Dynamic Window-Constrained Scheduling of Real-Time Streams in Media Servers

Richard West, *Member*, *IEEE*, Yuting Zhang,
Karsten Schwan, *Senior Member*, *IEEE*, and Christian Poellabauer, *Student Member*, *IEEE*

Abstract—This paper describes an algorithm for scheduling packets in real-time multimedia data streams. Common to these classes of data streams are service constraints in terms of bandwidth and delay. However, it is typical for real-time multimedia streams to tolerate bounded delay variations and, in some cases, finite losses of packets. We have therefore developed a scheduling algorithm that assumes streams have *window-constraints* on groups of consecutive packet deadlines. A window-constraint defines the number of packet deadlines that can be missed (or, equivalently, must be met) in a window of deadlines for consecutive packets in a stream. Our algorithm, called Dynamic Window-Constrained Scheduling (DWCS), attempts to guarantee no more than x out of a *window* of y deadlines are missed for consecutive packets in real-time and multimedia streams. Using DWCS, the delay of service to real-time streams is bounded, even when the scheduler is overloaded. Moreover, DWCS is capable of ensuring independent delay bounds on streams, while, at the same time, guaranteeing minimum bandwidth utilizations over tunable and finite windows of time. We show the conditions under which the total demand for bandwidth by a set of window-constrained streams can exceed 100 percent and still ensure all window-constraints are met. In fact, we show how it is possible to strategically skip certain deadlines in overload conditions, yet fully utilize all available link capacity and guarantee worst-case per-stream bandwidth and delay constraints. Finally, we compare DWCS to the "Distance-Based" Priority (DBP) algorithm, emphasizing the trade-offs of both approaches.

Index Terms—Real-time systems, multimedia, window-constraints, scheduling.

1 Introduction

Low latency, high bandwidth integrated services networks have introduced opportunities for new applications such as video conferencing, telemedicine, virtual environments [8], [20], groupware [14], and distributed interactive simulations (DIS) [32]. Already, we have seen streaming multimedia applications (e.g., RealNetworks RealPlayer and Windows Media Player) that have soft real-time constraints become commonplace among Internet users. Moreover, advances in embedded systems and ad hoc computing have led to the development of large-scale distributed sensor networks (and applications), requiring data streams to be delivered from sensors to specific hosts [25], hand-held PDAs, and even actuators.

Many of the applications described above require strict performance (or *quality of service*) requirements on the information transferred across a network. Typically, these performance objectives are expressed as some function of throughput, delay, jitter, and loss-rate [11]. With many multimedia applications, such as video-on-demand or streamed audio, it is important that information is received and processed at an almost constant rate (e.g., 30 frames per second for video information). However, some packets comprising a video frame or audio sample can be lost or delayed beyond their deadlines, resulting in little or no noticeable degradation in the quality of playback at the receiver. Similarly, a data source can lose or delay a certain fraction of information during its transfer across a network as long as the receiver processes the received data to compensate for the lost or late packets. Consequently, loss-rate is an important performance measure for this category of applications. We define the term *loss-rate* [31] as the fraction of packets in a stream either received *later than allowed or not received at all* at the destination.

One of the problems with using loss-rate as a performance metric is that it does not describe when losses are allowed to occur. For most loss-tolerant applications, there is usually a restriction on the number of *consecutive* packet losses that are acceptable. For example, losing a series of consecutive packets from an audio stream might result in the loss of a complete section of audio, rather than merely a reduction in the signal-to-noise ratio. A suitable performance measure in this case is a *windowed loss-rate*, i.e., loss-rate constrained over a finite range, or *window*, of consecutive packets. More precisely, an application might tolerate x packet losses for every y arrivals at the various service points across a network. Any service discipline attempting to meet these requirements must ensure that the number of violations to the loss-tolerance specification is minimized (if not zero) across the whole stream.

This paper describes the real-time properties of Dynamic Window-Constrained Scheduling (DWCS), an algorithm that is suitable for packet scheduling in real-time media servers [38], [39]. DWCS is designed to explicitly service packet streams in accordance with their loss and delay

- *R. West and Y. Zhang are with the Computer Science Department, Boston University, 111 Cummington St., Boston, MA 02215. E-mail: {richwest, danazh}@cs.bu.edu.*
- *K. Schwan and C. Poellabauer are with the College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., Atlanta, GA 30332-0280. E-mail: {schwan, chris}@cc.gatech.edu.*

Manuscript received 21 Feb. 2003; revised 17 Oct. 2003; accepted 13 Jan. 2004.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118330.

constraints, using just two attributes per stream. It is intended to support multimedia traffic streams in the same manner as the SMART scheduler [27], but DWCS is less complex and requires maintenance of less state information than SMART.

DWCS is closely related to weakly hard algorithms [5], [6] and those that attempt to guarantee at least m out of k packet deadlines for each and every stream [15]. Unlike other approaches, DWCS is capable of fully utilizing all available resources to guarantee at least m out of k deadlines are met (or, equivalently, no more than x out of y deadlines are missed) per stream. That is, in situations where it is impossible to meet all deadlines due to resource demands exceeding availability, DWCS can strategically skip some deadlines and still meet service constraints over finite windows of deadlines, while using all available resources if necessary.

1.1 Contributions

The significant contributions of this work include the description and analysis of an *online* version of DWCS that is capable of fully utilizing 100 percent of resources to meet per-stream window-constraints. We show how DWCS is capable of ensuring the delay bound of any given stream is independent of other streams and is finite even in overload situations. This work focuses on the characteristics of DWCS from the point of view of a *single* server, rather than a multihop network of servers.

As will be shown in Section 4.5, DWCS is a flexible algorithm, supporting many modes of operation that include not only window-constrained service, but also earliest deadline first, static priority, and fair scheduling. While DWCS is primarily intended for use in end hosts (i.e., media servers), we believe it is efficient enough to work in network access points or even programmable switches. In fact, we have shown in prior work how DWCS can be efficiently implemented in Intel i960-based I20 network interface cards to support packet scheduling at Gigabit wire-speeds [22], [21]. Similarly, we have implemented DWCS as a CPU scheduler in the Linux kernel, where it has been shown to be effective at meeting window-constraints on periodic real-time threads [23], [36]. In the latter case, DWCS successfully serviced CPU and I/O-bound threads 99 percent of the time even when the scheduler was fully loaded and the rest of the Linux kernel was left essentially non-real-time.

The remainder of this paper is organized as follows: Section 2 describes related work. The scheduling problem is then defined in Section 3, which includes a detailed description of Dynamic Window-Constrained Scheduling. Section 4 analyzes the performance of DWCS, including the bounds on service delay for competing streams and constraints under which real-time service guarantees can be made. Section 5 compares the performance of DWCS to the well-known "Distance-Based" Priority (DBP) algorithm [15] for a number of simulations. Finally, conclusions are described in Section 6.

2 Related Work

Hamdaoui and Ramanathan [15] were the first to introduce the notion of (m,k)-firm deadlines in which statistical service guarantees are applied to activities such as packet streams or periodic tasks. Their algorithm uses a "distance-based" priority scheme to increase the priority of an activity in danger of missing more than m deadlines over a window of k requests for service. This is similar to the concept of "skip-over" by Koren and Shasha [19], but, in some cases, skip-over algorithms unnecessarily skip service to one or more activities, even if it is possible to meet the deadlines of those activities.

By contrast, Bernat and Burns [4] schedule activities with (m,k)-*hard* deadlines, but their approach requires such hard temporal constraints to be guaranteed by offline feasibility tests. Moreover, Bernat and Burns' work focuses less on the issue of providing a solution to online scheduling of activities with (m,k)-hard deadlines, but more on the support for fast response time to best-effort activities, in the presence of activities with hard deadline constraints.

Pinwheel scheduling [17], [9], [2] is also similar to DWCS. With pinwheel scheduling, resources are allocated in fixed-sized time slots. For a given set of n activities, each activity $a_i \mid 1 \leq i \leq n$ requires service in at least m_i out of k_i consecutive slots. To draw an analogy with window-constrained scheduling, a time slot can be thought of as the interval between a pair of deadlines for each and every activity, but only one activity can be serviced in a single slot. By comparison, generalized window-constrained scheduling allows each activity to have its own arbitrary request period that defines the time between consecutive deadlines and, in each request period, the corresponding activity may have its own service time requirement, as in rate-monotonic scheduling [24]. If an activity receives less than its specified service requirement in one request period, it is said to have missed a deadline. Different activities may have different service time requirements, request periods, and window-constraints on the number of consecutive deadlines that can be missed (or, equivalently, met). Based on this information, DWCS is capable of producing a feasible pinwheel-compatible schedule, over *finite* windows of deadlines, when 100 percent of available resources (such as bandwidth) are utilized. By comparison, Baruah and Lin [2] have developed a pinwheel scheduling algorithm that is capable of producing a feasible schedule when the utilization of resources approaches 100 percent, given that window sizes approach infinity.

Other notable work includes Jeffay and Goddard's Rate-Based Execution (RBE) model [18]. As will be seen in this paper, DWCS uses similar service parameters to those described in the RBE model. However, in the RBE model, activities are expected to be serviced with an average rate of x times every y time units and there is no notion of missing, or discarding, service requests.

