An Efficient, Deadline Credit-based Transport Scheme for Prerecorded, Semi-Soft Continuous Media Applications

Zoe Antoniou
Nokia Research Center
5 Wayside Road
Burlington, MA, 01803, USA
zoe.antoniou@nokia.com

Ioannis Stavrakakis
Department of Informatics
University of Athens
Athens, 15784, Greece
istavrak@di.uoa.gr

Abstract

In this paper an efficient scheme is proposed based on the introduced Deadline Credit-based (DC) policy. This scheme is appropriate for any prerecorded media but particularly relevant for prerecorded, semi-soft Continuous Media (CM) applications. Semi-soft are applications with very small initial delay tolerance and, thus, for which very small amount of content may be sent in advance. The proposed policy pushes content toward the end-user during the session by taking advantage of any bandwidth underutilisation periods, exploiting available storage and building up fairly a deadline credit to be consumed during periods of overutilisation. The scheduling policy is studied for the single hop case (applicable to the server of the content), as well as for the multihop case (applicable to the server and network nodes). The derived results demonstrate the ability of the proposed scheme to decrease the amount of required bandwidth (or equivalently induced losses) with respect to alternative schemes without requiring large initial delay which is not acceptable for semi-soft CM applications.

1 Introduction

It is well known that Variable Bit Rate (VBR) encoding achieves better quality in the representation of Continuous Media (CM) compared to Constant Bit Rate (CBR) encoding for the same amount of resulting information bits. Or, equivalently, VBR encoding results in a smaller amount of information for the same level of quality and, thus, it is preferred for the representation of media to be stored. Yet, in contrast to CBR encoding, the variable bit rate required for the timely reconstruction of VBR encoded media translates into a variable bandwidth requirement if the media is to be reconstructed immediately upon request subject only to a network propagation delay. This is particularly relevant in the case of hard real-time (live or prerecorded) CM applications. Thus, the effective transport of delay intolerant (hard real-time) CM applications relies solely on the availability of the necessary bandwidth resources, as dictated by the their stochastic rate characteristics.

The aforementioned stringent bandwidth requirements can be relaxed in the case of delay tolerant (referred to as soft) prerecorded CM applications by capitalising on the delay tolerance, the availability of the content in advance, the available network and end-user storage and the use of prefetching. If the delay tolerance and storage capacity are large then the bandwidth requirement becomes very low and ceases to be a practical issue. This can find application with very soft, prerecorded CM streams in the presence of ample storage.

Between the two extremes lies the case of semi-soft, prerecorded CM applications for which the initial delay tolerance is very small and, thus, no substantial amount of content can be sent in advance. Storage availability in this case may still play a significant role in reducing the bandwidth requirement by pushing content toward the end-user mainly *during* the transport session as opposed to prefetching *before* the session. This is possible by taking advantage of the bandwidth availability during low demand periods of the session which are typically present due to the stochastic nature of traffic.

The proposed scheme is applicable for transporting soft, prerecorded applications but it is particularly tailored to handle effectively the typically bandwidth demanding, semi-soft, prerecorded CM applications. As it will become clear later, the proposed scheme can potentially serve almost hard, real-time, CM applications or even hard real-time applications that can tolerate some minor quality degradations mainly in the beginning of the session. The proposed scheme capitalises on the *a priori* content availability and exploits the variability of VBR encoding, the bandwidth availability, as well as, the storage availability at the client end network nodes to send content ahead of playback time. The scheduling policy associated with proposed scheme is referred to as the Deadline Credit-based (DC) policy. It has the following properties:

(1) utilise all (if possible) of the available capacity by allocating bandwidth *dynamically* to a number of CM data streams.

(2) distribute the available bandwidth *fairly* between all competing CM streams.

1.1 Focus of this work

In this paper, the content to be streamed is assumed to be organised in units containing logical blocks of information referred to as Application Data Units (ADUs). These ADUs have an underlying periodicity associated with their generation and reconstruction process. In the case of video streams, bits generated over a video frame form a logical block of information (ADU) which is central to both the traffic generation process at the transmitter and the reconstruction process at the receiver. Such a role may be played by a smaller encoding unit than the video frame such as a macroblock [1].

ADUs can be mapped into a collection of network date units[1] such as an ATM cell or an IP packet. In order to guarantee a timely reconstruction at the receiver, complete ADUs (e.g. frames) should be delivered at a constant display rate. This implies that all bits of an ADU should be delivered in time for the reconstruction process subject to the buffer at the receiving end and the smoothing algorithm (if any). When network resources are reserved at peak demand, all cells of the ADU will be delivered on time, leading to a timely and error-free reconstruction at the receiver. When resources are over-allocated to achieve statistical multiplexing gain, timely and error-free reconstruction is not a trivial problem.

The requirement for timely delivery of information associated with such continuous media traffic suggests that deadlines could be defined and deadline-driven policies be designed. If the nature of an application requires that a complete ADU should be received before it can be processed by the receiver, then it would be reasonable to associate a common deadline with all the bytes (or cells, packets) in the same data unit and develop scheduling policies that aim to maximise the number of complete data units that are delivered by their respective due date. This approach is different from many traditional deadline-driven approaches in which every cell (or network level data unit, in general) is assigned its own deadline and the scheduling policy typically attempts to maximise the number of individual cells transmitted by their due date [2], [3].

In order to achieve its objectives the DC policy uses all of the available bandwidth to (i) serve the ADUs with upcoming deadlines and (ii) send ADUs in advance. If the bandwidth is just enough to send one ADU per period then the server will transmit at a rate which is the minimum necessary to guarantee no starvation at the receiver. If, however, the system is underutilised the server can transmit additional ADUs in advance. This will result in *building up* a deadline credit for the future ADUs which have not been transmitted yet. In other words, if a stream is served at a rate of one ADU per period, the server will have to transmit at this rate throughout the session to avoid any losses. If, however, it sends a number of ADUs in advance when the system is underutilised, it will be able to transmit at a lower rate or not at all when network resources are not available due to bandwidth demand increase at a later time. Building a deadline credit means increasing the *Additional Delay Tolerance* (ADT) of future unscheduled ADUs. This deadline credit can be *consumed* at a later time when the demand for resources may increase. The proposed algorithm accumulates deadline credit by exploiting the variability of VBR encoding. Building deadline credit is equivalent to workahead smoothing.

The DC policy aims to accumulate the deadline credits of all competing streams fairly which ensures that no stream can monopolise the available bandwidth at the expense of other applications. The *fairness criterion* is briefly introduced here and explained in more detail in a later section. A mechanism is necessary to determine the relative priority between competing data streams, so that resources are distributed in a fair manner. Such a resource distribution is defined to be fair as long as it induces the same ADU deadline violation probabilities for all streams. In addition, if resources permit the transmission of ADUs in advance, the resource distribution is defined as fair, as long as it results in the same deadline credit build-up probability for all streams. An alternative measure of fairness could be based on the number of bits (or bytes, cells). With this measure, the DC policy would aim to equalise the number of bits (or bytes, cells) prefetched or dropped between all streams. The drawback of such an approach is that it can result in highly unbalanced levels both in prefetched ADUs in the client buffers and dropped ADUs. This can occur when some connections have a larger number of bits per frame on average than other connections. However, such an approach would aim to fill all the available buffer space fully in times of underutilisation. This paper uses the fairness criterion based on ADUs.

[1] A network level data unit can be a byte, an ATM cell or a specific type of packet. In this paper, without loss of generality, the basic network level data unit is assumed to be a fixed size packet called cell.

The DC policy uses the actual configuration of server, client and intermediate proxy sites. When the demand for bandwidth is low, ADUs can be sent in advance to the downstream nodes in the process of accumulating deadline credit for the stream (assuming buffer availability). Naturally, the larger the available buffer space the higher is the potential for the deadline credit build up. Also, the more links the ADUs traverse in advance the closer to the destination they reach. Consequently, large buffers can be useful in forwarding as *many* ADUs as possible, as *close* to the destination as possible *in advance* in an attempt to exploit the available bandwidth when the demand is low. This way, congestion is expected to be alleviated or avoided at a later time when the demand may increase.

The DC policy for a single hop network case is introduced in section 2 showing the potential it offers for efficient utilisation of the available server capacity. Its extension to the multihop network in section 4 illustrates the potential it provides for an improved end-to-end support of CM applications. Sections 3 and 5 include results for the single hop and multihop cases respectively. Finally, section 6 presents some concluding remarks.

1.2 Overview of related approaches

This subsection presents an overview of related policies both in the area of network and application level QoS parameters and in the area of scheduling and transmitting prerecorded multimedia traffic.

The concept that to an application the end-to-end delays of its data units are more relevant performance measures than ones specified for individual packets or cells has not been widely addressed. In [4] the problem of scheduling packets with a *group* as opposed to an individual deadline is considered. Packets of the same group are assumed to belong to the same application data unit for which a deadline is set, and they are assumed to arrive over fixed time intervals (frames). The Frame-based Shortest Time to Extinction (F-STE) scheduling policy is considered at a multiplexer fed by multiple streams with group deadlines.

In [5] and [6] the idea of *group scheduling* is proposed. Specifically, consecutive packet arrivals in a flow are partitioned into groups and the same deadline (group priority) is assigned to every packet in a group. The authors present an end-to-end delay guarantee theorem for a network of guaranteed-deadline (GD) servers. The concept of a *burst* (or block) is, also, introduced to represent a sequence of packets (or cells) that carry an application data unit. It is described how a real time VBR service, with QoS parameters for block transfer delay and block loss rate, can be provided by integrating concepts on burst scheduling and ATM block transfer (ABT).

The above mentioned references relate to the idea of network level versus application level data unit scheduling. The following references discuss the idea of smoothing and prefetching VBR encoded streams and how it relates to the notion of deadline credit scheduling.

The problem of scheduling and transmitting prerecorded traffic and, in particular, video has been addressed in [1] and [7] where two policies called Constant-Rate Transmission and Transport (CRTT) and Piecewise-Constant Rate Transmission and Transport are proposed. According to CRTT, a number of frames (ADUs) are sent in advance (prior to the commencement of playback). This provides an initial "build up" and enables the remaining frames to be transmitted at a constant (CRTT) or piecewise constant (PCRTT) rate during playback. The amount of "build up" and the rate of transmission are computed given the characteristics of the movie which are known a priori and the amount of buffering available at the receiving end. However, the issue of multiplexing a number of streams carrying prerecorded traffic or how the losses will be distributed between streams sharing the same link for high network utilisation is not directly addressed as it is for the policy proposed here. In addition, the allocation of network resources is static and is determined once at the beginning of the session, as opposed to the dynamic allocation performed by the Deadline

Credit policy based on the varying network resource requirements from all users. The CRTT policy is rate based and deadlines are not explicitly defined as opposed to the Deadline Credit policy.

