cell i belonging to the connection can be upper bounded by*

$$D(i) \leq \max_{1\leq j\leq i} \left\{ a_1(j) + \max_{j=l_1\leq l_2\leq\cdots\leq l_{n+1}=i} \left\{ \sum_{h=1}^{n} S_w(l_h, l_{h+1}) \right\} \right\} - a_1(i)$$

Proof: Follows trivially from theorem 1 by substituting S_hs, $h = 1,2,\ldots n$ by S_w. □

Corollary 4.1 expresses the worst case delay encountered by any cell under the assumption that for any p and q there exists a function S_w such that $S_w(p,q) \geq S_h(p,q)$, for $h = 1,2,\ldots,n$. The closer S_w is to S_h, the tighter is the bound. The choice of S_w depends on the particular scheduling discipline used at the multiplexing nodes. In case of CORR it is simply the service time at the minimum guaranteed rate of service. Following corollary instantiates the delay bound for CORR service discipline.

Corollary 4.2 *Consider a connection traversing n nodes, each of which employs CORR scheduling discipline. Let R_w be the minimum rate of service offered to a connection at the bottleneck node, and T be the maximum length of the allocation cycle. Then the worst case delay suffered by the i^{th} cell belonging to the connection is bounded by,*

$$D^{CORR}(i) \leq \left[n + (n-1)\frac{2+\delta_w}{R_w} \right] T + \max_{1\leq j\leq i} \{a_1(j) + S_w(j,i)\} - a_1(i)$$

Proof: This follows from corollary 4.1 by replacing

$$\max_{j=l_1\leq l_2\leq\cdots\leq l_{n+1}=i} \left\{ \sum_{h=1}^{n} S_w(l_h, l_{h+1}) \right\} \quad with \quad \left[n + (n-1)\frac{2+\delta_w}{R_w} \right] T + S_w(j,i)$$

Following steps explain the details,

$$\begin{aligned} \sum_{h=1}^{n} S_w(l_h, l_{h+1}) &= \sum_{h=1}^{n} \left\lceil \frac{l_{(h+1} - l_h + 1) + 1 + \delta_w}{R_w} \right\rceil \qquad (from \ \ theorem\ 4.1) \\ &\leq \sum_{h=1}^{n} \left[\frac{l_{h+1} - l_h + 2 + \delta_w}{R_w} + 1 \right] T \end{aligned}$$

$$
\begin{aligned}
&\leq \left[n + (n-1)\frac{2+\delta_w}{R_w}\right]T + \left[\frac{(l_{n+1} - l_1 + 1) + 1 + \delta_w}{R_w}\right]T \\
&\leq \left[n + (n-1)\frac{2+\delta_w}{R_w}\right]T + S_w(l_1, l_{n+1}) \qquad (\mathit{from\ \ theorem}\ 4.1) \\
&\leq \left[n + (n-1)\frac{2+\delta_w}{R_w}\right]T + S_w(j, i) \qquad (\mathit{putting}\ \ l_1 = j\ \ \mathit{and}\ \ l_{n+1} = i)
\end{aligned}
$$

The final result follows immediately. □

The expression for D^{CORR} derived above consists of two main terms. The first term is a constant independent of the cell index. If we observe the second term carefully, we realize that it is no other than the delay encountered by the i^{th} cell at CORR server with a cycle time T and a minimum rate of service R_w. Hence, end-to-end delay reduces to the sum of the delay encountered in a single node system and a constant. By substituting the delay bounds for the single-node system derived in the last section we can enumerate end-to-end delay in a multi-node system for different traffic envelopes.

4.2 Fairness Analysis

In the last section we analyzed some of the worst case behavior of the system. In the worst case analysis it is assumed that the system is fully loaded and each connection is served at the minimum guaranteed rate. However, that is often not the case. In a work conserving server, when the system is not fully loaded the spare capacity can be used by the busy sessions to achieve better performance. One of the important performance metric of a work conserving scheduler is the fairness of the system. That is how fair is the scheduler in distributing the excess capacity among the active connections.

Let us define by $D_p(t)$, the number of packets of connection p transmitted during [0,t). We define the normalized work received by a connection p as $w_p(t) = D_p(t)/R_p$. Accordingly, $w_p(t_1, t_2) = w_p(t_2) - w_p(t_1)$, where $t_1 \leq t_2$ is the normalized service received by connection p during (t_1, t_2).

In an ideally fair system, the normalized service received by different connections in their busy state increase at the same rate. For sessions that are not busy at t, normalized service stays constant. If two connections p and q are both in their busy period during $[t_1, t_2)$, we can easily show that $w_p(t_1, t_2) = w_q(t_1, t_2)$.

Unfortunately, the notion of ideal fairness is only applicable to hypothetical fluid flow model. In a real packet network, a complete packet from one connection has to be transmitted before service is shifted to another connections. Therefore, it is not possible to satisfy equality of normalized rate of services for all busy sessions at all times. However, it is possible to keep the normalized services received by different connections close to each other. The *Packet-by-Packet Generalized Processor Sharing* (PGPS) and the *Self-Clocked-Fair-Queuing* (SFQ) are close approximations to *ideal-fair-queuing* in the sense that they try to keep the normalized services received by busy sessions close to that of an ideal system. Unfortunately, the realization of PGPS and SFQ are quite complex. In the following we will show that the CORR scheduling is almost as fair as PGPS and SFQ, albeit its simplicity in terms of implementation. For reasons of simplicity we will assume that our sampling points coincide with the beginning of the allocation cycles only. If frame sizes are small, this approximation is quite reasonable.

Lemma 4.3 *If a connection p is in a busy period during the cycles c_1 through c_2, where $c_2 \geq c_1$, the amount of service received by the connection during $[c_1, c_2]$ is bounded by*

$$max\{0, \lfloor (c_2 - c_1)R_p - \delta_p \rfloor\} \leq D_p(c_1, c_2) \leq \lceil (c_2 - c_1)R_p + \delta_p \rceil$$

Proof: Follows directly from lemma 3.2.

Corollary 4.3 *If a connection p is in a busy period during the cycles c_1 through c_2, where $c_2 \geq c_1$, the amount of normalized service received by the connection during $[c_1, c_2]$ is bounded by*

$$max\left\{0, \frac{\lfloor (c_2 - c_1)R_p - \delta_p \rfloor}{R_p}\right\} \leq w_p(c_1, c_2) \leq \frac{\lceil (c_2 - c_1)R_p + \delta_p \rceil}{R_p}$$

Proof: Follows directly from lemma 4.3 and the definition of normalized service. □

Theorem 4.3 *If two connections p and q are in their busy periods during the cycles c_1 through c_2, where $c_2 \geq c_1$, then*

$$\Phi(p, q) = |w_p(c_1, c_2) - w_q(c_1, c_2)| \leq \frac{1 + \delta_p}{R_p} + \frac{1 + \delta_q}{R_q}$$

Proof: From the last corollary we get,

$$\begin{aligned}
\Phi(p, q) &= |w_p(c_1, c_2) - w_q(c_1, c_2)| \\
&\leq max\left\{\left|\frac{\lceil (c_2 - c_1)R_p + \delta_p \rceil}{R_p} - \frac{\lfloor (c_2 - c_1)R_q - \delta_q \rfloor}{R_q}\right|,\right. \\
&\qquad \left.\left|\frac{\lceil (c_2 - c_1)R_q + \delta_q \rceil}{R_q} - \frac{\lfloor (c_2 - c_1)R_p - \delta_p \rfloor}{R_p}\right|\right\} \\
&\leq max\left\{\left|\frac{\left\lceil \left([c_2 - c_1] + \frac{\delta_p}{R_p}\right) R_p \right\rceil}{R_p} - \frac{\left\lfloor \left([c_2 - c_1] - \frac{\delta_q}{R_q}\right) R_q \right\rfloor}{R_q}\right|,\right. \\
&\qquad \left.\left|\frac{\left\lceil \left([c_2 - c_1] + \frac{\delta_q}{R_q}\right) R_q \right\rceil}{R_q} - \frac{\left\lfloor \left([c_2 - c_1] - \frac{\delta_p}{R_p}\right) R_p \right\rfloor}{R_q}\right|\right\} \\
&\leq max\left\{\left|\left([c_2 - c_1] + \frac{1 + \delta_p}{R_p}\right) - \left([c_2 - c_1] - \frac{1 + \delta_q}{R_q}\right)\right|\right. \\
&\qquad \left.\left|\left([c_2 - c_1] + \frac{1 + \delta_q}{R_q}\right) - \left([c_2 - c_1] - \frac{1 + \delta_p}{R_p}\right)\right|\right\} \\
&\leq max\left\{\frac{1 + \delta_p}{R_p} + \frac{1 + \delta_q}{R_q}, \frac{1 + \delta_q}{R_q} + \frac{1 + \delta_p}{R_p}\right\}
\end{aligned}$$