By meeting window-constraints, DWCS can guarantee a minimum fraction of link bandwidth to each stream it services. This is possible for finite windows of time and, hence, deadlines. As a result, fair bandwidth allocation is possible with DWCS over tunable time intervals for each stream. In essence, this is similar to the manner in which fair queuing algorithms [10], [41], [12], [3], [13], [30], [34] attempt to provide proportional share service over the smallest time intervals,[1] thereby approximating the Generalized Processor Sharing model [29]. However, DWCS differs in its explicit support for resource-sharing guarantees over specific time windows.

It is worth noting that DWCS adjusts the importance of servicing a stream by a function of the number of missed packet deadlines in a given window of consecutive deadlines. This is similar to the dynamic priority approach used in Distance-Based Priority scheduling [15]. In contrast, other researchers have developed static priority approaches that provide statistical guarantees on a periodic activity meeting a subset of all deadlines in situations where service times may vary and meeting all deadlines is impossible. For example, Atlas and Bestavros developed an algorithm called Statistical Rate Monotonic Scheduling [1], but this approach makes no explicit attempt to make service guarantees over a specific window of deadlines.

3 Dynamic Window-Constrained Scheduling (DWCS)

This section describes the DWCS algorithm for providing window-constrained service to real-time streams. Before we describe how DWCS works, we must precisely define the requirements for a feasible schedule. In doing so, we begin by clarifying the relationship between packet service times and scheduling granularity. Observe that the service time of a packet is a function of its length (in bits) and service rate (in bits per second), due to server capacity, or link bandwidth (whichever is limiting). If we assume the scheduler has the capacity to process packets fast enough to saturate a network link and link bandwidth is constant, then all packets will have the same service time if they have the same length. However, if packets vary in length or if the server capacity fluctuates (either due to variations in link bandwidth or variations in the service rate due to scheduling latencies associated with supporting different numbers of streams), then packet service times can be variable. In such circumstances, if it is possible to impose an upper bound on the worst-case service time of each and every packet, then DWCS can guarantee that no more than x packet deadlines are missed every y requests.

Note that, for these service guarantees to be made with DWCS, resources are allocated at the granularity of one *time slot* (see Fig. 1), where the size of a time slot is typically determined by the (worst-case) service time of the largest packet in any stream requiring service. Therefore, it is assumed that, when scheduling packets from a chosen stream, at least one packet in that stream is serviced in a time slot and no other packet (or packets) from any other stream can be serviced until the start of the next time slot. Unless stated otherwise, we assume throughout this paper that at most one packet from any given stream is serviced in a single time slot, but, in general, it is possible for multiple packets from the same stream to be aggregated together and serviced in a single time slot as if they were one large packet.

1. Actually, the granularity of the largest packet service time.

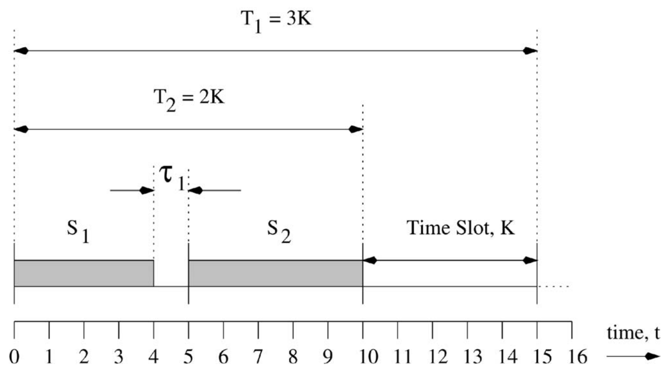


Fig. 1. Example of two packets from different streams, S_1 and S_2 being serviced in their respective time slots. Each time slot is of constant size K. Observe that the packet in S_1 requires $K - \tau_1$ service time, thereby wasting τ_1 time units before the packet in S_2 is serviced. In this example, S_1 has a request period of three time slots, while S_2 has a request period of two time slots.

3.1 Problem Definition

In order to define the real-time scheduling problem addressed as part of this paper, we introduce the following definitions, after which we describe the DWCS algorithm in more detail.

Bandwidth Utilization. This is a measure of the fraction (or percentage) of available bandwidth used by streams to meet their service constraints. A series of streams is said to *fully utilize* [24] available bandwidth, B, if all streams using B satisfy their service constraints and any increase in the use of B violates the service constraints of one or more streams.

Dynamic Window-Constrained Scheduling (DWCS). DWCS is an algorithm for scheduling packet streams, each having a set of service constraints that include a *request period* and *window-constraint*, as follows:

- *Request Period*—A request period, T_i, for a packet stream, S_i, is the interval between the deadlines of consecutive pairs of packets in S_i. Observe that the end of a request period, T_i, determines a *deadline* by which a packet in stream S_i must be serviced. If we consider all request periods begin from time, $t = 0$, the first deadline of S_i is $d_{i,1} = T_i$, while the mth deadline is $d_{i,m} = m.T_i$.
- *Window-Constraint*—This is specified as a value $W_i = x_i/y_i$, where the window-numerator, x_i, is the number of packets that can be lost or transmitted late for every fixed *window*, y_i (the window-denominator), of consecutive packet arrivals in the same stream, S_i. Hence, for every y_i packet arrivals in stream S_i, a minimum of $y_i - x_i$ packets must be scheduled for service by their deadlines; otherwise, a service violation occurs. At any time, all packets in the same stream, S_i, have the same window-constraint, W_i, while each successive packet in a stream, S_i, has a deadline that is offset by a fixed amount, T_i, from its predecessor. After servicing a packet from S_i, the scheduler adjusts the window-constraint of S_i and all other streams whose head packets have just missed their deadlines due to servicing S_i. Consequently, a stream S_i's *original* window-constraint, W_i, can differ

TABLE 1
Precedence among Pairs of Packets in Different Streams

Pairwise Packet Ordering
Earliest deadline first (EDF)
Equal deadlines, order lowest window-constraint first
Equal deadlines and zero window-constraints, order highest window-denominator first
Equal deadlines and equal non-zero window-constraints, order lowest window-numerator first
All other cases: first-come-first-serve

The precedence rules are applied top-to-bottom in the table.

from its *current* window-constraint, W_i'. Observe that a stream's window-constraint can also be thought of as a *loss-tolerance*.

Stream Characterization. A stream S_i is characterized by a 3-tuple (C_i, T_i, W_i), where C_i is the service time for a packet in stream S_i. This assumes all packets in S_i have the same service time or C_i is the worst-case service time of the longest packet in S_i. For the purposes of this paper, where time is divided into fixed-sized slots, each and every packet can be serviced in one such slot. However, the general DWCS algorithm does not require service (and, hence, scheduling) at the granularity of fixed-sized time slots. The concept of scheduling in fixed-sized time slots is used only to enforce predictable service guarantees with DWCS.

Feasibility. A schedule, comprised of a sequence of streams, is feasible if no original window-constraint of any stream is ever violated. DWCS attempts to schedule all packet streams to meet as many window-constraints as possible.

Problem Statement. The problem is to produce a feasible schedule using an online algorithm. The algorithm should attempt to maximize network bandwidth. In fact, we show in Section 4 that, under certain conditions, Dynamic Window-Constrained Scheduling can guarantee a feasible schedule as long as the *minimum* aggregate bandwidth utilization of a set of streams does not exceed 100 percent of available bandwidth. This implies it is possible to have a feasible schedule, even in overload conditions, whereby insufficient server capacity (or link bandwidth) exists to guarantee all packet deadlines.

3.2 The DWCS Algorithm

DWCS orders packets for service based on the values of their *current* window-constraints and deadlines, where each deadline is derived from the current time and the request period. Precedence is given to packets in streams according to the rules shown in Table 1. This table of precedence rules differs from the original table used in earlier versions of DWCS [38], [39]. The basic difference is that the top two lines in the table are reversed: The original table *first* compares packets based on their streams' current window-constraints, giving precedence to the packet in the stream with the lowest (numeric-valued) window-constraint. If there are ties, the packet with the earliest deadline is chosen. This approach works well for situations when packets in different streams rarely have the same deadlines due to working in real-time at a given clock resolution. Unfortunately, in underload situations, earliest deadline first (EDF) scheduling is often more likely to meet deadlines and, hence, window-constraints. Notwithstanding, the original DWCS algorithm is still better than EDF in overload cases where it is impossible to meet all deadlines.

The desirable property of EDF, that all deadlines can be met as long as the load does not exceed 100 percent [24], is the motivation for revising the table of precedence rules. However, since DWCS is table-driven, it is easy to change the table of precedence rules to finetune the characteristics of the algorithm. In this paper, we require that all packet deadlines are aligned on time slot boundaries, thereby forcing comparison of current window-constraints only if two packets have equal deadlines when competing for the same time slot. As stated earlier, we assume that scheduling decisions are made once every time slot. This is merely for analysis purposes in subsequent sections and not a requirement of the algorithm in general.