In [8] the idea of prefetching is exploited through a protocol called Join-the-Shortest Queue (JSQ). The driving idea of this protocol is to utilise buffer space at the receiving end in order to prefetch frames when the transmission link is underutilised. The frames are prefetched in a way such that all the ongoing connections have similar number of prefetched frames. This characteristic is similar to the fairness criterion in the DC policy. Centralised JSQ prefetching, where a centralised server serves different users over a common link, can achieve high network utilisation due to the user interaction. As with JSQ, the DC approach selects the next ADU for transmission based on the amount of prefetched ADUs. By giving priority to connections with the lowest deadline credit counter, the DC policy aims to build deadline credit uniformly while taking into account the periodic deadlines associated with the reconstruction process. In addition, the DC policy gives priority of resources (bandwidth) to the streams that have suffered the most losses as it is explained in the fairness criterion. The decentralised version of JSQ allows prefetching when there are many distributed servers in the system. In [8] it is described for a single hop case though the authors note that the presented ideas can apply in a mutlihop environment. Compared to the multihop DC policy, the decentralised JSQ (a) performs prefetching for each connection independently as opposed to the fairness criterion in DC and (b) the lack of co-ordination between different servers can result in unnecessary retransmissions which will increase the load of the network.

Paper [9] introduces a smoothing and prefetching scheme for MPEG traffic. The basic idea is to smooth the MPEG traces over a few groups of pictures (GOP) and use a central controller to coordinate the transmission of frames from multiple servers. Smoothing is not performed in order to reduce the peak rate, as in typical video smoothing approaches. Smoothing aims to keep the bit-rate of each connection constant for a small period of time, in order to facilitate the coordination of transmission from multiple servers. One of the criteria of prefetching in this approach is that all connections have a similar number of prefetched frames, as in the JSQ protocol. In [9] an assumption is made that the network between the servers and the clients is ATM based. The main differences with the DC policy are the same as in the JSQ case. The DC approach selects the next ADU for transmission based on the amount of prefetched ADUs. By giving priority to connections with the lowest deadline credit counter, the DC policy aims to build deadline credit uniformly while taking into account the periodic deadlines associated with the reconstruction process. In addition, the DC policy gives priority of resources (bandwidth) to the streams that have suffered the most losses as it is explained in the fairness criterion.

Proxy prefix caching is proposed in [10] and [11]. The authors propose that proxy caching can be used to reduce the burden on a server hosting platform or to reduce latency to the client. The driving idea is to store a prefix of frames of popular audio and video streams in a proxy between the server and the client. Upon receiving the first client request for a particular stream, the proxy retrieves the entire stream from the server and caches the prefix. For future requests, the proxy can initiate transmission directly from the cache, while receiving the remainder of the stream from the server. This allows the proxy to deliver good QoS to the client, while hiding the weaker service model between the server and the proxy and reducing the startup latency. The idea of video staging is exploited in [12]. The basic idea behind this paper is to prefetch a predetermined amount of data at a proxy server, situated between the backbone (WAN) and the LAN. It is designed using an efficient utilisation of the disk bandwidth and storage space such that data can be delivered across the backbone in a CBR rate, whereas traffic between the proxy and the client can be variable. The main differences as compared to the DC policy are (a) the DC policy builds deadline credit dynamically throughout the session as compared to the above approaches where a static prefix is stored at the proxy once upon connection set up and (b) the notion of deadline credit as a means of scheduling and sharing the resources is not considered.

There is a number of algorithms in the literature which aim to smooth VBR traffic based on various criteria. For example, the Minimum Changes Bandwidth Allocation (MCBA) algorithm [13], the Minimum Variability Bandwidth Allocation (MVBA) algorithm [14], the Rate-Constrained Bandwidth Smoothing (RCBS) algorithm [15] and the On-Off algorithm [16]. [17] presents a smoothing technique for transmitting VBR video across a sequence of nodes, which may be a subset of the path between the video server and the client. The authors present a method for minimising the playback delay and the buffer allocation subject to a constraint for the peak rate for a single link case. This is extended to compute an optimal transmission schedule for multiple links. Compared to the DC policy, these algorithms focus on a single stream at a time. Allocation of resources and scheduling are not based on any fairness criteria. In addition, they do not address the issues of application data unit deadlines and dynamic deadline credit building in a best-effort network.

In [18] a priority-based video adaptation algorithm for the streamed delivery of stored video in best-effort networks is introduced. The proposed technique first ensures a minimal playback rate can be achieved over a longer period before adding "enhancement" layer data to increase the frame rate. This scheme offers the advantage of an adaptable critical bandwidth-based approach.

The idea proposed in the Deadline Credit policy can, also, be related to the Virtual Clock (VC) mechanism proposed in [19] to schedule both real time and prerecorded traffic. In the VC algorithm deadlines are not explicitly defined and resources are wasted when the system is underutilised. In addition, there is no credit building mechanism which can reduce the demand for bandwidth during congestion.

2 The proposed Deadline Credit-based Transport Scheme: Single hop case

2.1 Preliminaries

Although the design of the DC algorithm for the multihop case is a significant contribution of this paper, the single hop case is presented first because it is easier to follow. The single hop case corresponds to the study of the improved server capacity that is achieved in retrieving content associated with multiple streams when the DC policy is employed. The algorithm described in this section is extended in section 4 to a multihop environment.

For the single hop case a pair of source and destination nodes connected with one link are considered. The source node consists of a mass storage device, a file server and the scheduler. The file server retrieves, upon request, streams of continuous media traffic. The file server is responsible for reading ADUs from a mass storage device, segmenting and encapsulating them into cells for transmission. These streams are multiplexed and transmitted into the link by the scheduler. They are transported to the receiver where they are decoded and reconstructed. The issues associated with retrieving streams of data from a storage device are beyond the scope of this paper. It is assumed that the file server is capable of retrieving ADUs from its storage at adequate speeds.

The system is assumed to be slotted. A slot may correspond to a cell, if the transport is ATM-based, or any fixed-size packet if the transport is packet-based. All departures from the source node occur at slot boundaries. Let j denote a data stream (application); j takes values in the range $0 \leq j \leq J-1$. A variable with superscript j represents a quantity associated with stream j. Also, let P_i^j denote the i^{th} ADU from stream j with $\hat{P}_i^j$ representing its length in an appropriate data unit; in this paper it is measured in cells (without loss of generality).

The per stream (application) traffic is assumed to be bursty in nature. It consists of application data units, ADUs; each ADU is of variable length. For live real-time traffic a new ADU is assumed to be generated periodically every T^j slots, where T^j denotes the period of the generation process of the application measured in slots. Thus, for live real-time traffic the number of ADUs available to the server at any time is limited by the generation process rate of the application. For prerecorded traffic though, all of the ADUs of the session are available prior to the commencement of transmission.

The application is assumed to have a periodic reconstruction rate; an ADU has to be available to the receiver every T^j slots for processing (T^j is assumed to be both the generation and the reconstruction periodic interval of the application). In order to meet this constraint, an ADU can tolerate a maximum end-to-end delay, ETE^j, which is defined to be the same for all ADUs of stream (application) j. This ETE^j delay is the sum of all propagation, queuing and transmission delays at each node for stream j and it is defined for both the single and multi hop cases. Based on the ETE^j, a deadline is defined for each node. The multihop case will be addressed later. For the single hop case and for both live real time and prerecorded traffic, the deadline of an ADU is defined as D^j slots from the instant marking the beginning of the corresponding periodic interval. For instance, the i^{th} ADU corresponds to the i^{th} periodic interval and its deadline is defined as D^j slots from the instant marking the beginning of the i^{th} periodic interval and so on, as shown in Figure 1. For the single hop case, $D^j = ETE^j$. An assumption is made in this paper that the length of any ADU be less than or equal to its default deadline.

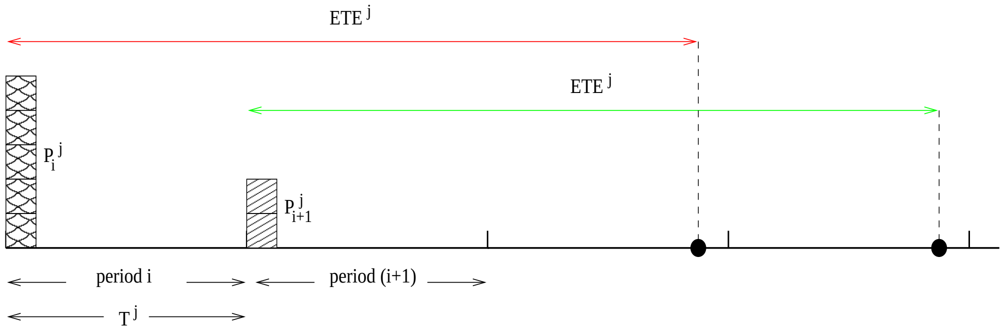

Figure 1: Periodic generation intervals with corresponding deadlines for stream j.

The QoS of the applications is assumed to be directly impacted by the number of complete ADUs received by the end user. Thus, the QoS metric considered here is the ADU loss probability. An ADU is assumed to be lost if any one of its cells is lost.

2.2 The proposed DC scheduling algorithm

The proposed scheduling algorithm determines <u>when</u> the next stream is to be selected for service, <u>which</u> stream is to be selected and whether an ADU from the selected stream is to be <u>served</u> or <u>dropped</u>. A set of metrics is used in making the above decisions, namely, the deadline credit, the losses and the priority index of the stream as well as the buffer availability of the receiving buffer. A counter is associated with each metric and stream j:

(a) The deadline credit counter, D_CR^j.

(b) The losses counter, L_CR^j.

(c) The priority index counter, P_CR^j.

The definitions of the counters and their descriptions are given in the following subsections, which are followed by a complete description of the scheduling procedure. It should be noted that the DC policy is *work conserving* in the sense that it will serve as long as there are ADUs available for service and buffer space is available at the receiver.

To facilitate the description of the counters it is assumed that key events occur and actions are taken instantaneously at the end of slots. If during a slot either the last cell of an ADU or no cell is being served, the server will be idle at the end of this slot and this is referred to as a **decision slot**. At decision slots certain actions are taken leading to the selection of a stream to be served next. For the selected stream this decision slot becomes an **examination slot**; ADUs of stream j can be scheduled for service or dropped only at examination slots of stream j. The losses counter of a stream is updated only at the examination slots of this stream as it will be discussed later.

The **Additional Delay Tolerance** (ADT) of an ADU of stream j is defined to be equal to the amount of remaining time until the ADU expires reduced by its default deadline D^j. That is if ADT is equal to a numerical value equal to d_cr for an ADU of stream j, this ADU will expire after $d\_cr + D^j$ slots.

Finally, the ADU **at the head of stream** j is defined to be the ADU of stream j which has the earliest due date among all ADUs of stream j not served or dropped yet. As it will become clear later, expired ADUs remain in the system until the next examination slot for the corresponding stream at which time they are dropped.

2.2.1 The deadline credit counter

The purpose of the deadline credit counter is to keep track of the ADT of the ADU at the head of the corresponding stream. At the same time, the deadline credit counter indicates how far ahead or behind schedule is the service of the ADUs of the corresponding stream still in the system. The value of the deadline credit counter is limited by $D\_CR^j_{max}$, setting a limit on the amount of positive deadline credit build up or number of ADUs that can be sent ahead of schedule. $D\_CR^j_{max}$ represents the capacity of the corresponding receiving buffer. It is assumed that the receiving buffer of stream j can store a maximum of n data units. They correspond to a session duration of nT^j slots. Hence, the deadline credit counter is upper bounded by $D\_CR^j \leq (n+1)T^j = D\_CR^j_{max}$ slots; $D\_CR^j_{max}$ is known *a priori*. The additional unit is added to account for the ADU of the current periodic interval. [2]

At the beginning of session j, D_CR^j is set equal to zero. During the session D_CR^j is updated as follows:

(a) During every elapsed slot:
$$D\_CR^j = D\_CR^j - 1$$
This step captures the fact that any slot that passes by reduces the ADT of the ADU at the head of the queue of stream j by one.