This completes the proof. □

To compare fairness of CORR with other schemes, such as PGPS and SFQ, we can use $\Phi(p,q)$ as the performance metric. As discussed earlier, $\Phi(p,q)$ is the absolute difference in normalized work received by two sessions over a time period where both of them were busy. We proved earlier that if our sample points are at the beginning of the allocation cycles,

$$\Phi^{CORR}(p,q) \leq \frac{1+\delta_p}{R_p} + \frac{1+\delta_q}{R_q}.$$

Under the same scenario described in the last section, it can be proved that in the SFQ scheme the following holds at all times,

$$\Phi^{SFQ}(p,q) \leq \frac{1}{R_p} + \frac{1}{R_q}$$

Due to difference in the definition of busy periods in PGPS similar result is difficult derive. However, Golestani [6] has shown that the maximum permissible service disparity between a pair of busy connections in the SFQ scheme is never more than two times the corresponding figure for any real queueing scheme. This proves that,

$$\Phi^{PGPS}(p,q) \geq \frac{1}{2}\Phi^{SFQ}(p,q)$$

Note that, $0 \leq \delta_i \leq 1$ for all connection i. Hence, the fairness index of CORR is within two times that of SFQ and at most four times that of any other queuing discipline including PGPS.

5 Numerical Results

In this section we compare the performance of CORR with PGPS and SG using a number of MPEG coded video traces with widely varying traffic characteristics. We used four 320×240 video clips (see Table 1), each approximately 10 minutes long in our study. The first video is an excerpt from a very fast scene changing basketball game. The second clip is a music video (MTV) of the rock group REM. It is composed of rapidly changing scenes in tune with the song. The third sequence is a clip from CNN Headline news where the scene alternates between the anchor reading news and different news clips. The last one is a lecture video with scenes alternating between the speaker talking and the viewgraphs. The only moving objects here are the speaker's head and hands. Figure 4 plots frame sizes against frame number (equivalently time) for all four sequences for an appreciation of the burstiness in different sequences. In all traces, frames are sequenced as IBBPBB and frame rate is 30 frames/sec. Observe that, in terms of the size of GoP [5] and that of an average frame, BasketBall and Lecture videos are at the two extremes (the largest and the smallest, respectively), with the other two videos in between.

[5] The repeating sequence (IBBPBB in this case) is called a GoP or Group of Pictures.

Traces	Type of Frame	Maximum Frame Size	Minimum Frame Size	Average Frame Size	Variation (Std. Dev)
Basketball	I	41912	8640	26369.14	4672.12
	P	40128	6400	15570.31	1846.77
	B	35648	4288	11137.18	2856.51
	Avg. Frame			14414.69	
	GoP	215232	59008	86488.12	6284.27
MTV Video	I	34496	8512	21770.88	4808.31
	P	28544	9152	15833.20	1437.66
	B	32640	4608	12211.39	3123.87
	Avg. Frame			14408.27	
	GoP	146304	65472	86443.91	4017.07
News Clip	I	43200	17144	27728.71	3537.86
	P	20160	11392	16025.23	547.62
	B	26304	6208	10643.04	2691.66
	Avg. Frame			14387.68	
	GoP	118464	81664	86326.10	839.50
Lecture	I	13504	5312	10956.67	1498.87
	P	13312	2048	4673.82	642.53
	B	6592	768	3292.95	617.45
	Avg. Frame			4800.38	
	GoP	33984	23424	28802.33	555.95

Table 1: Characteristics of the MPEG traces. Size is in bytes and frame sequence is IBBPBB.

Results presented in the rest of the section demonstrate (1) CORR achieves high utilization irrespective of the shaping mechanism used, (2) when used in conjunction with composite shapers CORR can exploit the precision in traffic characterization and can achieve even higher utilization.

CORR and PGPS

PGPS is a packet-by-packet implementation of the fair queuing mechanism. In PGPS, incoming cells from different connections are buffered in a sorted priority queue and is served in the order in which they would leave the server in an ideal fair queueing system. The departure times of cells in the ideal system is enumerated by simulating a reference fluid flow model. Simulation of the reference system and maintenance of the priority queue are both quite expensive operations. Hence, the implementation of the PGPS scheme in a high-speed switch is difficult, to say the least. Nevertheless, we compare CORR with PGPS to show how it fares against an ideal scheme.

In the results presented below we consider a system configuration where all connections from the source to the sink pass through five switching nodes connected via T3 (45 Mb/s) links (see figure 5). We also assume that each switch is equipped with 2000 cell buffers on each output port. As shown in figure 5 traffic from a source passes through shaper(s) before entering the network. For the results

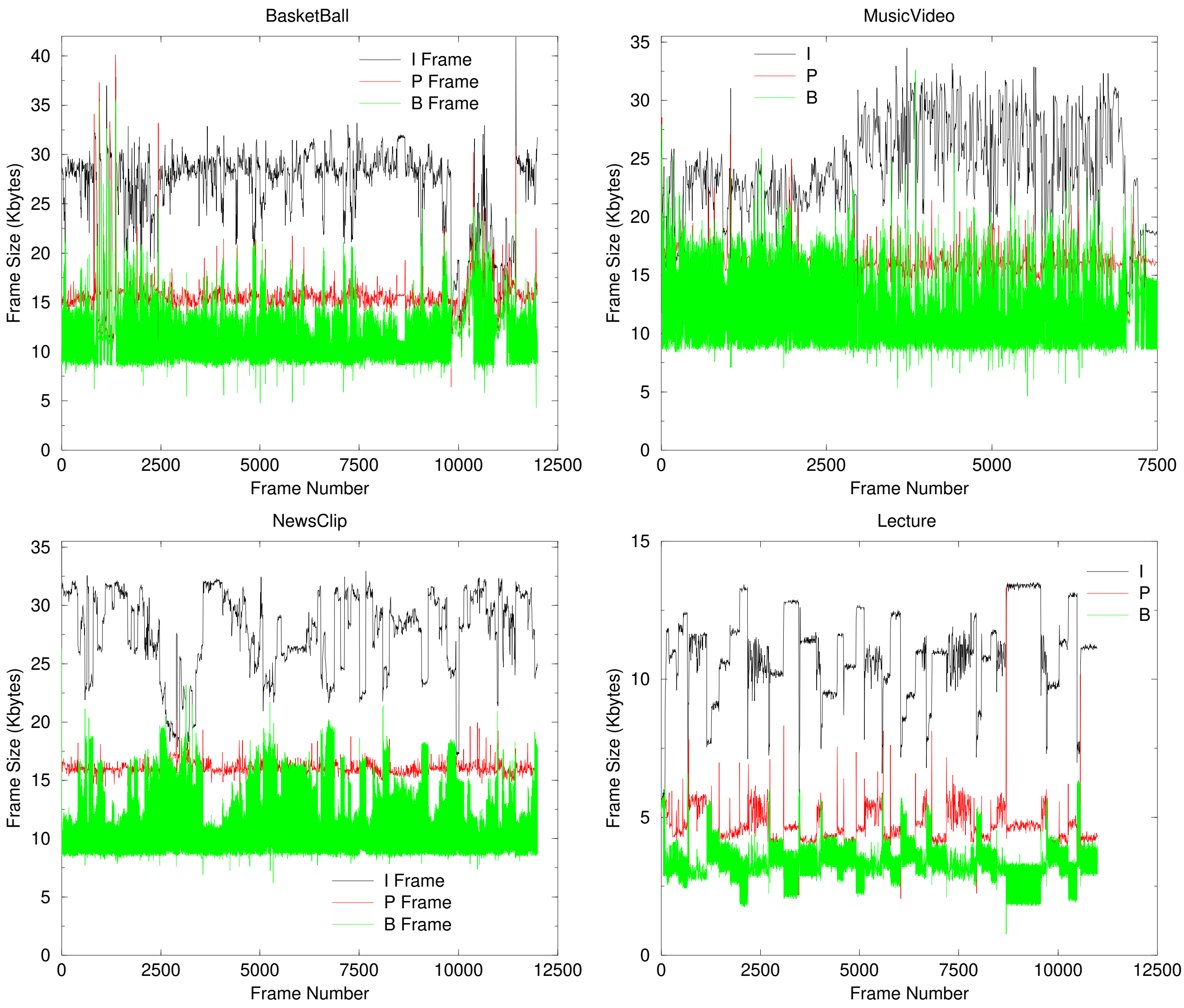

Figure 4: MPEG compressed video traces. Frame sequence is IBBPBB.

reported in this section we assume that the shapers used at the network entry point employ leaky bucket shaping mechanism.