Now, whenever a packet in S_i misses its deadline, the window-constraint for all subsequent packets in S_i is adjusted to reflect the increased importance of servicing S_i. This approach avoids starving the service granted to a given stream and attempts to increase the importance of servicing any stream likely to violate its original window-constraint. Conversely, any packet in a stream serviced before its deadline causes the window-constraint of any subsequent packets in the same stream (yet to be serviced) to be increased, thereby reducing their priority.

The window-constraint of a stream changes over time, depending on whether or not another (earlier) packet from the same stream has been serviced by its deadline. If a packet cannot be serviced by its deadline, it is either transmitted late or it is dropped and the next packet in the stream is assigned a deadline corresponding to the latest time it must complete service.

It should be clear that DWCS combines elements of EDF and static priority scheduling to result in a *dynamic* priority algorithm. Observe that EDF scheduling considers each packet's importance (or priority) increases as the urgency of completing that packet's service increases. By contrast, static priority algorithms all consider that one packet is more important to service than another packet, based solely on each packet's time-invariant priority. DWCS combines both the properties of static priority and earliest deadline first scheduling by considering each packet's individual importance when the urgency of servicing two or more packets is the same. That is, if two packets have the same deadline, DWCS services the packet which is more important according to its current window-constraint. In practice, it makes sense to set packet deadlines in different streams to be some multiple of a, possibly worst-case, packet service time. This increases the likelihood of multiple head packets of different streams having the same deadlines.

Notice from Table 1 that packets are ordinarily serviced in earliest deadline first order. Let the deadline of the head packet in S_i be $d_{i,head}$ and the deadline of the mth subsequent packet be $d_{i,head} + m.T_i$. If at least two streams have head packets with equal deadlines, the packet from stream S_i with the lowest *current* window-constraint W_i' is serviced first. If $W_i' = W_j' > 0$ and $d_{i,head} = d_{j,head}$ for S_i and

```
if (y'_i > x'_i) then y'_i = y'_i - 1;
else if (y'_i = x'_i) and (x'_i > 0) then
    x'_i = x'_i - 1; y'_i = y'_i - 1;
if (x'_i = y'_i = 0) or (S_i is tagged) then
    x'_i = x_i; y'_i = y_i;
if (S_i is tagged) then reset tag;
```

Fig. 2. Window-constraint adjustment for a packet in S_i serviced before its deadline.

S_j, respectively, S_i and S_j are ordered such that a packet from the stream with the lowest window-numerator is serviced first. By ordering based on the lowest window-numerator, precedence is given to the packet with *tighter* window-constraints since fewer consecutive late or lost packets from the same stream can be tolerated. Likewise, if two streams have zero-valued current window-constraints and equal deadlines, the packet in the stream with the highest window-denominator is serviced first. All other situations are serviced in a first-come-first-serve manner.

We now describe how a stream's window-constraints are adjusted. As part of this approach, a *tag* is associated with each stream S_i to denote whether or not S_i has violated its window-constraint W_i at the current point in time. In what follows, let S_i's *original* window-constraint be $W_i = x_i/y_i$, where x_i is the original window-numerator and y_i is the original denominator. Likewise, let $W'_i = x'_i/y'_i$ denote the *current* window-constraint. Before a packet in S_i is serviced, $W'_i = W_i$. Upon servicing a packet in S_i before its deadline, W'_i is adjusted for subsequent packets in S_i, as shown in Fig. 2. This adjustment policy is only applied to *time-constrained* streams, whose packets have deadlines. For non-time-constrained streams, window-constraints remain constant and serve as static priorities.

At this point in time, the window-constraint, W_j, of any other stream, $S_j \mid j{\neq}i$, comprised of one or more late packets, is adjusted as shown in Fig. 3. In the absence of a feasibility test, it is possible that window-constraint violations can occur. A violation actually occurs when $W'_j = x'_j/y'_j \mid x'_j = 0$ and another packet in S_j then misses its deadline. Before S_j is serviced, x'_j remains zero, while y'_j is increased by a constant, ϵ, every time a packet in S_j misses a deadline. The exception to this rule is when $y_j = 0$ (and, more specifically, $W_j = 0/0$). This special case allows DWCS to *always* service streams in EDF order, if such a service policy is desired.

If S_j violates its original window-constraint, it is tagged for when it is next serviced. Tagging ensures that a stream is never starved of service, even in overload. Later, Theorem 2 shows the delay bound for a stream which is tagged with window-constraint violations. Consequently, S_j is assured of service since it will eventually take precedence over all streams with a zero-valued current window-constraint.

```
if (x'_j > 0) then
    x'_j = x'_j - 1; y'_j = y'_j - 1;
    if (x'_j = y'_j = 0) then x'_j = x_j; y'_j = y_j;
else if (x'_j = 0) and (y_j > 0) then
    y'_j = y'_j + ε;
    Tag S_j with a violation;
```

Fig. 3. Window-constraint adjustment when a packet in $S_j \mid j \neq i$ misses its deadline.

```
while (TRUE) {
    for (each packet in all streams eligible for service at the current time, t)
        find the next packet in stream, S_i, with the highest priority,
        according to the rules in Table 1;
    service next packet in S_i;
    adjust W'_i according to rules in Figure 2;
    /* Adjust deadline of next packet in S_i. */
    d_{i,head} = d_{i,head} + T_i;
    for (each packet in S_j | j≠i, missing its deadline) {
        while (deadline missed) {
            adjust W'_j according to rules in Figure 3;
            if (current packet can be dropped) {
                drop current packet in S_j;
            }
            /* Adjust deadline of current packet in S_j
                by adding T_j to the current deadline. */
            d_{j,head} = d_{j,head} + T_j;
        }
    }
}
```

Fig. 4. The DWCS algorithm.

Consider the case when S_i and S_j both have current window-constraints, W'_i and W'_j, respectively, such that $W'_i = 0/y'_i$ and $W'_j = 0/y'_j$. Even if both deadlines, $d_{i,head}$ and $d_{j,head}$, are equal, precedence is given to the stream with the highest window-denominator. Suppose that S_i is serviced before S_j because $y'_i > y'_j$. At some later point in time, S_j will have the highest window-denominator since its denominator is increased by ϵ every request period, T_j, that a packet in S_j is delayed, while S_i's window-constraint is reset once it is serviced. For simplicity, we assume every stream has the same value of ϵ, but, in practice, it may be beneficial for each stream S_i to have its own value, ϵ_i, to increase its need for service at a rate independent of other streams, even when window-constraint violations occur. Unless stated otherwise, $\epsilon = 1$ is used throughout the rest of this paper.

We can now show the pseudocode for DWCS in Fig. 4. Usually, a stream is eligible for service if a packet in that stream has not yet been serviced in the current request-period, which is the time between the deadline of the previous packet and the deadline of the current packet in the same stream. That is, no more than one packet in a given stream is typically serviced in a single request period and the packet must be serviced by the end of its request period to prevent a deadline being missed. However, DWCS allows streams to be marked as eligible for scheduling multiple times in the same request period. This is essentially to provide work-conserving service to a nonempty queue of packets in one stream when there are no other streams awaiting service in their current request periods. That said, for the purposes of the analysis in this paper, we assume packets arrive (or are marked eligible) for service at a rate of one every corresponding request period.

To complete this section, Fig. 5 shows an example schedule using both DWCS and EDF for three streams, S_1,

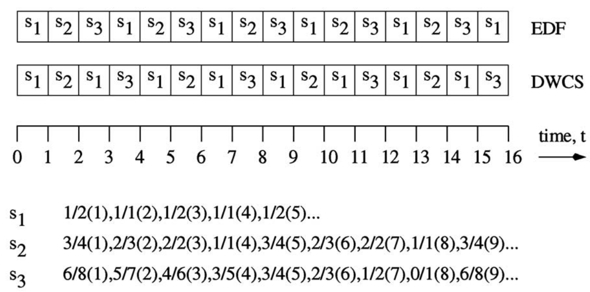


Fig. 5. Example showing the scheduling of three streams, S_1, S_2, and S_3, using EDF and DWCS. All packets in each stream have unit service times and request periods. The window-constraints for each stream are shown as fractions, x/y, while packet deadlines are shown in brackets.

S_2, and S_3. For simplicity, assume that every time a packet in one stream is serviced, another packet in the same stream requires service. It is left to the reader to verify the scheduling order for DWCS. In this example, DWCS guarantees that all window-constraints are met over nonoverlapping windows of y_i deadlines (for each stream, S_i) and no time slots are unused. Moreover, the three streams are serviced in proportion to their original window-constraints and request periods. Consequently, S_1 is serviced twice as much as S_2 and S_3 over the interval $t = [0, 16]$. By contrast, EDF arbitrarily schedules packets with equal deadlines, irrespective of which packet is from the more critical stream in terms of its window-constraint. In this example, EDF selects packets with equal deadlines in strict alternation, but the window-constraints of the streams are not guaranteed.

Note that EDF scheduling is optimal in the sense that if it is possible to produce a schedule in which all deadlines are met, such a schedule can be produced using EDF. Consequently, if C_i is the service time for a packet in stream S_i, then, if $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.0$, all deadlines will be met using EDF [24]. However, in this example, $\sum_{i=1}^{n} \frac{C_i}{T_i} = 3.0$, so not all deadlines can be met. Since, $\sum_{i=1}^{n} \frac{(1-W_i)C_i}{T_i} = 1.0$, it is possible to strategically miss deadlines for certain packets and thereby guarantee the window-constraints of each stream. By considering window-constraints when deadlines are tied, DWCS is able to make guarantees that EDF cannot, even in overload.