(b) At the end of an examination slot for stream j:
$$D\_CR^j = D\_CR^j + (k+m)T^j$$
where k is all the dropped ADUs (which are all expired) and the binary (0/1) variant m is 1 only if :

(i) $D\_CR^j + (k+m)T^j \leq D\_CR^j_{max}$

(ii) $D\_CR^j + (k+m)T^j + D^j \geq \hat{P}^j_i$

[2]The buffer availability at the end node can also be determined by the source node in the following way: the source node keeps a record of which ADU has been sent. Given the end buffer capacity and the reconstruction rate, the available buffer space can be computed before scheduling the next ADU.

where P_i^j is the non expired ADU at the head of stream j. Notice that condition (b-(i)) will never prevent dropping expired ADUs since D_CR^j is sufficiently negative when expired ADUs of stream j are present and dropping them and increasing D_CR^j by T^j will never exceed $D\_CR^j_{max}$. However, the $D\_CR^j_{max}$ constraint can prevent ADUs from being scheduled as it sets an upper bound to the maximum number of ADUs that can be sent in advance from stream j.

The non expired ADU at the head of stream j, P_i^j, will be scheduled provided that it will not expire before its transmission is completed (condition (b-(ii))), and provided that it will not increase the number of transmitted in advance ADUs from stream j beyond the limit $D\_CR^j_{max}$ (condition (b-(i))).

Step (b) "adjusts" the context of D_CR^j to describe the ADT of the ADU at the head of stream j after removing the dropped/scheduled for service ADUs of stream j. Step (b-(i)) checks the buffer availability and step (b-(ii)) checks the deadline condition for the ADU at the head of stream j.

During the session the deadline credit counter can take any positive or negative value. The counter is interpreted as follows for different ranges of values:

- If $D\_CR^j = 0$: stream j has no deadline credit and the ADU at the head of the stream j has an ADT equal to 0 slots; no ADUs have been sent in advance.
- If $D\_CR^j > 0$: the ADU at the head of the stream j has an ADT equal to D_CR^j slots.
  - If $0 < D\_CR^j \leq T^j$: no ADUs have been sent in advance.
  - If $D\_CR^j > T^j$: one or more ADUs have been sent in advance; the number is given by $D\_CR^j \bmod T^j$.
- If $-D^j < D\_CR^j < 0$: the ADU at the head of stream j has an ADT which is negative and, in effect, its total delay tolerance $D\_CR^j + D^j < D^j$ slots. It may or may not meet its deadline. If the current slot is an examination slot for stream j then this ADU will not be dropped but scheduled for service provided that the deadline condition is met (step (b-(ii)) above). No ADUs have been sent in advance if $D\_CR^j < 0$.
- If $D\_CR^j = -D^j$: the ADU at the head of stream j has just missed its deadline and it is dropped if the current slot is an examination slot for stream j.
- If $D\_CR^j < -D^j$: one or more ADUs have missed their deadline; their number is given by $\mid D\_CR^j \mid \bmod T^j$.

2.2.2 The losses counter

In addition to the D_CR^j counter, a second counter, L_CR^j, is set up to count the number of ADUs that are dropped from stream j up until the last examination slot for stream j. L_CR^j is updated at examination slots for stream j only as follows:

$$L\_CR^j = L\_CR^j + k$$

where k is equal to the number of ADUs of stream j dropped in the current examination slot for stream j.

2.2.3 The fairness criterion and the priority index counter

A mechanism is necessary to determine the relative priority between competing data streams at every decision slot, so that resources are distributed in a *fair* manner. Such a resource distribution is defined to be fair as long as it induces the same ADU deadline violation probabilities for all streams. In addition, if resources permit the transmission of ADUs in advance, the resource distribution is defined as fair, as long as it results in the same deadline credit build-up probability for all streams. In other words, if the system is underutilised the DC policy aims to build up the deadline credit of all competing streams such that the number of ADUs transmitted in advance is equal. If the system is overutilised and ADUs have to be dropped, it aims to distribute the number of dropped ADUs equally as well.

A metric needs to be defined to represent the ADUs dropped and the ADUs transmitted in advance for each stream. A metric based solely on the deadline credit would be insufficient. Giving priority to the stream with the smallest deadline credit translates to the stream with the smallest additional delay tolerance of the HoL ADU and service would be provided on a shortest time to extinction basis[3]. This would not be sufficient because the value of the D_CR^j counter does not reflect the losses stream j has suffered up to the current instant. Since the deadline credit counters are updated by the same amount for every ADU <u>served</u> or <u>dropped</u> it is not possible to distinguish among streams with the same deadline credit value which one has suffered the most losses. As a result, there is no guarantee that the available bandwidth will be distributed fairly between all competing streams.

The weight of a dropped ADU is considered equal to the weight of an ADU transmitted in advance. This raises the need to introduce the losses counter. The two counters are combined into a single metric, called priority index, that aims to capture both the current deadline credit of the stream and its cumulative losses. The priority index counter P_CR^j is defined for each stream j. Its value is determined by the values of D_CR^j and L_CR^j in the current slot as follows:

$$P\_CR^j = \frac{D\_CR^j - T^j \cdot L\_CR^j}{T^j}$$

The numerator is the total sum (measured in time slots) of the current deadline credit and the cumulative negative credit (losses). It is divided by the period T^j, to normalise the value to the number of periodic frames (or ADUs).

It should be noted that the scheduling approach is *non-preemptive*. Once a stream meets the criteria and the HoL ADU is selected to be scheduled, it is served fully before the next decision slot is established. This is the first fairness criterion (based on the authors' knowledge) that incorporates the loss history of the stream and the data units prefetched in advance.

The priority index counter represents the relative priority of competing data streams. The smaller the value of the priority index counter of a stream the higher the priority of the stream with respect to the others. This will be the stream with the most losses, if any, and among streams with no losses or the same losses it will be the stream with the lowest deadline credit. More specifically:

(a) If $P\_CR^j \geq 0$:

  (i) If stream j has suffered no losses ($L\_CR^j = 0$) P_CR^j is just equal to the ratio of the deadline credit counter to the period. The integer part represents the number of ADUs that have been sent in advance. The decimal part represents the remaining portion of

[3]The D_CR^j counter indicates the deadline credit of stream j in slots. In order to make a comparison between all competing stream and determine the one with the shortest deadline credit value in terms of number of ADUs, the D_CR^j counter should be divided by T^j.

the current periodic interval. For example, a value of 3.6 means that 3 ADUs have been sent in advance and 40% of the current periodic interval has elapsed.

(ii) If stream j has suffered losses ($L\_CR^j > 0$) P_CR^j can take any positive or negative value. For instance, suppose that at the current decision slot the losses and deadline credit counters for stream j have the following values before any update takes place: $L\_CR^j = 0$ and $D\_CR^j = -3T^j$. This implies that the priority index counter for stream j is: $P\_CR^j = (-3T^j - 0)/T^j = -3.0$. If P_CR^j is the lowest among all streams the current decision slot becomes an examination slot for stream j. The scheduler registers the first 3 losses for stream j, P_i^j, P_{i+1}^j and P_{i+2}^j. Hence, the counters are updated instantaneously to: $L\_CR^j = 3$, $D\_CR^j = 0$. Next, the ADU currently at the head of stream j, P_{i+3}^j, is scheduled and the deadline credit is updated again (in the same examination slot) to: $D\_CR^j = T^j$. Following the service completion of P_{i+3}^j the priority index of stream j is updated to $P\_CR^j = (T^j - 3T^j)/T^j = -2.0$ and the priority index counters of all other streams are updated with the current values of the corresponding deadline and losses counters. If P_CR^j is still the lowest among all streams the scheduler will schedule P_{i+4}^j (next examination slot) and D_CR^j will be updated to $D\_CR^j = T^j - \hat{P}_{i+3}^j + T^j = 2T^j - \hat{P}_{i+3}^j$.

(b) If $P\_CR^j < 0$:

(i) If the stream has suffered no losses ($L\_CR^j = 0$) then its value is always greater than -1 i.e. $P\_CR^j > -1$. A negative value represents the portion of the current interval which has elapsed but indicates that the corresponding ADU has not been scheduled yet.

(ii) If the stream has suffered losses ($L\_CR^j > 0$) then the counter interpretation is the same as for the case (b-(ii)).

2.2.4 The scheduling algorithm

The proposed algorithm is described next and it is summarised in the flowchart shown in Figure 2. The notation includes a subscript n; it is meaningful in the multihop case and should be ignored for this section. Part of the algorithm is already presented in view of the description of the evolution of the three counters in earlier subsections.

Suppose that an ADU is currently being served. The scheduler serves all of the cells of a scheduled ADU successively during a series of slots.

At every decision slot, the deadline credit counters are decremented for the number of slots elapsed since the last decision slot. At every decision slot the P_CR^j counters are updated with the current values of the D_CR^j and L_CR^j counters. The stream with the smallest priority index counter is selected. For the selected stream j the decision slot becomes an examination slot:

(a) If $D\_CR^j > -D^j$, then no ADUs have missed their deadline since the last examination slot of stream j. In this case, a scheduling decision needs to be made for the next eligible ADU, say P_i^j. In order to make a scheduling decision the deadline credit and buffer availability conditions need to be checked (subsection 2.2.1). There can be three possibilities:

(i) If both the deadline and buffer availability conditions are met for the new ADU, it is scheduled for service and D_CR^j is updated by T^j.

(ii) If the deadline condition is met but the buffer availability condition is not met then the next stream from the priority index list is selected and the procedure repeats from the beginning. If no ADU from any stream can be scheduled at the current slot due to the buffer availability condition the server remains idle in the current slot. The next slot is also a decision slot and the procedure repeats from the beginning [4].

(iii) If the deadline condition can not be met, the ADU is dropped, D_CR^j is updated by $+T^j$, L_CR^j is updated by 1 and P_CR^j is updated by the new values of D_CR^j and L_CR^j. The stream with the smallest priority index counter will need to be re-selected in order to be consistent with the scheduling algorithm. Clearly, in this case, this will be the same stream. Hence, the algorithm does not need to loop back to the selection of the shortest priority index but simply retrieve the next ADU from the same stream.

(b) If $D\_CR^j \leq -D^j$, then one or more ADUs have missed their deadline since the last examination slot of stream j. Once L_CR^j and D_CR^j are updated, P_CR^j is updated as well. Following this, the stream with the smallest priority index counter will need to be re-selected in order to be consistent with the scheduling algorithm. Again, in this case, this will be the same stream and the next ADU from the same stream is retrieved. Notice that this re-selection of the stream is clearing (dropping) all expired ADUs before leaving this stream.

The computational complexity of this algorithm can be determined as follows. At every decision slot the following computations are performed:

1) the three counters (D_CR^j, L_CR^j, P_CR^j) need to be updated. This requires $O(n)$ computations, where n is the number of streams.