In figures 6,7,8, and 9 we compare the number of connections admitted by CORR and PGPS for different traffic sources, end-to-end delay requirements, and shaper configurations. In order to make the comparison fair, we have chosen the leaky bucket parameters for different sources in such a way that it maximizes the number of connections admitted by PGPS given the end-to-end delay, and buffer sizes in the switch and the shaper. This is a tricky and time consuming process (we use

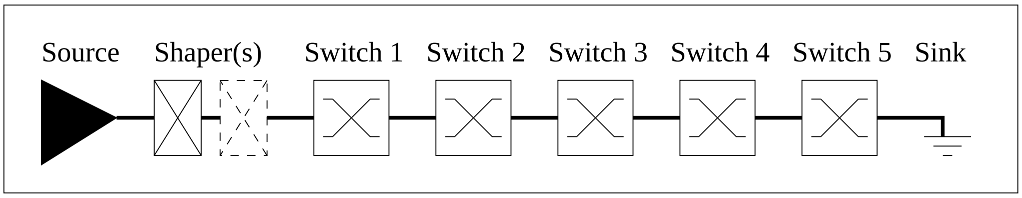

Figure 5: Experimentation model of the network. All the links are 45 Mbps. Both the shapers are used for CORR. Only one shaper is used for other scheduling disciplines.

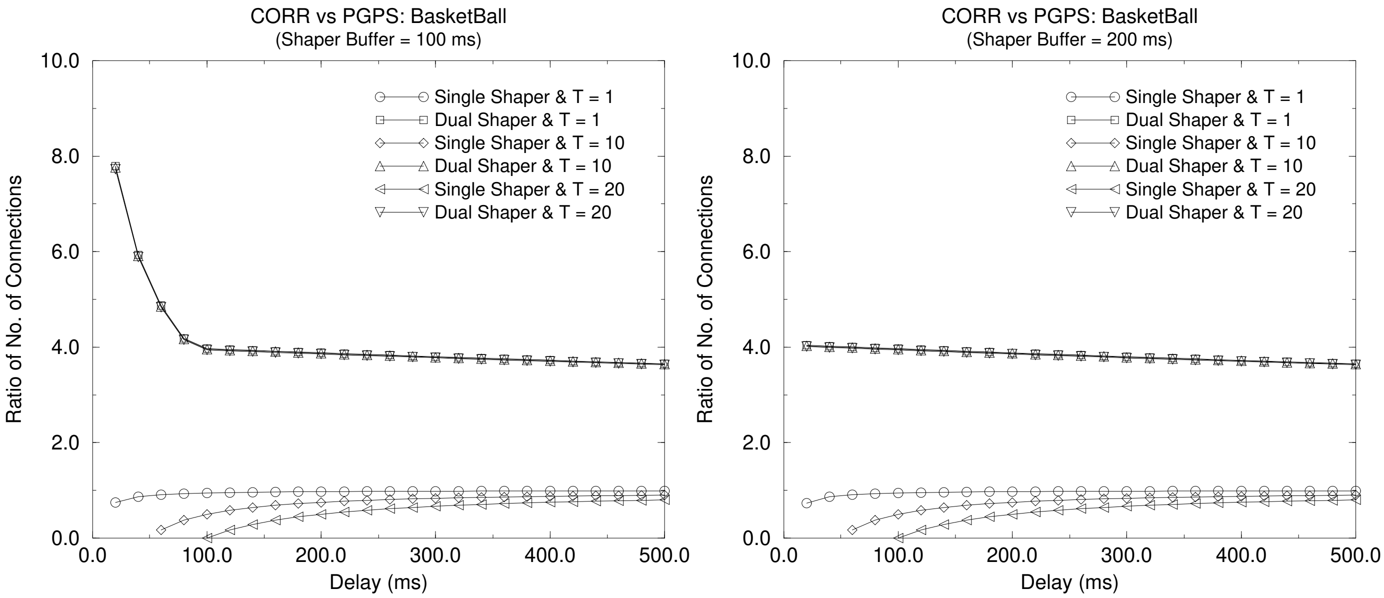

Figure 6: Relative performance of CORR and PGPS on BasketBall video with different shaper buffers.

linear programming techniques) and requires a full scan of the entire traffic trace for each source. For a description of this procedure please refer to [9]. We have plotted the ratio of the number of connections admitted by CORR used in conjunction with single and dual leaky bucket shapers with that of PGPS under this best case scenario.

In figure 6 we have compared the number of connections admitted by CORR and PGPS for the BasketBall video. The two sets of graphs correspond to two different sizes of shaper buffers, 100ms and 200ms in this case. Quite expectedly PGPS outperforms CORR for delay bounds less than 150ms when CORR is used in conjuction with a single leaky bucket. Note however that the ratio of the number of connections is very close to 1 and approaches 1 for higher delay bounds. This is due to the fixed frame synchronization overhead in CORR that is more conspicuous in low delay regions. The effect of this fixed delay fades for higher end-to-end delays. We also observe that the lower the frame size(T), the more competitive CORR is to PGPS in terms of number of connections admitted.

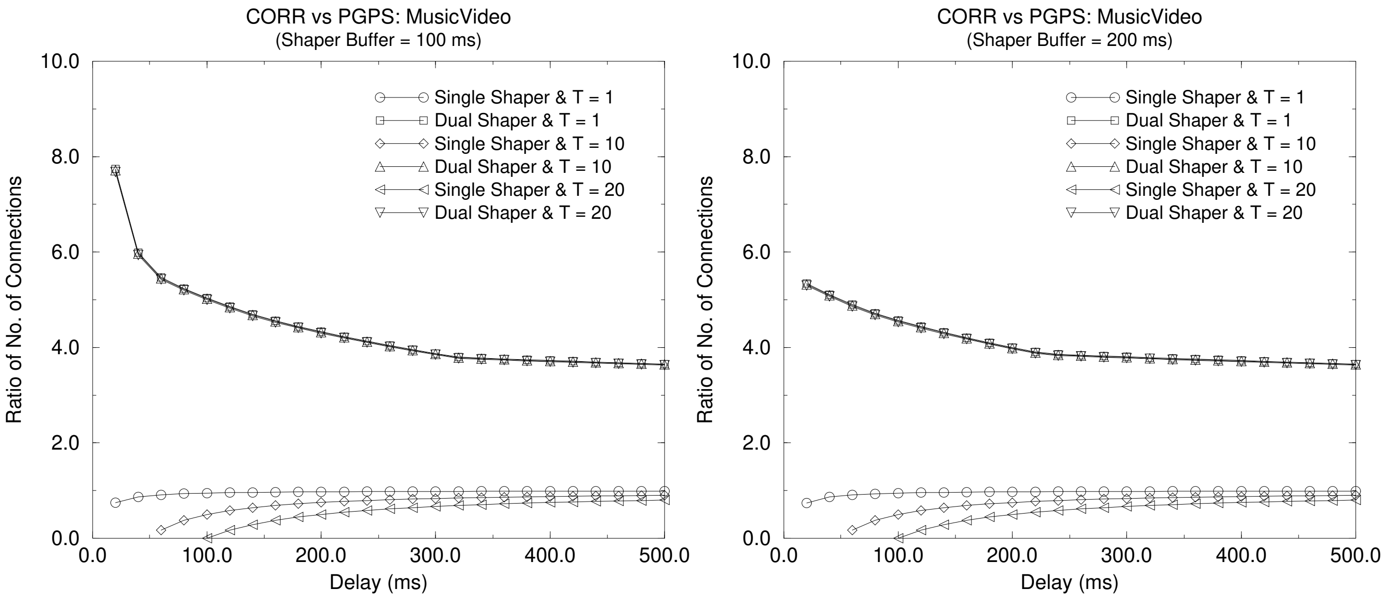

Figure 7: Relative performance of CORR and PGPS on MusicVideo with different shaper buffers.