3.3 DWCS Complexity

DWCS's time complexity is divided into two parts: 1) the cost of *selecting* the next packet according to the precedence rules in Table 1 and 2) the cost of *adjusting* stream window-constraints and packet deadlines *after* servicing a packet. Using heap data structures for prioritizing packets, the cost of selecting the next packet for service is $O(log(n))$, where n is the number of streams awaiting service. However, after servicing a packet, it may be necessary to adjust the deadlines of the head packets, and window-constraints, of all n queued streams. This is the case when all $n-1$ streams (other than the one just serviced) have packets that miss their current deadlines. This can lead to a worst-case complexity for DWCS of $O(n)$. However, the average case performance is typically a logarithmic function of the number of streams (depending on the data structures used for maintaining scheduler state) because not all streams always need to have their window-constraints adjusted after a packet in any given stream is serviced. When only a *constant* number of packets in different streams miss their deadlines after servicing some other packet, a heap data structure can be used to determine those packet deadlines and stream window-constraints that need to be adjusted. It follows that a constant number of updates to service constraints using heaps, as described in an earlier paper [39], requires $O(log(n))$ operations. Additionally, there is an $O(1)$ cost per stream to update the corresponding service constraints, *after servicing a packet*.

In reality, the costs associated with DWCS compare favorably to those of many fair queuing, pinwheel, and weakly hard algorithms. Observe that, with fair queuing algorithms, the time complexity consists of: 1) the cost of calculating a per packet virtual time, $v(t)$, *upon packet arrival* at the input to the scheduler, which is then used to derive an ordering *tag* (typically a packet start or finish tag) and 2) the cost of determining the next packet for service based on each packet's tag. The cost of part 2) is the same as the cost of selecting the next packet for service in DWCS and can be implemented in $O(log(n))$ time using a heap. The calculation of the virtual time, $v(t)$, in part 1), is $O(n)$ in WFQ since it is a function of all backlogged sessions (i.e., streams) at time t.

We acknowledge that algorithms such as Start-time Fair Queuing (SFQ) [13], [30], Self-Clocked Fair Queuing (SCFQ) [12], or Frame-Based Fair Queuing (FFQ) and Starting-Potential Fair Queuing (SPFQ) [33], have an $O(1)$ complexity for calculating virtual timestamps (and ordering tags) *per packet*, making their overall costs $O(log(n))$ per packet. However, these algorithms typically suffer increased packet delays. It is also worth noting that Xu and Lipton [40] showed that the lower bound on algorithmic complexity for fair queuing to guarantee $O(1)$ "GPS-relative" delay guarantees [29] is $O(log(n))$, *discounting the cost of calculating per packet virtual times.*

As stated earlier in Section 1, DWCS has more in common with pinwheel [17], [9], [2] and weakly hard [5], [6], [15], [19] real-time schedulers than fair queuing algorithms, although DWCS can provide fairness guarantees. Most variants of these algorithms have time complexities that are no better than that of DWCS. Irrespective of the asymptotic scheduling costs, DWCS has been shown to be an effective scheduler for packet transmission at Gigabit wire-speeds, by overlapping the comparison and updating of stream service constraints with the transmission of packets [22], [21].

The per-stream state requirements of DWCS include the head packet's deadline (computed from a stream's request period and the current time), a stream's window-constraint, and a single-bit violation tag. Due to the time and space requirements of DWCS, we feel it is possible to implement the algorithm at network access points and, possibly, within switches too. In fact, we have demonstrated the efficiency of DWCS by implementation in firmware on $I20$ network interface cards (NICs) [22], [21].

Other work [39] shows how DWCS can be approximated, to further reduce its scheduling latency, thereby improving service scalability [35] at the cost of potentially violating some service constraints. Moreover, it may be appropriate to combine multiple streams into one session, with DWCS used to service the aggregate session. Such an approach would reduce the scheduling state requirements and increase scalability. In fact, this is the approach taken in our simulation experiments in Section 5.

4 Analysis of DWCS

In this section, we show the following important characteristics of the DWCS algorithm, as defined in this paper:

- DWCS ensures that the maximum delay of service, incurred by a real-time stream enqueued at a single server, is bounded even in overload. The exact value of this maximum delay is characterized below.
- In specific overload situations, DWCS can guarantee a feasible schedule by strategically skipping deadlines of packets in different streams.
- A simple online feasibility test for DWCS exists, assuming each stream is serviced at the granularity of a fixed-sized time slot and all request periods are multiples of such a time slot (see Fig. 1). A time slot can be thought of as the time to service one or more packets from any given stream and no two streams can be serviced in the same time slot. For simplicity, we assume that at most one packet from any given stream is serviced in a single time slot. Consequently, if the *minimum* aggregate bandwidth requirement of all real-time streams does not exceed the total available bandwidth, then a feasible schedule is possible using DWCS.
- For networks with fixed-length packets, a time slot is at the granularity of the service time of one packet. However, for variable rate servers or in networks where packets have variable lengths, the service times can vary for different packets. In such circumstances, if it is possible to impose an upper bound on the worst-case service time of each and every packet, then DWCS can still guarantee that no more than x packet deadlines are missed every y requests. In this case, service is granted to streams at the granularity of a time slot, which represents the worst-case service time of any packet. Alternatively, if it is possible to fragment variable-length packets and later reassemble them at the destination, per-stream service requirements can be translated and applied to fixed-length packets with constant service times, representing a time slot in a DWCS-based system.
- Apart from providing window-constrained guarantees, DWCS can behave as an EDF, static priority, or fair scheduling algorithm.

4.1 Delay Characteristics

Theorem 1. *If a feasible schedule exists, the maximum delay of service to a stream,*[2] $S_i \mid 1 \leq i \leq n$, *is at most* $(x_i+1)T_i - C_i$, *where* C_i *is the service time for one packet in* S_i.[3]

2. The "maximum delay" is that imposed by a single server and is not the maximum end-to-end delay across a network.

3. For simplicity, we assume all packets in the same stream have the same service time. However, unless stated otherwise, this constraint is not binding and the properties of DWCS should still hold.

Proof. Every time a packet in S_i misses its deadline, x'_i is decreased by 1 until x'_i reaches 0. A packet misses its deadline if it is delayed by T_i time units without service. Observe that, at all times, $x'_i \leq x_i$. Therefore, service to S_i can be delayed by at most x_iT_i until $W'_i = 0$. If S_i is delayed more than another $T_i - C_i$ time units, a window-constraint violation will occur since service of the next packet in S_i will not complete by the end of its request period, T_i. Hence, S_i must be delayed at most $(x_i+1)T_i - C_i$ if a feasible schedule exists. □

We now characterize the delay bound for a stream when window-constraint violations occur, assuming all request periods are greater than or equal to each and every packet's service time. That is, $T_i \geq C_i, x_i \geq 0, y_i > 0, \forall i \mid 1 \leq i \leq n$.

Theorem 2. *If window-constraint violations occur, the maximum delay of service to* S_i *is no more than*

$$T_i(x_i + y_{max} + n - 1) + C_{max},$$

where $y_{max} = max[y_1, \cdots, y_n]$ *and* C_{max} *is the maximum packet service time among all queued packets.*

Proof. The details of this proof are shown in Appendix A.□

If $T_i \rightarrow \infty$, then S_i experiences unbounded delay in the worst case. This is the same problem with static-priority scheduling since a higher priority stream will always be serviced before a lower priority stream. Observe that, in calculating the worst-case delay experienced by S_i, it is assumed that $dy'_i/dt = \epsilon/T_i|\epsilon = 1$ (see Fig. 10). If $\epsilon>1$ or there is a unique value, $\epsilon_i>1$ for each stream S_i, then the worst-case delay experienced by S_i is $\frac{T_i(x_i+y_{max}+n-1)}{\epsilon_i} + C_{max}$. If $\epsilon_i = (x_i + y_{max} + n - 1)$, then the worst-case delay of S_i is $T_i + C_{max}$, which is independent of the number of streams. Consequently, the worst-case delay of service to each stream can be made to be independent of all other streams, even in overload situations.

4.2 Bandwidth Utilization

As stated earlier, $W_i = x_i/y_i$ for stream S_i. Therefore, a minimum of $y_i - x_i$ packets in S_i must be serviced in every window of y_i consecutive packets, for S_i to satisfy its window-constraints. Since one packet is required to be serviced every request period, T_i, to avoid any packets in S_i being late, a minimum of $y_i - x_i$ packets must be serviced every y_iT_i time units. Therefore, if each packet takes C_i time units to be serviced, then y_i packets in S_i require at least $(y_i - x_i)C_i$ units of service time every y_iT_i time units. For a stream, S_i, with request period, T_i, the *minimum* utilization factor is $U_i = \frac{(y_i - x_i)C_i}{y_iT_i}$, which is the minimum required fraction of available service capacity and, hence, bandwidth by consecutive packets in S_i. Hence, the utilization factor for n streams is at least $U = \sum_{i=1}^{n} \frac{(1-W_i)C_i}{T_i}$. Furthermore, the *least upper bound* on the utilization factor is the minimum of the utilization factors for all streams that fully utilize all available bandwidth [24]. If U exceeds the least upper bound on bandwidth utilization, a feasible schedule is not guaranteed. In fact, it is necessary that $U \leq 1.0$ is true for a feasible schedule, using any scheduling policy.