2) the stream with the smallest value of P_CR^j needs to be determined. Based on step a(ii) of the algorithm, it is desirable to sort the P_CR^j counter values in ascending order. The computational complexity of this operation depends on the sorting technique used. One of the most efficient approaches is the heap sort with complexity $O(nlog_2(n))$. The computational complexity of sorting can be improved with more efficient techniques.

So the worst case computational complexity is dependent on the computational efficiency of the sorting algorithm and the number of traffic streams.

3 Numerical results for the single hop case

The following is a brief description of the simulation model of both the single hop and multihop cases. It is followed by numerical results for the single hop case.

The simulations were conducted in C. Traffic streams were modeled as traffic generation processes. There are two types of traffic models. The first generates variable size ADUs based on a uniform distribution (1-20 cells) at a periodic rate. The second generates fixed size ADUs in an On-Off mode. Their length is equal to 1 cell with probability p and equal to $LP_{max} = 20$ with probability $1 - p$. Simulations for the single hop case use the first type of traffic model, whereas simulations for the multihop cases use both. In all cases, the traffic streams were active throughout the simulation.

The node models implemented the DC scheduling policy and the associated counters. Each simulation was run for 10^{10} slots and the results were averaged over 10 simulation runs (unless otherwise noted).

For the single hop case, the performance of the DC policy is investigated with the following objectives in mind:

- To show its fairness when applied to a set of data streams.

[4] A more efficient but perhaps slightly more complex approach would be to determine the first slot at which an ADU from any stream can meet both the deadline and buffer conditions so that the above procedure is not repeated.

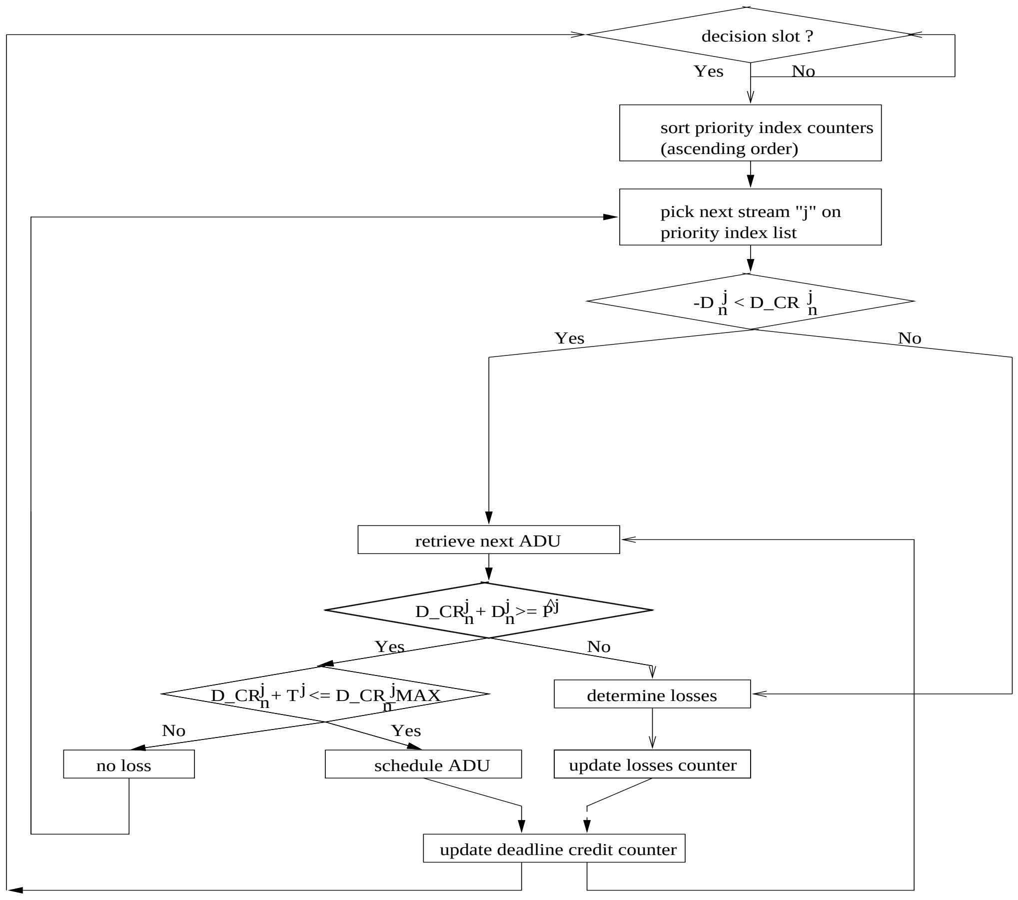

Figure 2: The DC scheduling algorithm flowchart.

- To measure the induced losses under varying system load.

Some plots focus on comparison with the performance of the FIFO policy, which is of much lower complexity, and an approximation (described later in the section) of the CRTT policy as described in [1] and [7]. The performance is measured in terms of the ratio of the induced ADU losses (U_{lost}) to the total number of ADUs offered to the scheduler (U_{off}) versus increasing system load.

The average bandwidth required by all streams, U_{req}, is defined as the sum of the average cell rate generation of N data streams, that is, $U_{req} = \sum_{j=0}^{N-1} \frac{\hat{P}^j_{ave}}{T^j}$. The system load is measured as the ratio of offered load to the link bandwidth.

In order to test the *fairness* of the DC policy when applied to a set of data streams with the same traffic characteristics and the same QoS requirements, the variation of the deadline credit of each stream is recorded for the duration of the simulation. Figure 3 shows the deadline credit counter for $S = 4$ streams with $T = 40$ slots, $D\_CR_{max} = 500$ slots and $D = T$. The system load is at 100%. The simulation is run for 10^6 slots and a measurement is taken every 1000 slots. As it can be seen from the plots the deadline credit counters of all four streams oscillate around the same average level throughout the entire simulation. No stream builds up credit at the expense of competing streams. The resources are distributed *equally* between all four streams. Since the system load is at 100% there is no capacity left to build up credit (0% credit build up capacity). The deadline credit counters oscillate mostly in the range of 0-100 slots and never reach the D_CR_{max} value of 500 slots. The losses are also distributed *equally* between all streams, as shown in Figure 4. Since there is no credit build up capacity there is no transient period.

The behavior of the deadline credit counter is plotted for the same system parameters at a lower system load as shown in Figure 5. In this case, U_{req} is at 85% so 15% of the total capacity is used as *credit build up capacity*. This capacity is used during the transient period of the system whereby the deadline credit counters increase. Once it reaches the steady state region, part of the extra 15% capacity is wasted since the deadline credit can not increase any further but can only be maintained at this level. In this plot only the first 10^4 slots are shown to highlight the transient period of the system in the beginning. The average value of the deadline credit grows steadily from 0 to just below the D_CR_{max} value of 500 at the same rate for all streams and then remains at this steady state.

The performance of the DC policy is compared against that of the FIFO policy which is of much lower complexity. It does not exploit the available bandwidth when the system is underutilised in order to build up credit. For the FIFO implementation it is assumed that ADUs from all streams arrive at the scheduler in a random order at a rate of one ADU per stream and period. Two versions of the FIFO policy are considered. The first one, referred to here as pure-FIFO or p-FIFO, serves one ADU per period from each stream on a first in-first out basis without considering the deadlines of the ADUs. The second version is slightly more complex and is referred to here as d-FIFO. It discards the ADUs which can not meet their deadline.

Figure 6 shows the losses induced by p-FIFO, d-FIFO and DC for $S = 4$ data streams with $D\_CR_{max} = 100$ and 500 with the period varying in the range of $T = 25$ to $T = 100$ slots. The loss ratio reaches 1.0 for the p-FIFO implementation when the system load reaches 100%. The losses for the d-FIFO follow the p-FIFO curve for a system load between 50% and 70% and then grow at a lower rate than the p-FIFO and converge to the DC curve for system load higher than 130%. The losses under the DC policy are at least 4 orders of magnitude less than the losses induced by the FIFO approaches for a system load less than 90%. The case of $D\_CR_{max} = 100$ corresponds to a DC scheduler that can only send one ADU per period from each stream. This is the *limiting case* of the DC policy since *no deadline credit can be built*[5]. For the case of $D\_CR_{max} = 500$ the losses are significantly reduced.

Figure 7 shows the ADU losses induced under the DC policy for period $T = 100$ slots and varying values of the $D\_CR_{max} = 100, 200, 500$ slots which correspond to a buffer capacity of 1, 2, and 5 ADUs respectively. The losses are measured as the number of multiplexed streams is increased. $D\_CR_{max} = 100$ corresponds to the limiting case as explained earlier. As D_CR_{max} increases the level of the induced losses falls, i.e. for $D\_CR_{max} = 500$ the losses for up to and including 8 data streams are of the order of 10^{-8} or less.

In the next figure the DC policy is compared to an approximation of the CRTT scheme proposed in [1] and [7]. The CRTT scheme is simulated here in the following way: A number of ADUs from each stream are initially transmitted in advance (prior to the commencement of playback) which corresponds to the initial $BUILD\_UP$. The remaining ADUs are transmitted at a pre-defined constant rate of r cells/period during playback. The rate r is guaranteed for each stream. A stream can not transmit more than r cells/period. Figure 8 shows plots for three different values of $BUILD\_UP = 100, 500, 1000$ slots which correspond to 1, 5 and 10 ADUs respectively. In the plots the rate r takes values in the range of $r = 1.0$(average ADU length/period) to $r = 1.3$(average ADU length/period) for each stream. For each rate, the losses are plotted as a function of the increasing number of multiplexed streams. For a given rate r, the losses fluctuate

[5] An ADU can be sent in advance only if the ADU corresponding to the the current periodic interval misses its deadline.

around an average value as the number of streams increase which is expected as each competing stream has a guaranteed amount of bandwidth (r). Since a rate is guaranteed for each stream, as r increases the total number of streams that can be served by the system decreases. Notice that the loss curves extend up to the maximum number of streams that can be accommodated by the system for a given r and that as r increases the loss curves become shorter in length. The losses for rate $r = 1.0$(average ADU length/period) are higher than 90% and the maximum number of streams than can be served is 10. As r takes higher values the average value of the losses decreases and the maximum number of streams that can be served ranges between 7 and 9.

For the case of $BUILD\_UP = 100$, which corresponds to an initial build up of 1 ADU, the DC policy is implemented for $D\_CR_{max} = 100$ and 200 slots, which corresponds to a buffer capacity of 1 ADU (limiting case) and 2 ADUs respectively. For $D\_CR_{max} = 100$ the performance of the DC policy is comparable to that of the CRTT implementation for various values of r. For the case of $D\_CR_{max} = 200$, however, the DC policy provides the same or lower levels of losses for the same number of multiplexed streams.

For the case of $BUILD\_UP = 500$, and 1000 which corresponds to an initial build up 5 and 10 ADUs respectively, the DC policy is implemented for $D\_CR_{max} = 500$ slots, which corresponds to a buffer capacity of 5 ADUs. In both plots, when compared against the CRTT implementation for a rate $r = 1.1$(average ADU length/period) the DC policy has the same level of losses for $S = 9$ streams and lower levels of losses by at least two orders of magnitude for a smaller number of streams.