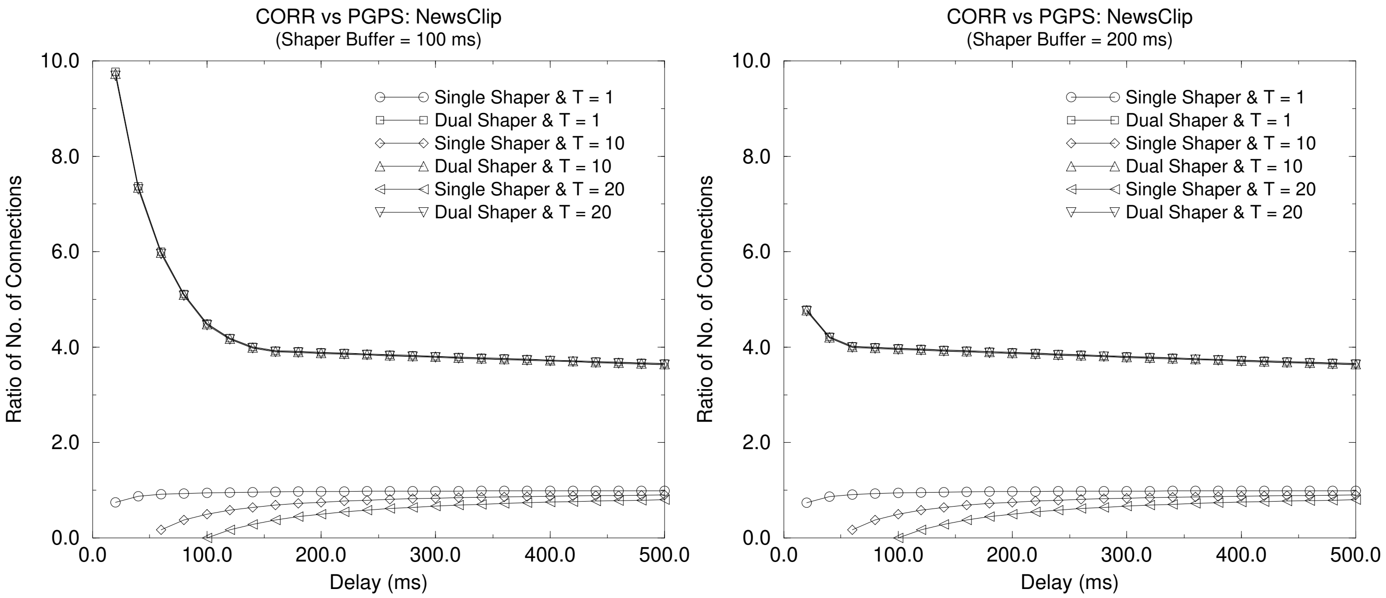

Figure 8: Relative performance of CORR and PGPS on NewsClip video with different shaper buffers.

For traditional frame based (or round robin) scheduling a small size frame (short cycle time) leads to large bandwidth allocation granularity and hence is not useful in practice. CORR however does not suffer from this shortcoming and can use very small cycle time.

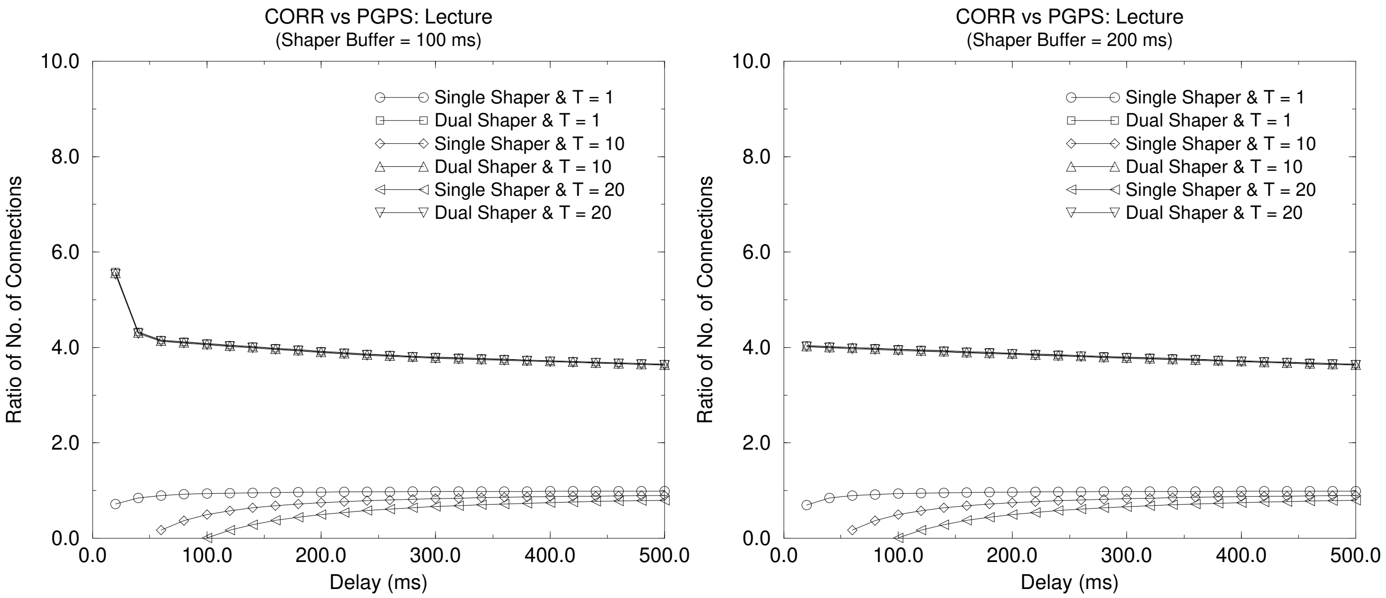

Figure 9: Relative performance of CORR and PGPS on Lecture video with different shaper buffers.

When used in conjunction with dual leaky bucket shapers, CORR outperforms PGPS irrespective of delay bounds, shaper buffer sizes, and cycle times. PGPS cannot take advantage of multi-rate shaping. So, no matter what the delay bound is, the number of connections admitted by PGPS depends only on the leaky bucket parameters. CORR on the other hand can choose the lowest rate of service sufficient to guarantee the required end-to-end delay bound. The benefits of this flexibility is reflected in figure 6 where the connections admitted by CORR outnumbers the connections admitted by PGPS by more than 4:1 margin. For a shaper buffer size of 100ms the ratio of number of connections admitted by CORR and that by PGPS is around 8 for end-to-end delay bound of 20ms. The ratio falls sharply and flattens out at around 4 for end-to-end delay of 100ms or more. The higher gain seen by CORR for lower end-to-end delay budget can be explained by its by effective use of shaper and switch buffers. Unlike PGPS, CORR uses a much lower service rate, just enough to guarantee the required end-to-end delay. It effectively uses the buffers in the switches and the shaper to smooth out the burstiness in the traffic. As delay budget increases, PGPS uses a lower rate of service. Consequently, gain seen by CORR decreases and eventually stabilizes around 4.

A trend similar to the one seen in figure 6 is observed in figures 7,8,and 9. In all cases PGPS outperforms CORR (used in conjuction with single leaky bucket) for low end-to-end delay. For higher delays the ratio of number of connections admitted by CORR and PGPS is practically 1. When used in conjunction with two leaky buckets, CORR outperforms PGPS by a margin higher than 4:1. We also observe that gain seen by CORR when used with two leaky buckets is higher for lower delays. Careful observation reveals that gain also depends on the size of the shaper buffer used and the traffic pattern of the source. The smaller is the shaper buffer, the higher is the gain.