Mok and Wang extended our original work by showing that the *general* window-constrained problem is NP-hard for arbitrary service times and request periods [26]. The general window-constrained scheduling problem can be defined in terms of n streams each characterized by a 3-tuple $(C_i, T_i, W_i = x_i/y_i)$ having arbitrary values. However, DWCS guarantees that no more than x_i deadlines are missed out of y_i deadlines for n streams, if $U = \sum_{i=1}^{n} \frac{(1-x_i/y_i)C_i}{T_i} \leq 1.0$, given $1 \leq i \leq n$, $C_i = K$ and $T_i = qK$, where $q \in Z^+$,[4] K is a constant, and U is the minimum utilization factor for a feasible schedule.[5]

This implies a feasible schedule is possible even when the server capacity, or link bandwidth, is 100 percent utilized, given: 1) All packets have constant, or some known worst-case, service time and 2) all request periods are the same and are multiples of the constant, or worst-case, service time. Although this sounds restrictive, it offers the ability for a DWCS scheduler to proportionally share service among a set of n streams. Moreover, each stream, S_i, is guaranteed a minimum share of link bandwidth over a *specific* window of time, independent of the service provided to other streams. This contrasts with fair queuing algorithms that 1) attempt to share resources over the smallest window of time possible (thereby approximating the fluid-flow model) and 2) do not provide *explicit* isolation guarantees. In the latter case, the arrival of a stream at a server can affect the service provided to all other streams since proportional sharing is provided in a relative manner. For example, weighted fair queuing uses a weight, w_i, for each S_i such that S_i receives (approximately) $\frac{w_i}{\sum_{j=1}^{n} w_j}$ fraction of resources over a given window of time.

We now show the utilization bound for a specific set of streams in which each stream, S_i, is characterized by the 3-tuple $(C_i = K, T_i = qK, W_i)$. In what follows, we consider the maximum number of streams, n_{max}, that can guarantee a feasible schedule. It can be shown that, for all values of n, where $n < n_{max}$, a feasible schedule is always guaranteed if one is guaranteed for n_{max} streams.

Lemma 1. *Consider a set of* n *streams,* $\Gamma = \{S_1, \cdots, S_n\}$, *where* $S_i \in \Gamma$ *is defined by the 3-tuple*

$$(C_i = K, T_i = qK, W_i = x_i/y_i).$$

If the utilization factor, $U = \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1.0$, *then* $x_i = y_i - 1$ *maximizes* n.

Proof. Without loss of generality, we can assume $K = 1$. Further, for all nontrivial situations, n must be greater than q, otherwise we can always find a unit-length slot in any fixed interval of size q to service each stream at least once. Now, for any window-constraint, x_i/y_i, we can assume $x_i < y_i$ since, if $x_i = y_i$, then no deadlines need to be met for the corresponding stream, S_i. Consequently, for arbitrary S_i, $y_i - x_i \geq 1$.

Therefore, if we let $y_k = max(y_1, y_2, \cdots, y_n)$, it must be that $n \leq qy_k$ since:

$$\frac{n}{qy_k} = \sum_{i=1}^{n} \frac{1}{qy_k} \leq \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_k} \leq \sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1$$
$$\Rightarrow n \leq qy_k.$$

If all window-constraints are equal, for each and every stream, we have the following:

$$\sum_{i=1}^{n} \frac{(y_i - x_i)}{qy_i} \leq 1 \Rightarrow \frac{n(y_i - x_i)}{qy_i} \leq 1$$
$$\Rightarrow n \leq \frac{qy_i}{y_i - x_i} \leq qy_i$$

if $x_i = y_i - 1$, then $\frac{qy_i}{y_i - x_i} = qy_i$ and n is maximized. □

From Lemma 1, we now consider the conditions for a feasible schedule when each $S_i \in \Gamma$ is defined by the 3-tuple $(C_i = 1, T_i = q, W_i = x_i/y_i)$. If we envision Γ as a set of streams, each with infinite packets, we can define a *hyperperiod* in a similar fashion to that in periodic task scheduling. As with periodically occurring tasks, a stream with infinite packets can be seen to require service at regular intervals. The hyperperiod essentially defines a period in which a repeating schedule of service to all streams occurs. Let the hyperperiod, H, be $lcm(qy_1, qy_2, \cdots, qy_n)$. The following theorem can now be stated:

Theorem 3. *In each* nonoverlapping *window of size* q *in the hyperperiod,* H, *there cannot be more than* q *streams out of* n *with* current *window-constraint* $\frac{0}{y_i'}$ *at any time, when* $U = \sum_{i=1}^{n} \frac{y_i - x_i}{qy_i} \leq 1.0$.

Proof. The details of this proof are shown in Appendix B.□

We can now derive the least upper bound on bandwidth utilization, for the set Γ, in which each stream $S_i \in \Gamma$ is characterized by the 3-tuple $(C_i = K, T_i = qK, W_i = x_i/y_i)$.

Corollary 1. *Using DWCS, the least upper bound on the utilization factor is* 1.0, *for the set* Γ, *in which each stream* $S_i \in \Gamma$ *is characterized by the 3-tuple* $(C_i = K, T_i = qK, W_i = x_i/y_i)$.

From Theorem 3 (where $K = 1$ without loss of generality), there can never be more than q streams out of n with current window-constraint $\frac{0}{y_i'}$ when $U = \sum_{i=1}^{n} \frac{y_i - x_i}{qy_i} \leq 1.0$. For arbitrary values of K, Theorem 3 implies there can never be more than qK streams out of n with current window-constraint $\frac{0}{y_i'}$. These streams will be guaranteed service in preference to all streams with nonzero window-constraints since all streams are serviced in fixed-sized time slots, packet deadlines are aligned on time slot boundaries, and the assumption is that all streams have the same request periods. If all request periods are spaced qK time units apart, DWCS guarantees that each and every stream, S_i, with current window-constraint $\frac{0}{y_i'}$ is serviced without missing more than x_i deadlines in a window of y_i deadlines. Observe that, for S_i to have a current window-constraint $W_i' = \frac{0}{y_i'}$, exactly x_i deadlines have been missed in the current window of y_i deadlines.

4. Z^+ is the set of positive integers.

5. In the RTSS 2000 paper [37], we incorrectly stated $T_i = q_i K$. However, the utilization bound proven here and outlined in that paper holds for fixed q.

4.3 Fixed versus Sliding Windows

DWCS explicitly attempts to provide service guarantees over *fixed* windows of deadlines. That is, DWCS assumes the original window-constraint specified for each stream defines service requirements over nonoverlapping groups of deadlines. However, it is possible to determine a corresponding *sliding* window-constraint for a specified fixed window-constraint.

As stated earlier, DWCS tries to guarantee no more than x_i out of a fixed window of y_i deadlines are missed, for each stream S_i. This is the same as guaranteeing a minimum $m_i = y_i - x_i$ out of a fixed window of $k_i = y_i$ deadlines are met. It can be shown that, if no more than x_i deadlines are missed in a *fixed* window of y_i deadlines, then no more than $x_i^s = 2x_i$ deadlines are missed in a *sliding* window of $y_i^s = y_i + x_i$ deadlines. Likewise, this means that, by meeting m_i deadlines over fixed windows of size k_i, then it is possible to guarantee $m_i^s = m_i$ deadlines over sliding windows of size $k_i^s = 2k_i - m_i$. The proof is omitted for brevity.

Therefore, for any original window-constraint, $W_i = x_i/y_i = (k_i - m_i)/k_i$, we can determine a corresponding sliding window-constraint,

$$W_i^s = x_i^s/y_i^s = 2(k_i - m_i)/(2k_i - m_i).$$

We will use this relationship between fixed and sliding windows when comparing DWCS to Distance-Based Priority Scheduling in Section 5.

4.4 Supporting Packets with Variable Service Times

For variable rate servers, or in networks where packets have variable lengths, the service times can vary for different packets. In such circumstances, if it is possible to impose an upper bound on the *worst-case* service time of each and every packet, then DWCS can still guarantee that no more than x packet deadlines are missed every y requests. This implies that the scheduling granularity, K (i.e., one time slot), should be set to the worst-case service time of any packet scheduled for transmission. For situations where a packet's service time, C_i, is less than K (see Fig. 1), then a feasible schedule is still possible using DWCS, but the least upper bound on the utilization factor is less than 1.0. That is, if $\tau_i = K - C_i$, then the least upper bound on the utilization factor is $1.0 - \sum_{i=1}^{n} \frac{(1-W_i)\tau_i}{T_i}$.