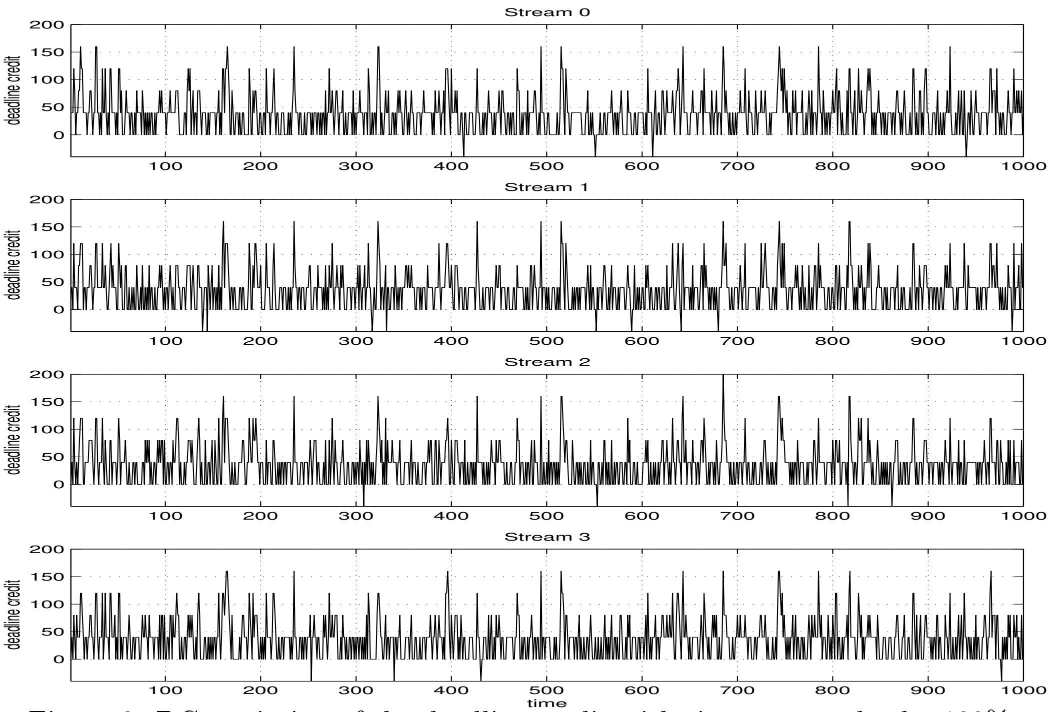

Figure 3: DC: variation of the deadline credit with time, system load= 100%.

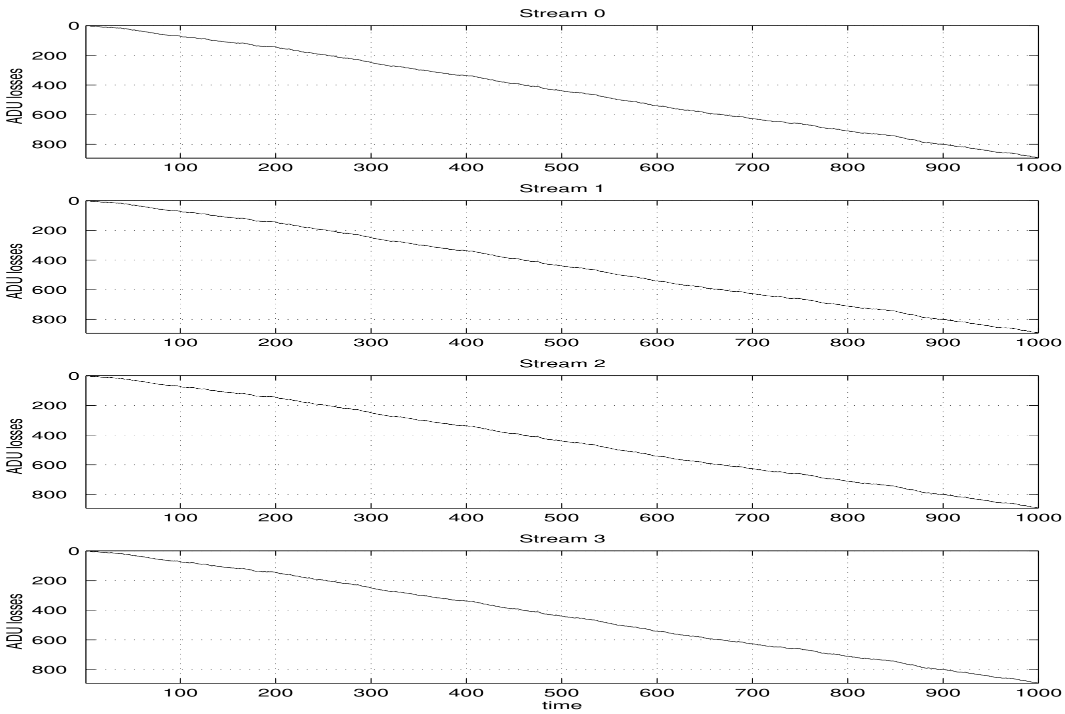

Figure 4: DC policy: variation of the losses with time, system load= 100%.

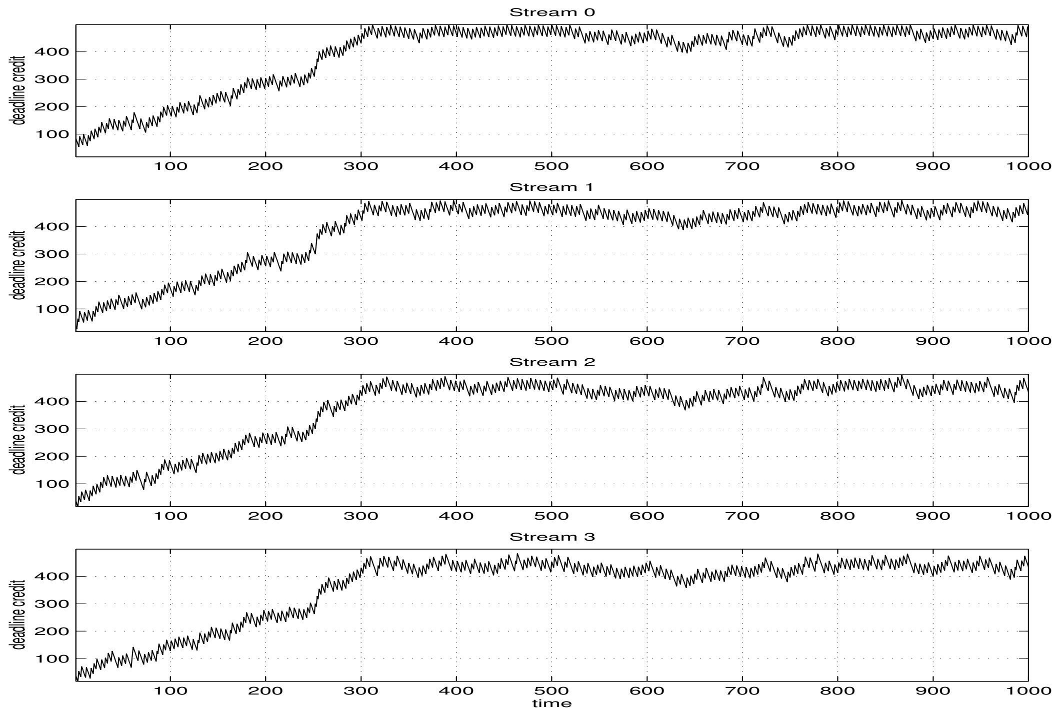

Figure 5: DC: variation of the deadline credit with time, system load= 85%.

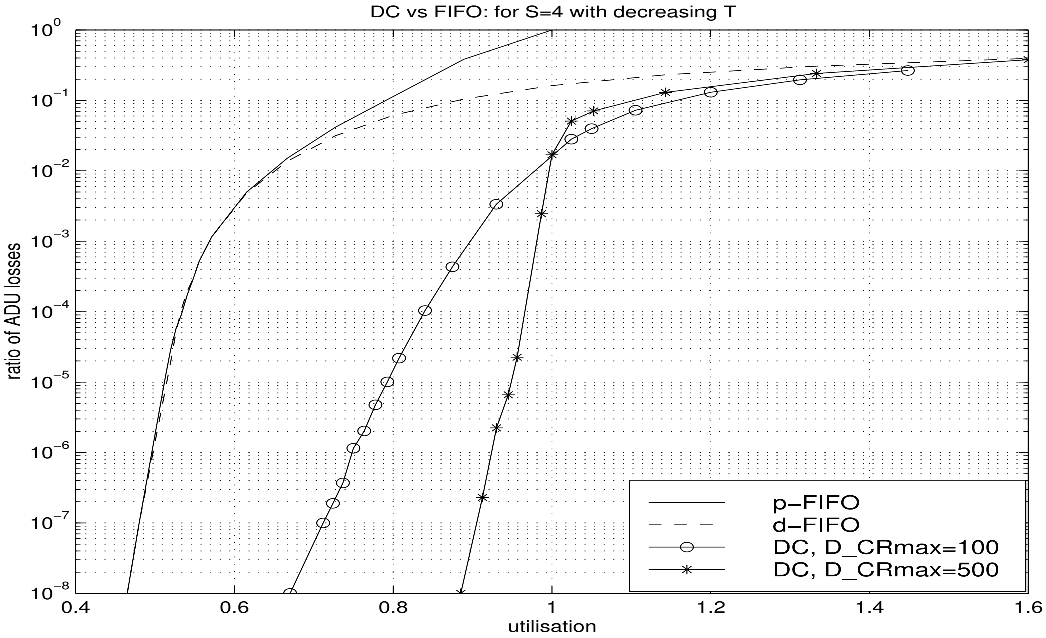

Figure 6: DC vs FIFO: ADU losses vs system load, $T = 25 - 100$ slots.

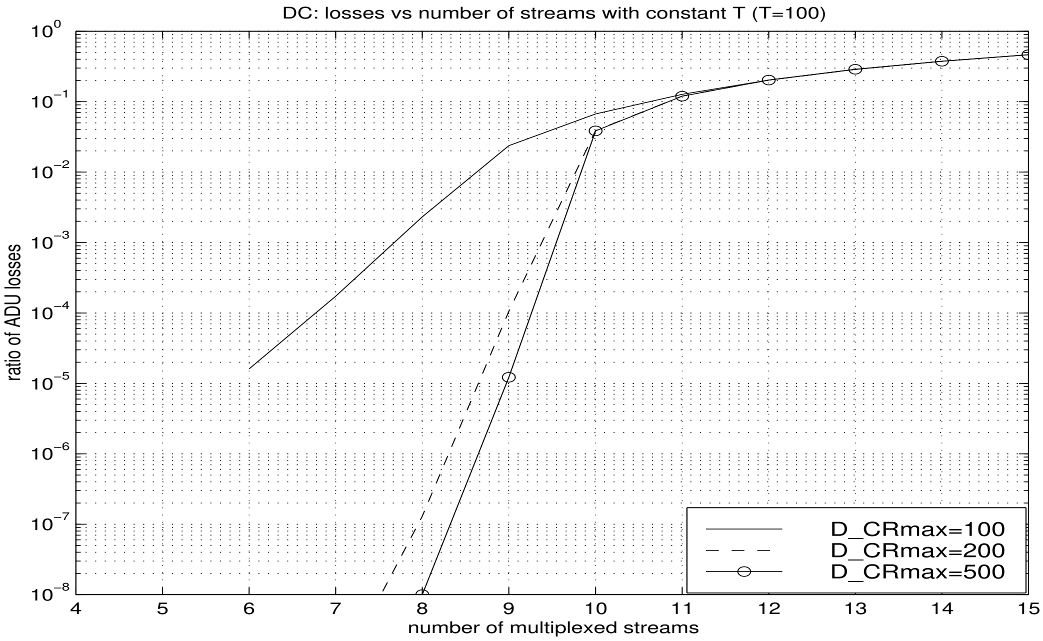

Figure 7: DC policy: ADU losses vs number of multiplexed streams.