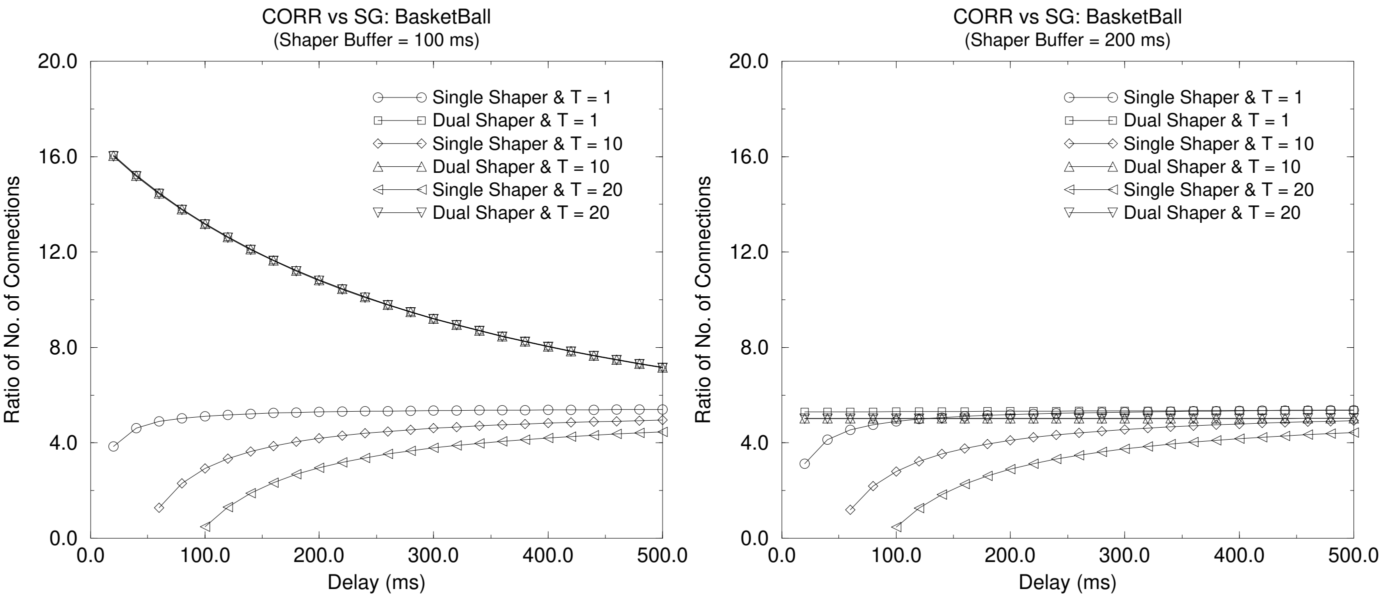

Figure 10: Relative performance of CORR and SG on BasketBall video with different shaper buffers.

Similarly, the more bursty is the traffic, the higher is the gain. This is due to the fact that unlike PGPS, CORR can exploit the multi-rate shaping of traffic by choosing a rate of service that is lower than the peak rate of traffic generation. The gain due to this flexibility is more conspicuous when delay budget is tighter. It is also evident from the graphs that the effect of cycle time has minimal impact on the performance of CORR, especially for higher delay budget or when traffic is shaped using multi-rate shapers.

CORR and SG

The purpose of these comparison is to demonstrate that (unlike PGPS and SG) CORR is not constrained to be used in conjunction with a specific traffic shaping mechanism. It works equally well with moving window shapers as it does with leaky bucket shapers. In the last section we have seen how the performance of CORR compares with that of PGPS for leaky bucket shapers. In this section we compare the performance of CORR with that of SG for moving window shapers. As PGPS is constrained to work with only leaky bucket shapers, the use of SG is restricted to moving window [6] shapers only.

We use the same system configuration as shown in figure 5 and described in the last section. As in the case of PGPS, we choose moving window parameters that maximizes the number of connections admitted by SG given the traffic trace, size of the shaper buffer, and the end-to-end delay. Figures 10,

[6] Although in the original description of SG does not use moving window shapers to smooth the traffic, the (r, T) smoothness defined in [5] can be exactly modelled by a moving window with $m = r$ and $w = T$.

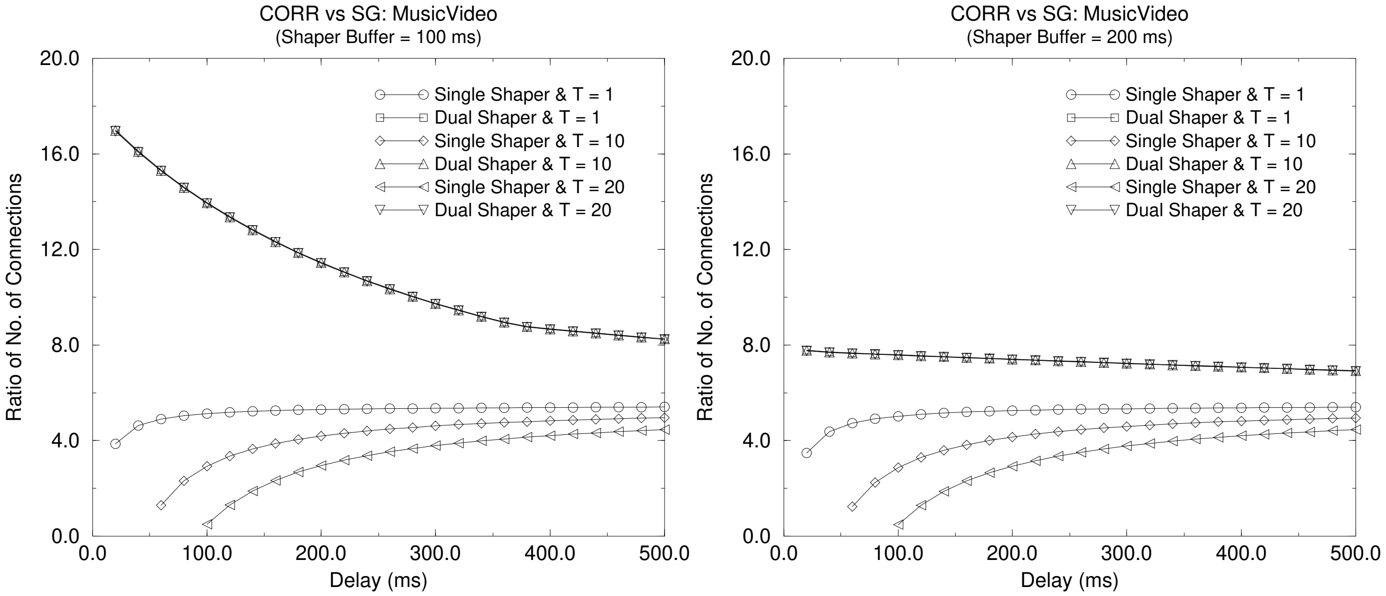

Figure 11: Relative performance of CORR and SG on MusicVideo with different shaper buffers.

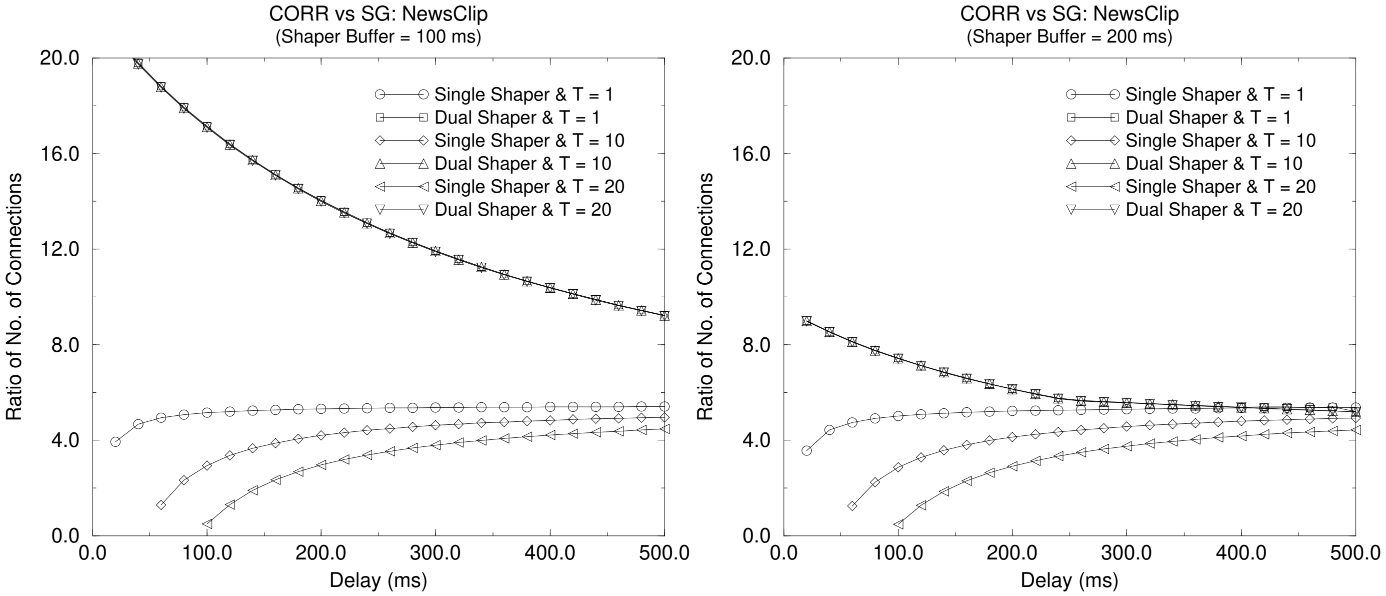

Figure 12: Relative performance of CORR and SG on NewsClip video with different shaper buffers.