Alternatively, if it is possible to fragment variable-length packets and later reassemble them at the destination, per-stream service requirements can be translated and applied to fixed-length packet fragments with constant service times. This is similar to the segmentation and reassembly (SAR) strategy employed in ATM networks—ATM networks have fixed-length (53 byte) cells and the SAR component of the ATM Adaptation Layer segments application-level packets into cells that are later reassembled. Consequently, the scheduling granularity, K, can be set to a time which is less than the worst-case service time of a packet.

For fragmented packets, it is possible to translate stream S_i's service constraints as follows: Let S_i have an original 3-tuple (C_i, T_i, W_i) and a translated 3-tuple

$$(C_i^* = K, T_i^* = qK, W_i^* = x_i^*/y_i^*),$$

where q and K are arbitrary positive integers. Then, x_i^* and y_i^* are the smallest values satisfying $x_i^*/y_i^* = 1 - \frac{q(1-W_i)C_i}{T_i}$.

Example. Consider three streams, S_1, S_2, and S_3, with the following constraints:

$$(C_1 = 3, T_1 = 5, W_1 = 2/3),$$
$$(C_2 = 4, T_2 = 6, W_2 = 23/35), \text{ and}$$
$$(C_3 = 5, T_3 = 7, W_3 = 1/5).$$

The total utilization factor is 1.0 in this example, but, due to the nonpreemptive nature of the variable-length packets, a feasible schedule cannot be constructed. However, if the packets are fragmented and the per-stream service constraints are translated as described above, assuming $q = K = 1$, we have:

$$(C_1^* = 1, T_1^* = 1, , W_1^* = 4/5),$$
$$(C_2^* = 1, T_2^* = 1, W_2^* = 27/35), \text{ and}$$
$$(C_3^* = 1, T_3^* = 1, W_3^* = 3/7),$$

then a feasible schedule exists. In the latter case, all fragments are serviced so that their corresponding stream's window-constraints are met. These translated window-constraints are equivalent to the original window-constraints, thereby guaranteeing each stream its exact share of bandwidth. Observe that $C_i^* = T_i^* = 1$ is the normalized time to service one fragment of a packet in S_i. This fragment could correspond to, e.g., a single cell in an ATM network, but, more realistically, it makes sense for one fragment to map to multiple such cells, thereby reducing the scheduling overheads per fragment. Similarly, a fragment might correspond to a maximum transmission unit in an Ethernet-based network.

4.5 Beyond Window-Constraints

In situations where it is not essential to guarantee all streams' window-constraints, DWCS still attempts to meet as many window-constraints as possible. It should be apparent that, for variable length packets, it is not possible to always guarantee a stream's window-constraints. This is because an arbitrarily long packet could require so much service time that a stream misses more than x consecutive deadlines. We shall show an example schedule for variable-length packets later in this section, but first we describe some of the additional behaviors of DWCS.

Earliest-Deadline First Scheduling using DWCS. When each and every stream, S_i, has a window-constraint set to $0/0$ (i.e., $x_i = 0$ and $y_i = 0$), DWCS degrades to EDF. Intuitively, this makes sense since all streams have the same importance, so their corresponding packets are serviced based upon the time remaining to their deadlines. It can be shown that, if all deadlines can be met, EDF guarantees to meet all deadlines [7]. If packets are dropped after missing their deadlines, EDF is optimal with respect to loss-rate in discrete-time G/D/1 and continuous-time M/D/1 queues [28].

Static Priority Scheduling using DWCS. If no packets in any streams have deadlines (i.e., they effectively have infinite deadlines), DWCS degrades to static priority (SP). Static-priority scheduling is optimal for a weighted mean delay objective, where weighted mean delay is a linear combination of the delays experienced by all packets [16]. In DWCS, the current window-constraints associated with each packet in every stream are always equal to their original window-constraints and each packet's window-constraint serves as its static priority. As expected, precedence is given to the packet with the lowest window-constraint (i.e., highest priority). For packets with infinite deadlines, DWCS has the ability to service non-time-constrained packets in static priority order to minimize weighted mean delay.

Fair Scheduling using DWCS. Fair Queuing derivatives share bandwidth among n streams in proportion to their weights. Specifically, let w_i be the weight of stream S_i and $B_i(t_1, t_2)$ be the aggregate service (in bits) of S_i in the interval $[t_1, t_2]$. If we consider two streams, S_i and S_j, the normalized service (by weight) received by each stream will be $\frac{B_i(t_1,t_2)}{w_i}$ and $\frac{B_j(t_1,t_2)}{w_j}$, respectively. The aim is to ensure that $|\frac{B_i(t_1,t_2)}{w_i} - \frac{B_j(t_1,t_2)}{w_j}|$ is as close to zero as possible, considering that packets are indivisible entities and an integer number of packets might not be serviced during the interval $[t_1, t_2]$.

DWCS also has the ability to meet weighted fair allocation of bandwidth. In this mode of operation, the original table of precedence rules [38] is more appropriate than the one in Table 1, unless we have fixed-sized time slots between scheduling invocations. The main difference with the original table of precedence rules is as follows: Packets are selected by first comparing their streams' window-constraints and only if there are ties are deadlines then compared. This has advantages for variable-length packets. For example, Fig. 6 shows an example of bandwidth allocation among two streams, S_1 and S_2, comprising packets of different lengths (i.e., $C_1 = 5$ and $C_2 = 3$). S_1 and S_2 each require 50 percent of the available bandwidth. The service times for each and every packet in streams S_1 and S_2 are five time units and three time units, respectively. Deadlines in this example are shown as *start* deadlines. Similarly, request periods for S_1 and S_2 are $T_1 = 5$ and $T_2 = 3$, respectively. In general, fair bandwidth allocation can be guaranteed over an interval that is the lowest-common-multiple of each value $y_i.T_i$.

Given stream weights, w_i, in a fair bandwidth-allocating algorithm, we can calculate the window-constraints and deadlines that must be assigned to streams in DWCS to give the equivalent bandwidth allocations. This is done as follows:

1. Determine the minimum time window, Δ_{min}, over which bandwidth is shared proportionally among n streams, each with weight $w_i | 1 \leq i \leq n, w_i \in Z^+$:

   Let $\omega = \sum_{i=1}^{n} w_i$ and let η_i be the number of packets from S_i serviced in some arbitrary time window Δ. (Note that $\eta_i C_i$ is the total service time of S_i over the interval Δ and $\sum_{i=1}^{n} \eta_i C_i = \Delta$. Furthermore, Δ is assumed sufficiently large to ensure bandwidth allocations among all n streams in *exact* proportions to their weights.) This implies that $\frac{\eta_i C_i}{\Delta} = \frac{w_i}{\omega}$.

   If w_i is a factor of ωC_i, let $\gamma_i = \frac{\omega C_i}{w_i}$, else let $\gamma_i = \omega C_i$.

   Then, $\Delta_{min} = lcm(\gamma_1, \ldots, \gamma_n)$, where $lcm(a, b)$ is the lowest-common-multiple of a and b.
2. For DWCS, set $T_i = C_i$, for each stream S_i.
3. To calculate the window-constraint, $W_i = x_i/y_i$ set: $y_i = \frac{\Delta_{min}}{C_i}$ and $x_i = \frac{\Delta_{min}}{C_i} - \eta'_i$,
   where $\eta'_i = \frac{\eta_i \Delta_{min}}{\Delta} = \frac{w_i \Delta_{min}}{\omega C_i}$.

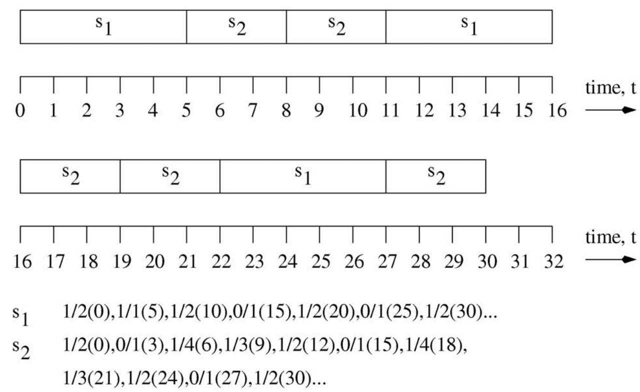


Fig. 6. Example DWCS scheduling of two streams, s_1, s_2, using the original table of precedence rules [38]. Note that the fine-grained loss-constraints of each stream are no longer met, but each stream gets 50 percent of the bandwidth every 30 time units.

If successive packet deadlines in S_i are offset by $T_i = C_i$, as in Step 2, we can translate packet window-constraints back into stream *weights*, w_i, as follows:

$$w_i = \frac{y_n(y_i - x_i)}{y_i(y_n - x_n)},$$

where $0 < \frac{x_i}{y_i} < 1$.

Summary. DWCS is a flexible scheduling algorithm. Besides having the ability to guarantee window-constraints in a slotted time system, DWCS can be configured to operate as an EDF, static priority, or fair scheduling algorithm. DWCS ensures the delay of service to real-time streams is bounded, even in the absence of a feasibility test, whereby the scheduler may be overloaded and window-constraint violations can occur. Consequently, a stream is guaranteed never to suffer starvation. Finally, the least upper bound on bandwidth utilization using DWCS can be as high as 100 percent.