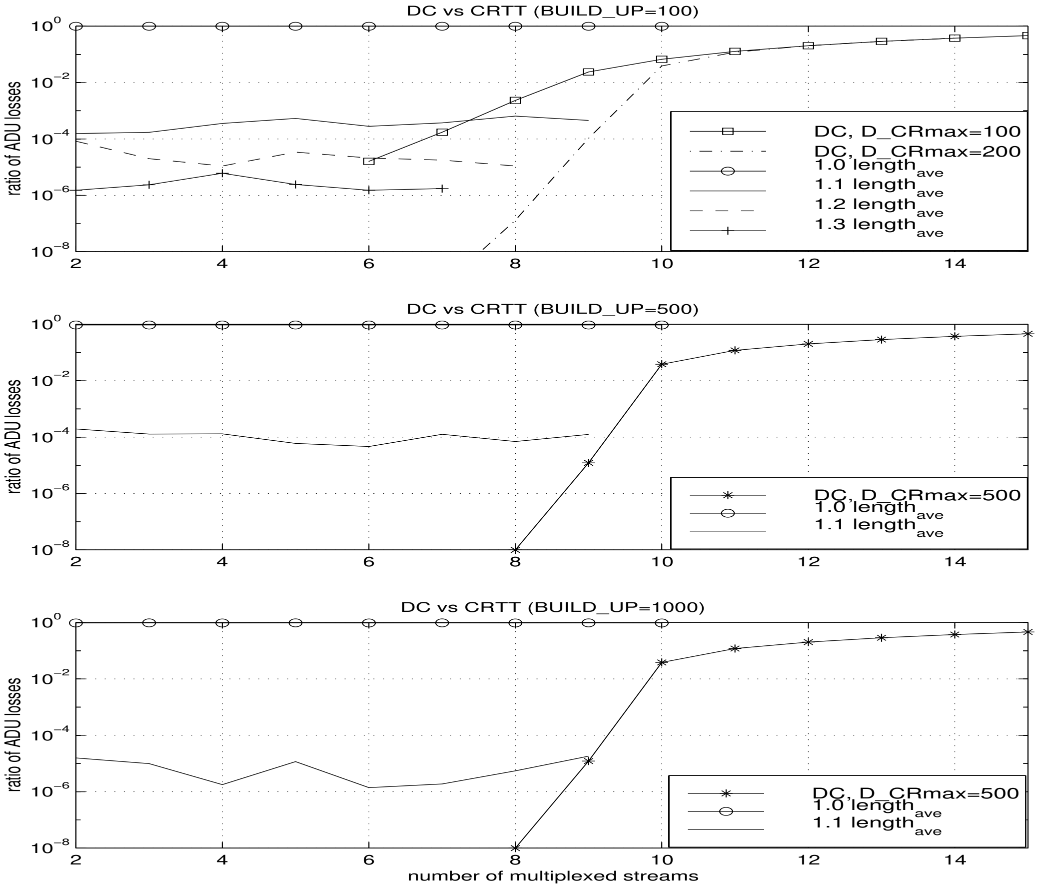

Figure 8: DC vs CRTT: ADU losses vs number of multiplexed streams.

4 The proposed Deadline Credit-based Transport Scheme: Multihop case

4.1 Preliminaries

The DC policy can be extended for a multihop network. A multihop network consists of source, intermediate and destination nodes (with bi-directional links). A node can have a number of incoming and outgoing links. Each link arrives at (or leaves from) a port. Ports for incoming links have an amount of buffer space allocated to the streams that arrive at this port. Ports for outgoing links have a scheduler. Streams are multiplexed and transmitted by the scheduler onto the correct output link in order to reach to the corresponding receiver where they are decoded and reconstructed.

The notation used is the same as for the single-hop case with the following addition: Let n denote a node; n takes values in the range $1 \leq n \leq N$. A variable with subscript n represents a quantity associated with node n. Also, let P_i^j denote the i^{th} ADU from stream j which corresponds to the i^{th} periodic interval of the session of stream j; let $\hat{P}_i^j$ represent its length in cells (or bytes, packets) [6].

In the multihop network, the sum of all propagation delays is subtracted from ETE^j; the remaining is the total amount of time an ADU can spend on queuing and transmission delays in all the nodes. This amount of time can be distributed in a variety of ways between nodes. For instance, it can be distributed equally between nodes or based on the expected delay per node. The distribution can be static (once upon connection set-up) or dynamic (varying based on the load of each node during the session). As it will be clarified later, in this paper the node delays are defined cumulatively from one node to the next. If part of the node delay is not used in one node it is available to be used at a later node.

The proposed scheduling algorithm for the multihop network is executed at every outgoing port of each node. The set of metrics used in making the above decisions is essentially the same as in the single hop case. A counter is associated with each metric per node n and stream j: the deadline credit counter, $D\_CR_n^j$, the losses counter, $L\_CR_n^j$, the priority index counter, $P\_CR_n^j$. In addition to the three metrics two more parameters are used in making scheduling decisions: the sequence number, SN and the buffer availability.

A sequence number is defined for each ADU of stream j. The definitions of the counters and parameters and their descriptions for the multihop network are given in the following subsections.

4.2 The deadline definition

The sum of all propagation, queuing and transmission delays from source to destination is equal to the end-to-end delay, ETE^j. Thus, ADU P_i^j is needed at the receiver ETE^j slots after the instant marking the beginning of the i^{th} periodic interval of stream j or $(i \cdot T^j + ETE^j)$ slots after the beginning of the session of stream j. The total propagation delay from source to destination consists of $PROP\_DELAY_{1,2}^j$ from node 1 to node 2, $PROP\_DELAY_{2,3}^j$ from node 2 to node 3 and so on. The maximum allowed queuing and transmission delay at each node is denoted as $NODE\_DELAY_1^j$ for node 1, $NODE\_DELAY_2^j$ for node 2 and so on. As it was mentioned earlier, this node delay can be determined in a variety of ways. This is out of the scope of this paper.

Using the assigned node and propagation delay tolerances a deadline is defined per stream and node. The deadline is defined *cumulatively* from one node to the next such that if an ADU leaves a node before the expiration of its deadline by an amount of t slots, these slots are automatically available at later nodes in addition to their default delay tolerance where there they may be used if there is higher contention for bandwidth. If an ADU can not meet its deadline at any node it is dropped. Dropping an ADU at a node n due to the node deadline violation, D_n^j, does not imply that the ETE^j delay would have necessarily been violated as well. There is always the possibility

[6] Subscript n is not used with variable P.

that this ADU could have met the ETE^j delay if it experienced delays at later nodes that were less than their respective deadlines. In that case, the ADU would have been dropped unnecessarily but this is a consequence of the distribution of the end-to-end deadline among the involved nodes.

More specifically, the deadline of any data unit P_i^j is defined as the number of time slots from the instant marking the beginning of the i^{th} periodic interval of stream j as follows:

- At node 1: $D_1^j = NODE\_DELAY_1^j$
- At node 2: $D_2^j = NODE\_DELAY_1^j + PROP\_DELAY_{1,2}^j + NODE\_DELAY_2^j$ $= D_1^j + PROP\_DELAY_{1,2}^j + NODE\_DELAY_2^j$
- At node 3: $D_3^j = D_2^j + PROP\_DELAY_{2,3}^j + NODE\_DELAY_3^j$

- At node n: $D_n^j = D_{n-1}^j + PROP\_DELAY_{n-1,n}^j + NODE\_DELAY_n^j$

Figure 9 shows an example of the definition of deadlines for an ADU P_i^j. Node 1 is the source and node 4 is the destination. For the first node the default deadline is defined as $D_1^j = NODE\_DELAY_1^j$ after the instant marking the beginning of the i^{th} periodic interval. At the second node, the deadline is defined as $D_2^j = D_1^j + PROP\_DELAY_{1,2}^j + NODE\_DELAY_2^j$. Similarly for node 3 and, finally, the sum of all allowable node and propagation delays is the ETE^j.

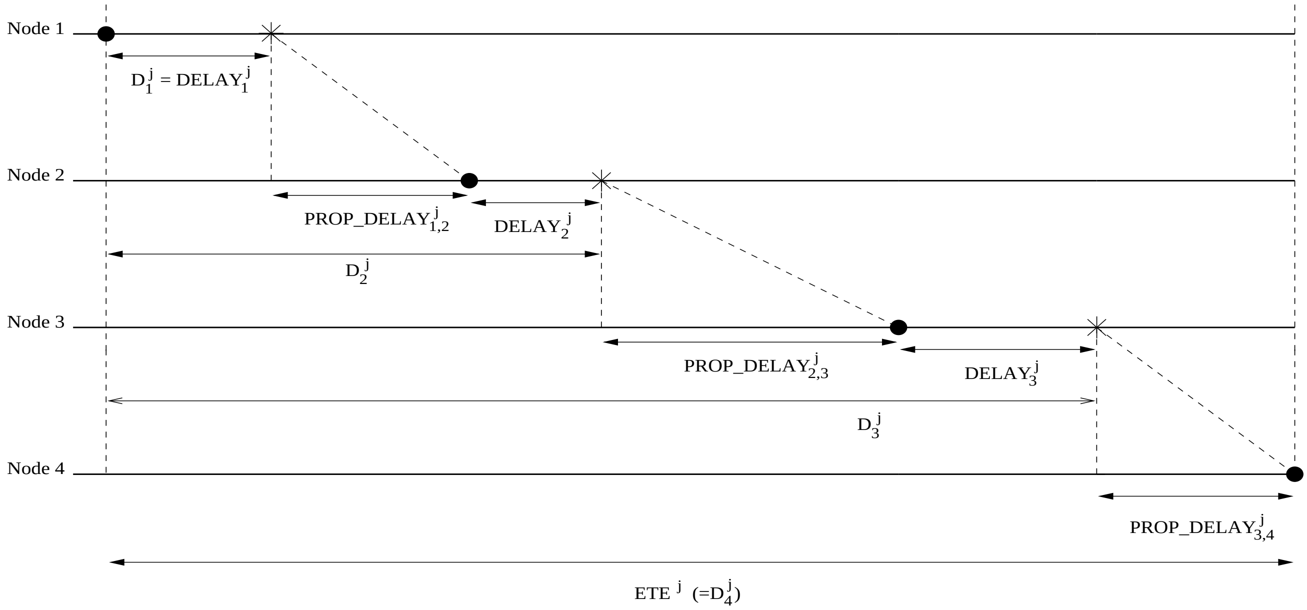

Figure 9: Example deadline definition for an ADU P_i^j.