11, 12 and 13 plot the ratio of the number of connections admitted by CORR and SG for different end-to-end delay and shaper configurations.

In general we observe similar trends in the plots as in the previous section, except that CORR

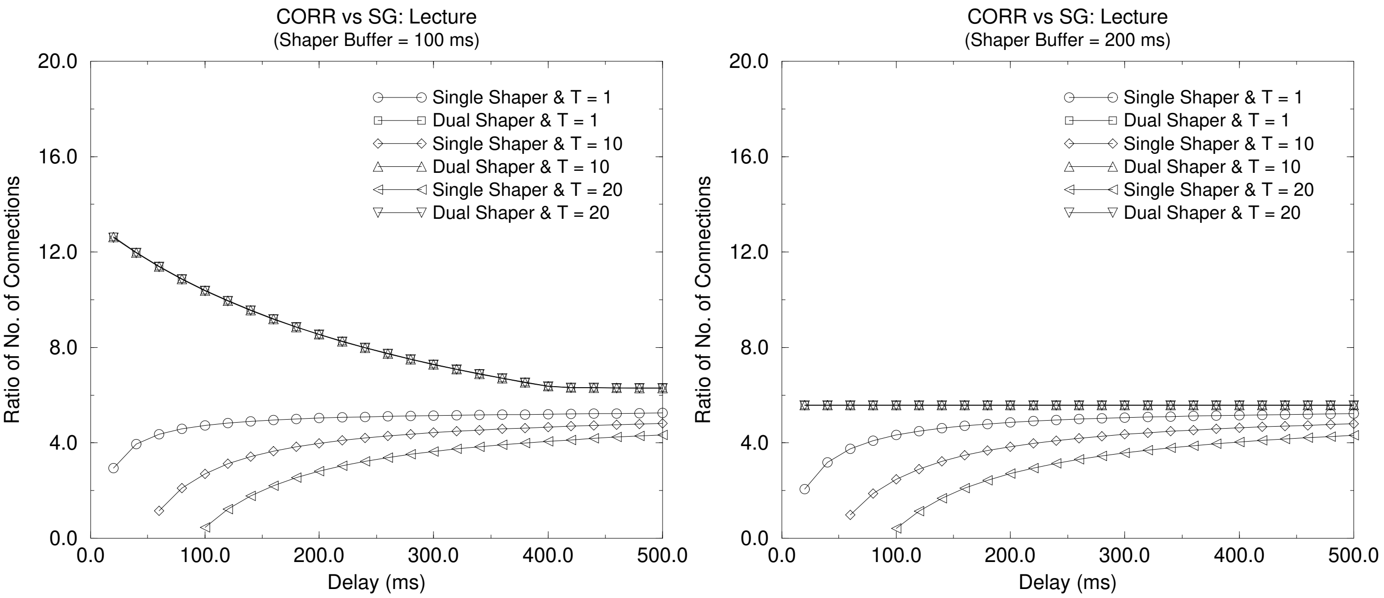

Figure 13: Relative performance of CORR and SG on Lecture video with different shaper buffers.

outperforms SG in most cases even when used in conjuction with a single moving window. The observations can be summarized as follows:

When used with single moving window, CORR outperforms SG for all delay requirements and all shaper configurations except for a few instances requiring low end-to-end delay.

When used with single moving window, CORR performs better when the cycle time is smaller. In all frame based or round robin scheduler, the smaller the frame size (or cycle time) better is the delay performance. The difference in the case of CORR is that a lower cycle time does not constrain the bandwidth allocation granularity.

CORR, used in conjuction with two moving windows, always configuration SG. The performance gap is higher for smaller shaper buffer, smaller delay budget, and more bursty traffic sources. When used with dual moving windows, cycle time has very little impact on the number of connections admitted by CORR.

Smaller is the end-to-end delay requirement higher is the gain seen by CORR when used with two moving windows. However, when used with single moving window, the gain seen by CORR is lower for tighter delay requirements and increases as the delay requirements loosen.

The results presented in this section demonstrate that despite its simplicity (1) CORR is very competitive with PGPS and almost always out performs SG when used in conjunction with simple shapers, (2) CORR performs equally well with leaky bucket as well as moving window shapers,

(3) when used in conjunction with multi-rate shapers CORR outperforms both PGPS and SG by significant margins. The results are consistent across traffic traces of wide variability and shows the effectiveness of CORR as a multiplexing mechanism.

6 Concluding Remarks

References

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

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

[3] A. Demers, S. Keshav, and S. Shenkar. Anslysis and Simulation of Fair Queuing Algorithm. In *Proceedings, SIGCOMM*, 1989.

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

[5] S. J. Golestani. Congestion Free Communication in High-Speed Packet Networks. *IEEE Transaction on Comminication*, 32(12), December 1991.

[6] S.J. Golestani. A Self-Clocked Fair Queuing Scheme for Broadband Applications. In *Proceedings, INFOCOM*, June 1993.

[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] D. Saha, S. Mukherjee, and S. K. Tripathi. Carry-over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks. In *Proceedings, INFOCOM. Extended version available as Technical Report CS-TR-3658/UMIACS-TR-96-45, University of Maryland*, March 1996.

[9] D. Saha, S. Mukherjee, and S.K. Tripathi. Multirate Scheduling of VBR Video Traffic in ATM Networks. *Technical Report CS-TR-3657/UMIACS-TR-96-44, University of Maryland*, May 1996.

[10] M. Shreedhar and George Varghese. Efficient Fair Queuing using Deficit Round Robin. *IEEE/ACM Transsactions on Networking*, 4(3), June 1996.

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

A Appendix: Shaping Mechanisms

A.1 Simple Shapers

Several shaping mechanisms defining various types of shaping envelopes have been proposed in the literature. The most popular among them are *leaky bucket*, and *moving window*. Following we briefly describe their working principles and the shaping envelope they enforce on a traffic flow.

Leaky Bucket Shapers: A leaky bucket regulator consists of token counter and a timer. The counter is incremented by one each t time and can reach a maximum value b. A cell is admitted into the system/network if and only if the counter is positive. Each time a cell is admitted, the counter is decremented by one. The traffic generated by the leaky bucket regulator consists of a burst of upto b cells followed by a steady stream cells with a minimum inter-cell time of t. The major attraction of the leaky bucket is its simplicity. A leaky bucket regulator can be implemented with two counters, one to implement the token counter and the other to implement the timer.

Moving Window Shapers: The moving window mechanism divides the time line into fixed size windows of length w. The number of arrivals in a window is limited to a maximum number m. Each cell is remembered for exactly one window width. That is, if we slide a window of size w on the time axis, the number of cells admitted within a window period would never exceed m irrespective of the position of the window. Hence, the worst case burst size in moving window never exceeds m. Since the departure time of each cell is remembered for a duration of one window period, the implementation complexity depends on m, the maximum number of cells admitted within a window period.