5 Simulation Results

To investigate the ability of DWCS to meet per-stream window-constraints, we conducted a series of simulations. A number of streams, comprising constant (i.e., unit) length packets, with different request periods and original window-constraints were scheduled by a single server. Each stream generated one packet arrival every request period and the scheduler was invoked at fixed intervals. The performance of DWCS, using the precedence rules shown in Table 1, was compared to that of the Distance-Based Priority (DBP) algorithm [15].

Scenario 1								
			DWCS			DBP		
n	U	U_{max}	D	V_f	V_s	D	V_f	V_s
240	0.4830	0.5000	0	0	0	0	0	0
320	0.6440	0.6666	0	0	0	0	0	0
400	0.8050	0.8333	0	0	0	0	0	0
480	0.9660	1.0000	0	0	0	0	0	0
488	0.9821	1.0166	16664	0	0	16664	19	0
496	0.9982	1.0333	33328	0	0	33328	2136	0
504	1.0143	1.0499	49992	12057	58494	49992	14271	609
512	1.0304	1.0666	66656	24608	154144	66656	25696	157728
520	1.0465	1.0833	83320	34305	327165	83320	30180	678510

Scenario 2								
			DWCS			DBP		
n	U	U_{max}	D	V_f	V_s	D	V_f	V_s
80	0.2810	0.2916	0	0	0	0	0	0
160	0.5620	0.5833	0	0	0	0	0	0
240	0.8430	0.8749	0	0	0	0	0	0
256	0.8921	0.9333	0	0	0	0	0	0
272	0.9554	0.9916	0	0	0	0	0	0
280	0.9835	1.0208	20820	0	0	37360	350	0
288	1.0116	1.0499	49968	11868	17436	69492	31200	2100
304	1.0678	1.1083	108264	40204	390066	108462	36086	983752
320	1.1240	1.1666	166560	42520	661320	166640	38480	1063000

Scenario 3								
			DWCS			DBP		
n	U	U_{max}	D	V_f	V_s	D	V_f	V_s
480	0.9156	0.9518	0	0	0	0	0	0
496	0.9461	0.9835	0	0	0	0	0	0
504	0.9613	0.9994	0	0	0	0	0	0
512	0.9766	1.0152	15152	0	0	36544	0	0
520	0.9919	1.0311	30990	25	150	38780	0	0
528	1.0071	1.0470	46828	10014	56342	57190	17236	0
544	1.0376	1.0787	78528	25584	398516	79028	36128	112760
560	1.0681	1.1104	110240	33880	722230	110330	37580	966630
640	1.2207	1.2690	268800	48080	1239120	268800	44160	1183200

Fig. 7. Comparison between DWCS and DBP scheduling, in terms of window-constraint violations and missed deadlines.

The DBP algorithm maintains a *sliding* window "history" for each stream. That is, for the most recent k_i deadlines of stream S_i, a binary string of k_i bits is used to track whether or not deadlines have been met or missed. Using this information, DBP scheduling assigns priorities to streams based on their minimum "distance" from a failing state, which occurs when a stream S_i meets fewer than m_i out of the most recent k_i deadlines. For two or more streams with the highest priority, DBP scheduling selects the packet at the head of the stream with the earliest deadline.

We compared the performance of DBP scheduling and DWCS over both fixed and sliding windows (as discussed in Section 4.3), for the following three simulation scenarios:

- *Scenario 1*: There were eight scheduling classes for all streams. The original (x/y) window-constraints for each class of streams were $1/10$, $1/20$, $1/30$, $1/40$, $1/50$, $1/60$, $1/70$, and $1/80$ and the request period for each packet in every stream was 480 time units. Each packet in every stream required at most one time unit of service in its request period or else that packet missed its deadline.
- *Scenario 2*: This was the same as *Scenario 1* except that the request periods for packets in streams belonging to the first four classes (with window-constraints $1/10$ to $1/40$) were 240 time units and the request periods for packets in streams belonging to the remaining four classes were 320 time units.
- *Scenario 3*: This was the same as *Scenario 1* except that the request periods for packets in streams belonging to each pair of classes (starting from the class with a window-constraint of $1/10$) were 400, 480, 560, and 640 time units, respectively.

The numbers of streams in each simulation case were uniformly distributed between each scheduling class and a total of a million packets across all streams were serviced. It should be noted that, in each of the three scenarios, the window-constraints for different streams were chosen simply to capture a range of utilization factors roughly centered around 1.0. It could be argued that more realistic window-constraints and simulation scenarios (e.g., those suitable for MPEG multimedia streams) would be more appropriate, but the above cases suffice to assess the performance of DWCS and DBP scheduling over a range of conditions.

Fig. 7 shows the results of Scenarios 1, 2, and 3, respectively, where the number of streams, n, was varied to yield a range of utilization factors, U, as defined in Section 4.2. U_{max} is the utilization factor considering we try to service a stream in every request period without missing deadlines, D is the actual number of missed deadlines, and V_f is the number of fixed window violations. In the latter case, V_f captures the situations where more than x_i deadlines are missed in fixed windows of y_i deadlines for each stream S_i. In contrast, V_s shows the number of *sliding* window violations over $y_i^s = y_i + x_i$ consecutive deadlines in stream S_i.

Deadline Misses and Fixed Window Violations. Figs. 8 and 9 show the total deadline misses (D) and fixed window violations (V_f) of DWCS and DBP scheduling for various values of U. As can be seen from Fig. 8, DWCS never results in more deadline misses than DBP scheduling. In Scenario 1, all streams have the same request periods, so both DWCS and DBP scheduling result in the same numbers of missed deadlines. Additionally, both algorithms have no missed deadlines when $U_{max} \leq 1.0$.

Although some packets miss their deadlines when U is less than 1.0 and $U_{max} > 1.0$, DWCS results in no fixed window violations, V_f, in either of Scenarios 1 and 2 until U exceeds 1.0. We showed U could rise to 1.0 in Corollary 1 and still guarantee a feasible schedule, under the assumption that all stream request periods were equal. While all request periods are equal in Scenario 1, which helps to validate the deterministic service guarantees of DWCS,

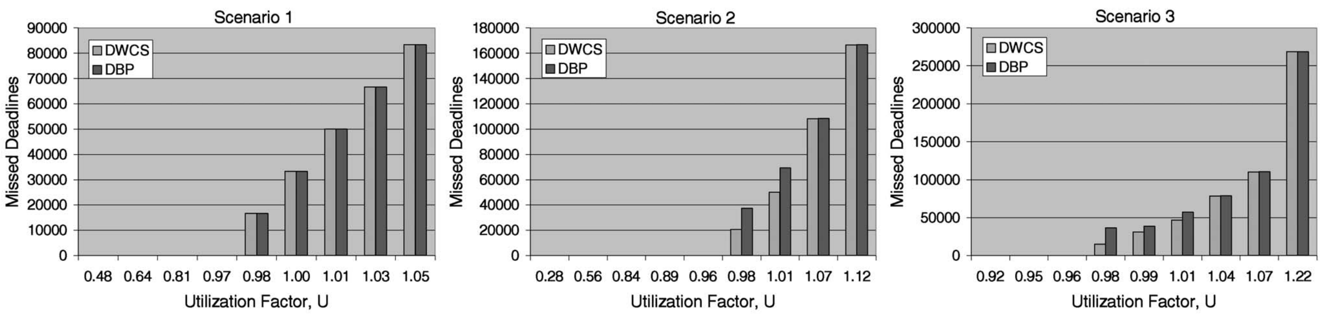


Fig. 8. Total deadlines missed versus utilization factor, U, for DWCS and DBP algorithms.

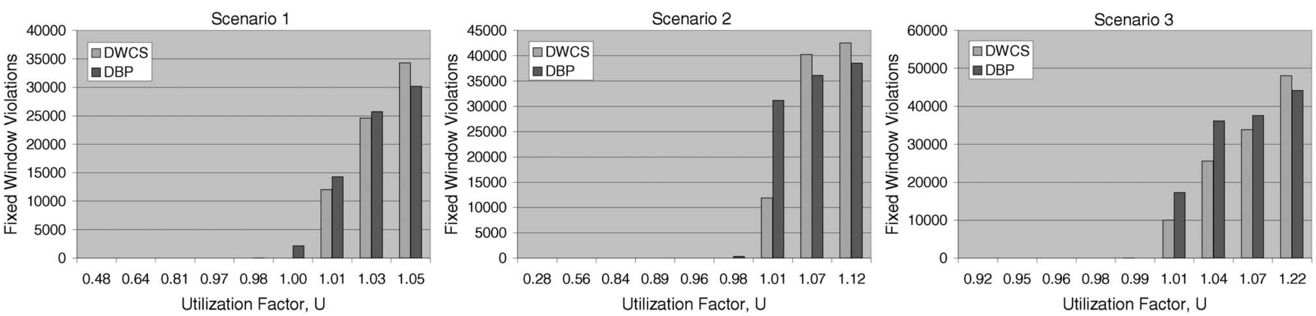


Fig. 9. Fixed window violations versus utilization factor, U, for DWCS and DBP algorithms.