4.3 The sequence number

The SN marks the sequence of ADUs of a particular stream j. The SN is stored in the header of the ADU. It is used to identify which ADU corresponds to which periodic interval in order to determine whether the ADU can meet its deadline or has expired and needs to be dropped.

More specifically, if an ADU can not meet its deadline at some node n it will be dropped and the next ADU will be scheduled for service. The scheduler can determine the number of expired ADUs at the waiting queue of the current node from the value of the deadline credit counter of the node. In the downstream nodes however, the value of their deadline credit counter is not enough to accurately determine the number of ADU that were dropped at upstream nodes (if any). It is in this case that the scheduler needs to read the SN of the incoming ADUs to determine how many ADUs have been dropped at upstream nodes (if any) and make the necessary adjustments to all the relevant counters. The SN could be used in all cases to identify the sequence of ADUs but the above paragraph explained the case where it would be necessary. Its use will be clarified further in the following sections.

4.4 The buffer

As it was mentioned in the beginning of section 4, ports for incoming links have an amount of buffer space allocated to the incoming streams. Buffer space may be assigned separately to each stream or collectively to a set of streams. This is out of the scope of this paper. As part of the scheduling process each node, n, needs to know the amount of available buffer space of the next downstream node, $n+1$.

In this paper, it is assumed that buffer space is assigned separately per stream at each incoming port. This guarantees that when the buffer condition (introduced in section 4.5) holds no ADUs will be dropped due to buffer overflow. For implementation purposes, the buffer availability can be checked in the following way. Each node n knows the total amount of buffer space available per stream at the next downstream node, $n+1$, referred to as $Buffer^j_{n+1}$. In addition, each node keeps a record of the total amount of data transmitted per stream measured in cells or bytes or any other appropriate unit, $data^j_n$. Finally, each node n requires from the downstream node that it transmits data to, $n+1$, the amount of data node $n+1$ has transmitted, again measured per stream, referred to as $data^j_{n+1}$. The buffer availability per stream in the next downstream node can be determined as follows:

$$Available\_buffer^j_{n+1} = Buffer^j_{n+1} - (data^j_n - data^j_{n+1})$$

The above formula will account for all the ADUs currently in the waiting queue of node $n+1$ as well as all the ADUs that are currently traveling toward node $n+1$.

If buffer space were assigned collectively for the set of streams arriving at a node $n+1$ from the same node n then it would still be possible to guarantee no buffer overflow using the above method. If, however, the buffer pool at a node $n+1$ is shared between streams arriving from different nodes then guaranteeing no buffer overflow may not be possible or it may require additional feedback of information between nodes, thus, increasing the processing overhead.

The amount of buffer space available at the next node at any instant determines how many ADUs can be sent in advance. Feedback is necessary to update upstream nodes on the value of the $data^j_n$ parameter. In this paper it is assumed that feedback is available periodically and each node uses the latest buffer update in order to determine how many ADUs can be sent in advance. There is a trade-off between the frequency of the feedback process and the accuracy of the parameter value. Instantaneous (per slot) feedback will be the most accurate but will generate the most processing overhead. Another option, which would provide the same accuracy, would be to send feedback only when an ADU is transmitted and the buffer occupancy changes. An option which would require less frequent updates is to send feedback when the buffer occupancy exceeds a threshold value. The larger the time interval becomes between consecutive feedbacks the more conservative the above method is in estimating the available buffer space.

Based on the latest buffer update a stream can build a maximum possible deadline credit from the current value of $D\_CR_n^j$ to $D\_CR_n^j\_MAX$ till the next buffer availability update.

4.5 The deadline credit counter

A deadline credit counter is defined for each stream j and node n that stream j traverses, denoted by $D\_CR_n^j$. The counter is initialised and updated such that upon arrival of a new ADU at a node n, the value of $D\_CR_n^j$ is the same as the value of $D\_CR_{n-1}^j$ when transmission of the ADU was completed at node $n-1$. This enables the scheduler at any node n to determine how far ahead or behind schedule is the service of the ADUs of stream j *in all the nodes up to and including* node n and use the value of the counter in making scheduling decisions.

The value of the deadline credit counter is limited by the available buffer at the next node for stream j, setting a upper bound on the amount of positive deadline credit build up or number of ADUs that can be sent ahead of schedule based on the latest feedback update.

At the beginning of session j, deadline credit counters are initialised at all the nodes that stream j will traverse. The counter of each node is initialised such that the propagation delays and maximum allowable node delays of upstream nodes are accounted for. For instance, suppose that stream j traverses nodes 1 through N. The deadline credit counters at each node are initialised as follows:

- Node 1: $D\_CR_1^j = 0$
- Node 2: $D\_CR_2^j = D\_CR_1^j + NODE\_DELAY_1^j + PROP\_DELAY_{1,2}^j$
- Node 3: $D\_CR_3^j = D\_CR_2^j + NODE\_DELAY_2^j + PROP\_DELAY_{2,3}^j$
- Node n: $D\_CR_n^j = D\_CR_{n-1}^j + NODE\_DELAY_{n-1}^j + PROP\_DELAY_{n-1,n}^j$

This is consistent with the earlier deadline definitions. The deadline credit counters are initialised cumulatively from one node to the next such that if the first ADU leaves a node before the expiration of its deadline by an amount of t slots, these slots are automatically available at later nodes in addition to their default deadlines.

During the session $D\_CR_n^j$ is updated as follows:

(a) During every elapsed slot:

$$D\_CR_n^j = D\_CR_n^j - 1$$

This step captures the fact that any slot that passes by reduces the ADT of the ADU at the head of the queue of stream j by one time slot.

(b) At the end of an examination slot for stream j:

$$D\_CR_n^j = D\_CR_n^j + (k+m)T^j$$

where k is all the dropped ADUs (which are all expired) and the binary (0/1) variant m is 1 only if :

(i) $\hat{P}_i^j \leq Available\_buffer_{n+1}^j$ (buffer condition), and

(ii) $D\_CR_n^j + (k+m)T^j + D_n^j \geq \hat{P}_i^j$ (deadline condition)

where P_i^j is the non expired ADU at the head of stream j. The above conditions update the deadline credit counters of each stream at every node in the same manner as in the single hop case.

If an ADU can not meet its deadline at any node n it will be dropped and the next ADU will be scheduled for service. The scheduler can determine the number of expired ADUs, k, either from the deadline credit counter or from the SN as it was mentioned earlier.

During the session the deadline credit counter can take any positive or negative value. The counter is interpreted in the same manner as in the single hop case for different ranges of values.

4.6 The losses counter

The losses counter, $L\_CR_n^j$, counts the number of ADUs that are dropped from stream j up until the last examination slot for stream j. L_CR^j is updated at examination slots for stream j only as follows:

$$L\_CR_n^j = L\_CR_n^j + k$$

where k is equal to the number of ADUs of stream j dropped in the current examination slot for stream j. The dropped ADUs may belong to two categories:

- ADUs that expired while waiting at node n (can be computed through the $D\_CR_n^j$ or the SN).
- ADUs that were dropped at an earlier node because their deadline could not be met (can be computed through the SN).

As it was mentioned earlier, in the first case the value of the deadline credit counter at node n is sufficient to determine the number of expired ADUs. In the second case, the scheduler uses the SN. In both cases, the losses counter, $L\_CR_n^j$, is updated with the losses for all the nodes up to and including node n.

4.7 The priority index counter

The priority index counter $P\_CR_n^j$ is defined for each stream j and node n. Its value is determined by the values of $D\_CR_n^j$ and $L\_CR_n^j$ in the current time slot as follows:

$$P\_CR_n^j = \frac{D\_CR_n^j - T^j \cdot L\_CR_n^j}{T^j}$$

The priority index counter represents the relative priority of competing data streams based on the same fairness criterion described earlier. The smaller the value of the priority index counter of a stream the higher the priority of the stream with respect to the others. This will be the stream with the most losses, if any, and among streams with no losses or the same losses it will be the stream with the lowest deadline credit. The value of the priority index counter is interpreted in same way as in the single hop case.

4.8 The scheduling algorithm

The scheduling algorithm runs at every node along the path of a stream and it is summarised in Figure 2.

5 Numerical results for the multihop case

The performance of the DC policy for the multihop network is investigated under two different scenarios which demonstrate the improvement in the induced ADU loss rate under the DC policy compared to an alternative approach which permits no deadline credit build up. To be specific, the alternative approach schedules only one ADU per period subject to buffer availability. The different scenarios are considered with respect to the traffic mix and the distribution of buffer space per node. The simulation model is described in section 3.

5.1 Simulation scenario 1

The first scenario considers the case where a set of streams has the same source, destination and the same intermediate nodes. Its purpose is to show that the DC algorithm utilises the available bandwidth and buffer space and improves on the induced ADU losses as compared to the alternative policy. The intermediate and destination nodes are assumed to have the same amount of available buffer space per stream. The node delays are distributed equally per node.

More specifically, there are three nodes. The source is node 1 and the destination is node 3. All streams have to travel from node 1 to node 3 through node 2. All streams, except from two background traffic streams, are assumed to have the same traffic characteristics. They have the same period, $T^j = 100$ slots, the same ADU length distribution (uniform with maximum length $LP_{max} = 20$ cells), the same end-to-end deadline which is equal to one period, $ETE^j = T^j$ slots, and the same QoS requirements. The two background traffic streams have the same characteristics as above with the exception that the length of ADUs has a bursty On-Off distribution. It is equal to 1 cell with probability p and equal to $LP_{max} = 20$ with probability $1 - p$. For this simulation scenario $p = 0.3$, which implies that the background traffic streams generate long ADUs 70% of the time.

In Figure 10, the ratio of ADU losses under the DC policy is plotted for the cases where the available buffer space at each node is at most equal to 1 ADU, 2 ADUs and 5 ADUs per stream. Each stream requires an average of 10.5% of the link bandwidth; for example, 5 streams correspond to 52.5% of each link's capacity.

For the DC policy, the case of $Buffer = 1$ ADU is not very indicative of its performance. When the available buffer space per stream can store at most one ADU of maximum length, the proposed algorithm can build a very limited amount of deadline credit and, in essence, available bandwidth is wasted. For larger buffer availability the potential build up is higher and the performance is improved.

With respect to the alternative of approach of serving one ADU per period, there is a significant difference in the losses, especially for the cases of $Buffer = 2$ ADUs and $Buffer = 5$ ADUs. For a system load approaching 100% the losses rates converge. This is expected since there is very little or no bandwidth available to be used for building deadline credit and, as a result, the available buffer space can not be utilised. Hence, the losses are comparable in all cases.

5.2 Simulation scenario 2

The second scenario considers the case where a set of streams with homogeneous traffic characteristics has to travel from the same source (node 1) to the same destination (node 3) through one intermediate node (node 2). In addition, there are background traffic streams (in this case three streams) which travel from node 1 to node 2. At node 2, they can be assumed to terminate or continue on to another link, separately from the rest of the streams. Their purpose is to consume bandwidth over the first hop in an ON-OFF fashion. As in simulation scenario 1, they are assumed to generate ADUs of length equal to 1 cell with probability p and equal to $LP_{max} = 20$ with probability $1 - p$. When in the OFF state they allow the DC policy to build deadline credit. When in the ON state they create a bottleneck over the first hop.