A.2 Composite Shapers

Simplicity is the main attraction of the shapers described in the last section. They all are quite easy to implement, and define traffic envelopes that are easily amenable to analytical treatment. Unfortunately, they are often too simple to capture the true characteristics of the real-life sources [9, 8]. All the shapers described in the previous section enforce a specific rate constraint on a source, typically a declared peak or average rate. However, most applications generate inherently bursty traffic. Hence, enforcing an average rate results in a higher delay in the shaper buffer, and a peak rate enforcement leads to an over allocation of system resources, and consequently lower utilization of the system. To alleviate this problem, or in other words, to enforce a shaping envelope that is close to the original shape of the traffic, yet simple to specify and monitor, we can use multiple shapers arranged in series. That is, we can choose multiple leaky buckets or moving windows with different parameters enforcing different rate constraints over different time intervals. Composite shapers provide us with a much richer and larger set of traffic envelopes with marginal increase in complexity.

In the rest of the section we derive the exact shapes of the traffic envelopes when multiple leaky buckets and moving windows regulators are arranged in cascade. Our objective is to determine the worst case bursty behavior of the traffic generated by the shaper. That is, traffic is generated by the shaper at the maximum rate permitted by the shaping envelope. These results are used in section 4 to find the worst case bounds on end-to-end delay.

Multiple Leaky Buckets

The shaping envelope defined by a composite leaky bucket is the intersection of the shaping envelopes of constituent leaky buckets. In figure 14 a composite leaky bucket consisting of leaky buckets LB_1, LB_2, LB_3 and LB_4 is shown [7]. The shaping envelope is the thick line. The exact shape of the envelope depends on the number of shapers and the associated parameters. Inappropriate choice of shaper parameters may give rise to redundant components which may not have any role in defining the shaping function. For example, LB_4 is a redundant component in the composite shaper shown in figure 14. We call a set of leaky buckets an *essential set* if none of the buckets is redundant.

Let us consider n leaky buckets (b_i, t_i), $i = 1, 2, \ldots, n$. Without loss of generality we number the buckets in such a fashion so that $t_i \geq t_j$, for $i < j$. We can show that if these leaky buckets from an *essential set* then $b_i \geq b_j$, for $i < j$.

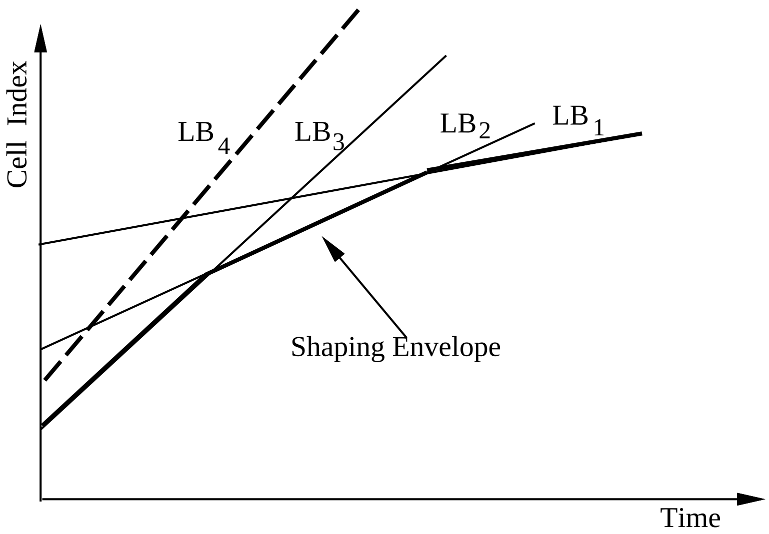

Figure 14: Shaping with multiple leaky buckets.

Definition A.1 *An n-component composite leaky bucket is an essential set of n leaky buckets* $(b_1, t_1), (b_2, t_2), \ldots, (b$ *where* $b_i > b_j$ *and* $t_i > t_j$ *for* $i < j$. *For the purpose of mathematical convenience we assume that the n-component shaper includes another pseudo leaky bucket* (b_{n+1}, t_{n+1}) *where* $b_{n+1} = 0$ *and* $t_{n+1} = 0$.

[7]Precisely speaking the shaping function due to each leaky bucket is a burst followed by a stair case function. For ease of exposition we have approximated the stair case function by a straight line with the same slope. This simplification is only for the purpose of explanation. The results derived later takes the stair case function into consideration.

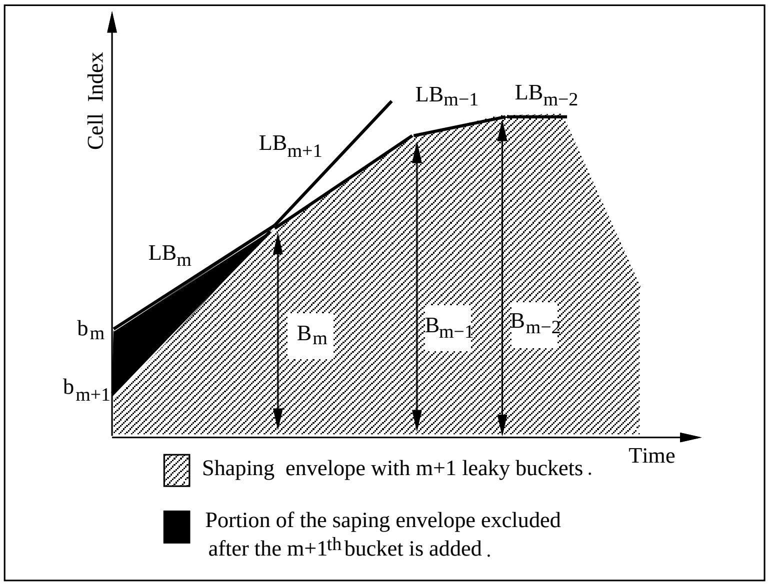

Figure 15: Departure function after adding $m+1_{th}$ bucket.

The following theorem determines the shaping function of a composite shaper consisting of an essential set of leaky buckets.

Theorem A.1 *Consider an n-component composite leaky bucket. Define*

$$B_k = \begin{cases} \infty & k = 0, \\ \left\lfloor \dfrac{b_k t_k - b_{k+1} t_{k+1}}{t_k - t_{k+1}} \right\rfloor & k = 1, 2, \ldots, n, \\ 0 & k = n+1. \end{cases}$$

Then the departure time of the i^{th} cell from the composite shaper, denoted by $a(i)$ is,

$$a(i) = \sum_{k=1}^{n+1} (i - b_k + 1)\ t_k\ [U(i - B_k) - U(i - B_{k-1})], \quad i = 0, 1, \ldots, \infty$$

where $U(x)$ is the unit step function defined as,

$$U(x) = \begin{cases} 0 & x < 0, \\ 1 & x \geq 0. \end{cases}$$

Proof: We will prove this theorem by induction.

Base Case: For $n = 1$, we have $B_0 = \infty$, $B_1 = b_1$, and $B_2 = 0$. Therefore,

$$a(i) = (i - b_1 + 1)\ t_1\ U(i - b_1).$$

This allows a burst of size b_1 to depart at time 0 and a cell after every t_1 henceforth. Clearly, the traffic envelope defined by the shaping function conforms to the traffic generated by a leaky bucket with parameters b_1 and t_1. Hence the hypothesis holds in the base case.

Inductive Hypothesis: Assume that the premise holds for all $n \leq m$. To prove that it holds for all n, we need to show that it holds for $n = m + 1$.

Figure 15 shows the cumulative departure function before and after the addition of the $(m+1)^{th}$ bucket. Since the set of buckets constitute an essential set, $b_{i+1} < b_i$ for $i = 1, 2, \ldots m$. Therefore, the effect of $(m+1)^{th}$ bucket is observed from time 0 to the time when the shaping function of bucket $m+1$ intersects the composite shaping function before the addition of the $(m+1)^{th}$ bucket (see figure 15). We observe that this cross over point is really the point of intersection between the shaping functions of the m^{th} and the $(m+1)^{th}$ leaky buckets. Using simple geometry we can find that these shaping functions intersects at the departure instant of the $\lfloor (b_m t_m - b_{m+1} t_{m+1})/(t_m - t_{m+1}) \rfloor$th cell, which, by definition, is B_m.