The objective of this scenario is to show that the DC algorithm dynamically builds deadline credit by utilising the available bandwidth and buffer space at nodes 2 and 3. This improves on the induced ADU losses as compared to the alternative policy. Again, the intermediate and destination nodes are assumed to have the same amount of available buffer space per stream and the node delays are distributed equally per node.

The traffic characteristics for the regular and background traffic streams remain the same as in simulation scenario 1. The only difference is that the available buffer space of the background traffic streams at node 2 is assumed to be infinite. This assumption is made in order to place as few restrictions on the background streams as possible and test the effectiveness of the DC algorithm. The ON-OFF states of the background streams are probabilistically equal, that is, $p = 0.5$.

Figure 11 shows the ratio of the induced ADU losses versus the number of multiplexed streams (not including the background traffic streams). Over the first hop (node 1 to node 2) the total number of streams is always the number of streams indicated in the x-axis plus three background traffic streams. Over the second hop (node 2 to node 3) the total number of streams is the number indicated on the x-axis. As in scenario 1, there is a significant improvement in the induced ADU losses between the two approaches and for the case of the DC policy, the performance improves with increasing available buffer space.

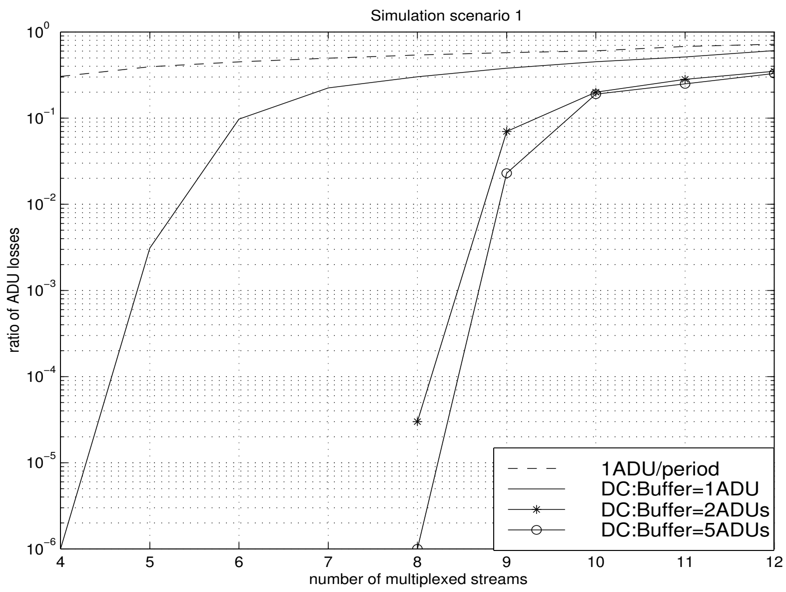

Figure 10: Simulation scenario 1: Ratio of induced ADU losses vs number of multiplexed streams.

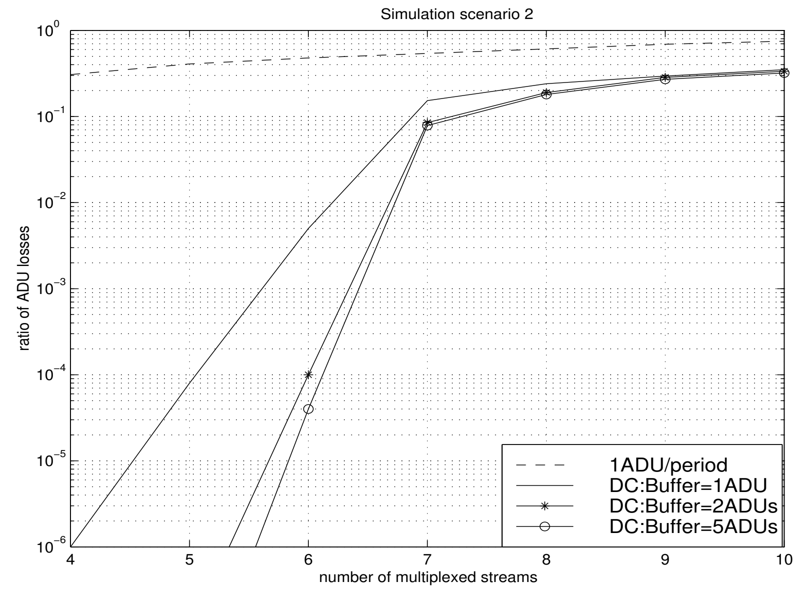

Figure 11: Simulation scenario 2: Ratio of induced ADU losses vs number of multiplexed streams (not including background traffic).

6 Conclusions

This paper presents the Deadline Credit scheduling policy which is applicable to prerecorded, VBR streams with deadline driven periodic requirements for the delivery of complete data units (ADUs). The Deadline Credit policy aims at utilising all (if possible) of the available capacity by allocating bandwidth *dynamically* to a number of CM data streams and distributing the available bandwidth *fairly* between all competing CM streams. The DC policy achieves workahead smoothing of rate variability by sending ADUs in advance during periods of system underutilisation. This translates to deadline credit build up which can be consumed during periods of increased bandwidth demand.

The DC policy is presented first for the case of the single hop network. In this case, the source server sends data directly into the receiver buffer. The source can compute the available buffer space at the receiver without feedback and use this information to regulate the rate at which ADUs are sent in advance. For the multihop case, there is a series of nodes from the source to the destination, each of which can (potentially) act as a memory depository. This paper describes the case where all the nodes from source to destination are DC capable and can act as memory depositories. In a more general scenario, not all nodes may have this capability. Some intermediate nodes may not act as memory depositories and may not be DC capable. These nodes may have simple forwarding mechanisms. In this case, virtual tunnels can be set up between two consecutive DC capable nodes, such as between node 1 and node 4 or between node 4 and node 6 in Figure 13. Node 1, for instance, will *see* the buffer availability of node 4 and the available bandwidth of the entire path node 1-node 2-node 3-node 4 and will use this information as input to the DC algorithm.

This is directly relevant to the Internet architecture where certain nodes are more *intelligent* (i.e. proxies) with enhanced functionality and additional memory capacity. In this way, not all nodes need to be compatible with the DC approach. The *intelligent* nodes would require a protocol to ensure the communication between them.

The simulation results for the single hop case focus on demonstrating the fairness of the DC policy which distributes the deadline credit and ADU losses equally between competing streams. They, also, show the reduction in losses when compared to the p-FIFO, d-FIFO and an approximation of CRTT. Lastly, they highlight the role of the available buffer at the receiver in reducing the losses.

The simulation results for the multihop case compare the DC policy with alternatives which do not allow deadline credit build up. They demonstrate how the DC algorithm exploits the available bandwidth during periods of underutilisation. The benefits are more profound in these network hops that carry bursty traffic. These may be all the links on the end-to-end path (simulation scenario 1) or a few links along the path (simulation scenario 2). Deadline credit build up helps keep the losses low when the links that carry the more bursty traffic experience an increased demand for resources

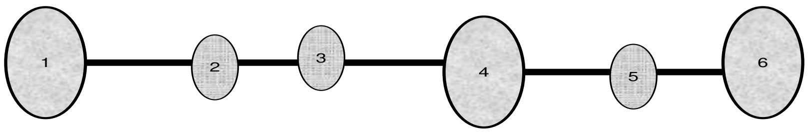

Figure 12: Example of a path including *intelligent* nodes among *simple forwarding* nodes.

References

[1] J.M. Mc Manus and K.W. Ross. Video on Demand over ATM: Constant-Rate Transmission and Transport. *IEEE JSAC*, November 1996.

[2] Zoe Antoniou and Ioannis Stavrakakis. Earliest Due First Scheduling for Application-Level QoS Delivery. *Proceedings of IEEE Conference on Protocols for Multimedia Systems - Multimedia Networking, pp.172-182, Santiago, Chile*, November 1997.

[3] Sujata Banerjee. Translating Application Requirements to ATM Cell Level Requirements. *Proceesings of IEEE International Conference on Communications*, 1997.

[4] X. Wang and I. Stavrakakis. Study of scheduling for group-based quality of service delivery. *Third volume on Performance Modeling and Evaluation of ATM Networks, ed.: D. Kouvatsos, Chapman & Hall*, 1997.

[5] S. S. Lam and G.G Xie. Group priority scheduling. *Technical Report TR-95-28, University of Texas at Austin, Austin, Texas*, January 1996.

[6] S. S. Lam and G.G Xie. Real-time block transfer under a link sharing hierarchy. *Technical Report, University of Texas at Austin, Austin, Texas*, 1996.

[7] J.M. Mc Manus and K.W. Ross. Prerecorded VBR Sources in ATM Networks: Piecewise-Constant-Rate Transmission and Transport. *Technical Report, University of Pennsylvania, Philadelphia, PA, http://www.seas.upenn.edu/~ross/*, September 1995.

[8] M. Reisslein and K.W. Ross. High-Performance Prefetching Protocols for VBR Prerecorded Video. *IEEE Network, Vol. 12, num. 6*, November/December 1998.

[9] S. Bakiras and V. O. K. Li. Smoothing and prefetching video from distributed servers. *Proceedings of IEEE International Conference on Networking Protocols (ICNP), Toronto, Canada*, October 1999.

[10] Sen Subhabrata, Jennifer Rexford and Don Towsley. Proxy Prefix Caching for Multimedia Streams. *CMPSCI Technical Report 98-27, University of Massachusetts*, 1998.

[11] Sen Subhabratar, Jennifer Rexford and Don Towsley. Proxy Prefix Caching for Multimedia Streams. *Proceedings of IEEE Infocom, New York, NY*, April 1999.

[12] Yuewei wang et al. A Network-Conscious Approach to End-to-End Video delivary over Wide Area Networks Using Proxy Servers. *Proceedings of IEEE Infocom, San Francisco, California*, April 1998.

[13] W. Feng, F. Jahanian and S. Sechrest. Optimal buffering for the delivery of compressed video. *ACM Multimedia Systems Journal, pp. 297-309*, September 1997.

[14] James D. Salehi, Zhi-Li Zhang, James F. Kurose and Don Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. *IEEE/ACM Transactions on Networking, Volume: 6 Issue: 4*, August 1998.

[15] W. Feng. Rate-constrainted bandwidth smoothing for the delivery of stored video. *Proceedings of IS&T/SPIE Multimedia Networking and Computing, pp. 58-66*, February 1997.

[16] J. Zhang and J. Hui. Traffic characteristics and smoothness criteria in VBR video transmission. *Proceeding of IEEE Conference on Multimedia Computing and Systems*, June 1997.

[17] Jennifer Rexford and Don Towsley. Smoothing Variable Bit Rate Video in an Iinterwork. *IEEE/ACM Transactions on Networking, Vol. 7, No. 2*, April 1999.

[18] Wu chi Feng, Ming Liu, Brijesh Krishnaswami and Arvind Prabhudev. A Priority-Based Technique for the Best-Effort Delivery of Stored Video. *SPIE/IS&T Multimedia Computing and Networking 1999, San Jose, California*, January 1999.

[19] Lixia Zhang. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. *ACM Computer Communications Review, pp. 19-29*, September 1990.