In other words, the bucket $m+1$ excludes the segment marked by the solid shadow from the shaping envelope as shown in figure 15. The new segment of the shaping function can be defined as,

$$(i - b_{m+1} + 1)\ t_{m+1}\ [U(i - B_{m+1}) - U(i - B_m)]$$

After some algebraic manipulation, we can write the entire shaping function as,

$$\begin{aligned}
a(i) &= \sum_{k=1}^{m+1} (i - b_k + 1)\ t_k\ \ [U(i - B_k) - U(i - B_{k-1})] \\
&\quad + (i - b_{m+1} + 1)\ t_m\ [U(i - B_{m+1}) - U(i - B_m)] \\
&= \sum_{k=1}^{m+2} (i - b_k + 1)\ t_k\ \ [U(i - B_k) - U(i - B_{k-1})]
\end{aligned}$$

This completes the proof. □

Multiple Moving Windows

A composite moving window shaper smooths the traffic over multiple time windows. Figure 16 shows the shaping envelope of a composite shaper consisting of moving windows $MW_1 = (w_1, m_1)$, $MW_2 = (w_2, m_2)$, $MW_3 = (w_3, m_3)$, where $w_1 = 3 \times w_2$, $w_2 = 4 \times w_3$ and $m_1 = 2 \times m_2$, $m_2 = 2 \times m_3$. The shaping envelope shown in figure 16 captures the worst case bursty behavior of the traffic generated by a composite moving window. The worst case occurs when s burst of cells are generated at the earliest instant satisfying the regulator constraints. In this particular example traffic is shaped over three different time windows. The first moving window limits the number of cells dispatched in any time window of size w_1 to m_1. However, MW_1 does not impose any restriction on how these m_1 cells are dispatched. In the worst case, they may be dispatched as a single burst of size m_1.

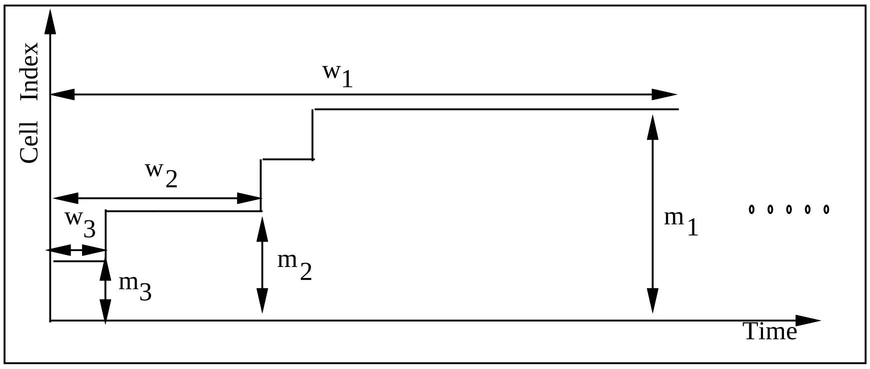

Figure 16: Shaping envelope of a composite moving window.

The moving window MW_2 determines the burst size distribution within w_1. The window w_1 can be broken down into three windows of duration w_2 each. The maximum amount of traffic that can be dispatched during any time interval of w_2 is limited to m_2 by MW_2. Hence, m_2 cells are dispatched in each of the first two w_2 intervals. Since $m_1 = 2 \times m_2$ no cells are dispatched in the third w_2 window to satisfy the constraint imposed by the first moving window. Similarly, $w_2 = 4 \times w_3$, but $m_2 = 2 \times m_3$. Hence, m_3 cells are dispatched in each of the first two w_3 windows within a w_2 window. In the remaining two w_3 windows no cells are dispatched to satisfy the constraint imposed by MW_2. In the following, we formally define a composite moving window shaper and characterize its traffic envelope.

Definition A.2 *An n-component composite moving window shaper consists of n simple moving windows (w_k, m_k), $k = 1, \ldots, n$, where $w_i \geq w_j$, $m_i \geq m_j$, and $m_i/w_i \leq m_j/w_j$, for $1 \leq i < j \leq n$. For the sake of mathematical convenience we assume that an n-component composite shaper also include another pseudo moving window (m_0, w_0) such that $m_0/m_1 = 0$. We also assume for the simplicity of exposition that m_{i+1} divides m_i and w_{i+1} divides w_i, for $i = 1, 2, \ldots, n-1$.*

Theorem A.2 *Consider an n-component moving window shaper. If $a(i)$ is the departure time of the i^{th} cell from the shaper then,*

$$a(i) = \sum_{k=1}^{n} \left(\left\lfloor \frac{i}{m_k} \right\rfloor - \left\lfloor \frac{i}{m_{k-1}} \right\rfloor \frac{m_{k-1}}{m_k} \right) w_k, \qquad i = 0, 1, \ldots, \infty$$

Proof: We will prove this by induction.

Base Case: For $n = 1$, we have $a(i) = \left\lfloor \frac{i}{m_1} \right\rfloor w_1$. This means that a burst of size m_1 appears at time kw_1, $k = 0, 1, \ldots, \infty$. Clearly, this represents the shaping function due to a single moving window with parameters (w_1, m_1). Hence, the premise holds in the base case.

Inductive Hypothesis: Assume that the premise holds for all $n \leq l$. To prove that it holds for all n, we need to show that it holds for $n = l + 1$.

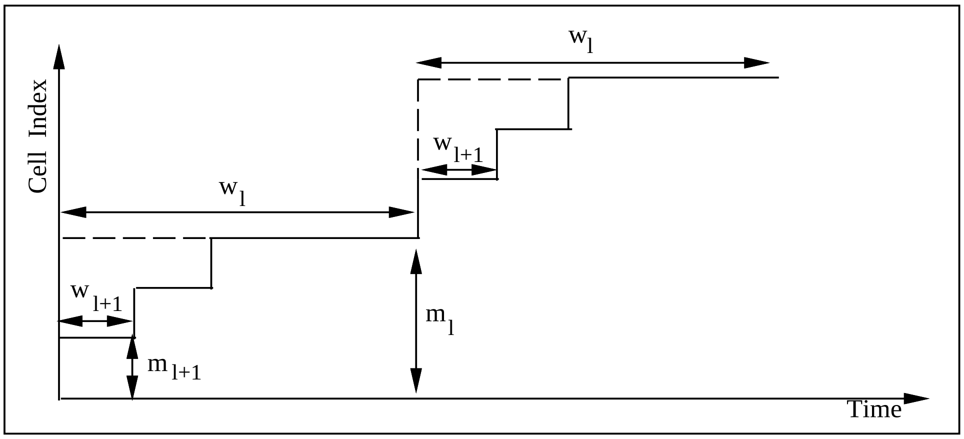

Figure 17: Shaping envelope after adding the $(l+1)^{th}$ moving window.

Consider the effect of adding $(l+1)^{th}$ moving window. In the worst case, the burst always comes at the beginning of a window. Therefore, as shown in figure 17, bursts of size m_l cells appear at the beginning of each window of length w_l, for m_{l-1}/m_l windows. Now, from the hypothesis, the arrival time of the i^{th} cell is given by,

$$a(i) = \sum_{k=1}^{l} \left(\left\lfloor \frac{i}{m_k} \right\rfloor - \left\lfloor \frac{i}{m_{k-1}} \right\rfloor \frac{m_{k-1}}{m_k} \right) w_k.$$

If a new shaper (w_{l+1}, m_{l+1}) is added, the burst appearing at the beginning of each w_l window will spread out into m_l/m_{l+1} bursts of size m_{l+1} each and separated by w_{l+1} as shown in figure 17. Due this spreading out of the bursts, the arrival time of the i^{th} cell will be postponed by,

$$\left(\left\lfloor \frac{i}{m_{k+1}} \right\rfloor - \left\lfloor \frac{i}{m_k} \right\rfloor \frac{m_k}{m_{k+1}} \right).$$

Hence the arrival time of the i^{th} cell after the addition of the $(l+1)^{th}$ moving window is,

$$\begin{aligned} a(i) &= \sum_{k=1}^{l} \left(\left\lfloor \frac{i}{m_k} \right\rfloor - \left\lfloor \frac{i}{m_{k-1}} \right\rfloor \frac{m_{k-1}}{m_k} \right) w_k \\ &\quad + \left(\left\lfloor \frac{i}{m_{k+1}} \right\rfloor - \left\lfloor \frac{i}{m_k} \right\rfloor \frac{m_k}{m_{k+1}} \right) \\ &= \sum_{k=1}^{l+1} \left(\left\lfloor \frac{i}{m_k} \right\rfloor - \left\lfloor \frac{i}{m_{k-1}} \right\rfloor \frac{m_{k-1}}{m_k} \right) w_k. \end{aligned}$$