Scenarios 2 and 3 involve streams with different request periods. In practice, DWCS does well in all three scenarios at minimizing fixed window violations when $U \leq 1.0$. While DBP scheduling also performs well, its values of V_f are generally worse than with DWCS, for cases when $U \leq 1.0$. For example, DBP scheduling results in 2,136 fixed window violations in Scenario 1, when $n = 496$ streams and $U = 0.9982$. In other words, DBP scheduling fails to guarantee zero fixed window violations for $U \leq 1.0$ when all stream request periods are equal. In contrast, this guarantee is assured with DWCS. Notwithstanding, DBP scheduling tends to perform slightly better than DWCS at minimizing the value of V_f as U starts to rise significantly above 1.0. In this case, it is impossible to meet all window-constraints over fixed intervals of time. However, DWCS tends to evenly spread deadline misses over fixed windows, resulting in more fixed window violations, while DBP scheduling may concentrate deadline misses in a fewer number of nonoverlapping windows. It could be argued that, in overload, DWCS is providing the more desirable characteristics by attempting to prevent long bursts of consecutive deadline misses.

Sliding Window Violations. As stated above, for each stream S_i, if no more than x_i deadlines are missed in a *fixed* window of y_i deadlines, then no more than $x_i^s = 2x_i$ deadlines are missed in a *sliding* window of $y_i^s = y_i + x_i$ deadlines. The tables in Fig. 7 include the total number of violations, V_s, for each and every stream over sliding windows of y_i^s deadlines. As can be seen, both DWCS and DBP scheduling result in $V_s = 0$ when the corresponding value of V_f is zero.

To measure V_s using DBP scheduling, we had to rerun our experiments with larger histories for each stream. Observe that, for measuring values of V_f, we ran the DBP scheduler with k_i bit histories for each S_i. However, for sliding window violations, we maintained per-stream strings of bits for the most recent $k_i^s = y_i^s = y_i + x_i = 2k_i - m_i$ deadlines. If more than $m_i^s = m_i$ deadline misses occurred in the current history, a sliding window violation was recorded.

The measured values of sliding window violations, V_s, for DWCS are based on original window-constraints specified over smaller fixed windows. In effect, DBP scheduling is using more information from the recent past to make scheduling decisions that minimize V_s. As a result, DBP scheduling has lower values for V_s when U first exceeds 1.0. However, with the exception of Scenario 3, DWCS has smaller values for V_s when U is significantly above 1.0.

Summary. Results show that DWCS and DBP scheduling have, in many cases, similar performance. DWCS explicitly uses fixed window-constraints to provide service guarantees, while DBP scheduling maintains a sliding window history. Of the two algorithms, only DWCS is capable of preventing fixed window violations when $U \leq 1.0$ and all request periods are equal.

6 Conclusions

This paper describes a version of Dynamic Window-Constrained scheduling (DWCS) [38], [39] for scheduling real-time packet streams in media servers. In this paper, we have shown how DWCS can guarantee no more than x_i deadlines are missed in a sequence of y_i packets in stream S_i. Using DWCS, the delay of service to real-time streams is bounded, even when the scheduler is overloaded. In fact, DWCS can ensure the delay bound of any given stream is independent of other streams even in overload. Additionally, the algorithm is capable of not only satisfying window-constraints, but can operate as an EDF, static priority or (weighted) fair scheduler.

If scheduling is performed at the granularity of fixed-sized time slots and only one stream is serviced in any slot, then DWCS behaves in a manner similar to pinwheel scheduling [17]. Both DWCS and pinwheel scheduling will attempt to provide service to an activity (such as a stream) in at least m out of k slots. The main difference is that, in window-constrained scheduling, activities may have their own request periods[6] and service time requirements within such periods. This is similar to rate-monotonic scheduling [24], but with the addition that service in some request periods can be missed or incomplete.

The "Distance-Based" Priority (DBP) algorithm by Hamdaoui and Ramanathan [15] is another example of a window-constrained scheduling policy. Simulation results show DWCS and DBP scheduling have similar performance characteristics. DWCS explicitly uses fixed window-constraints to provide service guarantees, while DBP scheduling maintains a sliding window history. Moreover, DWCS is capable of providing service guarantees over fixed windows, when resource usage reaches 100 percent, in situations where DBP scheduling fails.

While DWCS is intended to support real-time multimedia streams, it is often the case that such streams comprise packets, or frames, of different importances. For example, MPEG streams consist of I, P, and B frames, where the impact of losing an I frame is considerably more important than losing a B frame. In past work [38], we have demonstrated the use of DWCS on MPEG-1 streams, by packetizing each frame-type into a separate substream with its own window-constraint and request period. That is, I frames are placed in one substream, while P and B frames are placed in different streams and all frames received at a destination are buffered and reordered. Using this method, we are able to support media streams whose contents have multiple different service requirements.

Further information regarding DWCS, including kernel patches and source code, is available from our website [23]. We have implemented the algorithm as both a CPU and packet scheduler in the Linux kernel. Other work has focused on efficient hardware implementations [22], [21] of this algorithm and others.

APPENDIX A

Proof of Theorem 2. The worst-case delay experienced by S_i can be broken down into three parts: 1) the time for the next packet in S_i to have the earliest deadline among all packets queued for service, 2) the time taken for W'_i to become the minimum among all current window-constraints, $W'_k \mid 1 \leq k \leq n$, when the head packets in all n streams have the same (earliest) deadline, and 3) the time for y'_i to be larger than any other current denominator, $y'_j \mid j \neq i, 1 \leq j \leq n$, among each stream, S_j, with the minimum current window-constraint and earliest packet deadline. At this point, S_i may be delayed a further C_{max} due to another packet currently in service.

Part 1): The next packet in S_i is never more than T_i away from its deadline. Consequently, S_i will have a packet with the earliest deadline after a delay of at most T_i.

Part 2): $W'_i = 0$ is the minimum possible current window-constraint. From Theorem 1, $W'_i = 0$ after a delay of at most $x_i T_i$.

Parts 1) and 2) contribute a maximum delay of:

$$(x_i + 1)T_i. \quad (1)$$

Part 3): Assuming all streams have the minimum current window-constraint and are comprised of a head packet with the earliest deadline, the next stream chosen for service is the one with the highest current window-denominator. Moreover, the worst-case scenario is when all other streams have the same or higher current window-denominators than S_i and every time another stream, S_j, is serviced, deadline $d_{j,head} \leq d_{i,head}$. To show that $d_{j,head} \leq d_{i,head}$ holds, all deadlines must be at the same time, t, when some stream S_j is serviced in preference to S_i. After servicing a packet in S_j for C_j time units, all packet deadlines $d_{k,head}$ that are earlier than $t + C_j$ are incremented by a multiple of the corresponding request periods, $T_k \mid 1 \leq k \leq n$, depending on how many request periods have elapsed while servicing S_j. The worst case is that $T_j \leq T_i, \forall j \neq i$. Furthermore, every time a stream, S_j, other than S_i is serviced, $W'_j = 0$. This is true regardless of whether or not S_j is tagged with a violation, if $W_j = 0$, which is the case when $x_j = 0$.

Hence, the worst-case delay incurred by S_i when $W'_i = 0$ is $T_i + \delta_i$, where δ_i is the maximum time for y'_i to become larger than any other current denominator, $y'_j \mid j \neq i, 1 \leq j \leq n$, among all streams with the minimum current window-constraint and earliest packet deadline. Now, let state ϕ be when each stream, S_k, has $W'_k = 0$ for the first time. Moreover, $W'_k = 0/y'_{k_\phi}$ and $y'_{k_\phi} > 0$ is the current window-denominator for S_k when in state ϕ.

Suppose $T_j \leq T_i, \forall j \neq i$, and T_j is finite. For n streams, the worst-case δ_i is when $T_j = K$ and $T_i >> K$, for some constant, K, equal to the largest packet service time, C_{max}. Without loss of generality, it can be assumed in what follows that all packet service times equal C_{max}. Now, it should be clear that, if T_i tends to infinity, then the rate of increase of y'_i approaches 0. Moreover, if each and every stream, $S_j | j \neq i$, has a request period, $T_j = K$, then S_i will experience its worst delay before $y'_i \geq y'_j$. This is because y'_j rises at a rate of $1/K$ for each stream S_j experiencing a delay of K time units without service, while y'_i increases at a rate of $1/T_i$, which is less than or equal to $1/K$.

Fig. 10 shows the worst-case situation for three streams, S_i, S_l, and S_m, which causes S_i the largest delay, δ_i, before y'_i is the largest current window-denominator. From the figure, $y'_{l_\phi} = y'_{m_\phi}$, and y'_i increases at a rate $dy'_i/dt = \epsilon/T_i \mid \epsilon = 1$, until S_i is serviced. When S_m is serviced, y'_m decreases at a rate of $1/K$, while y'_l increases at a rate of $1/K$. Conversely, when S_l is serviced, y'_l decreases at a rate of $1/K$, while y'_m increases at a rate of $1/K$. Only when $y'_m = 0$ is W'_m reset. Likewise,

6. Pinwheel scheduling essentially assumes activities' request periods are all equal to the size of a time slot.

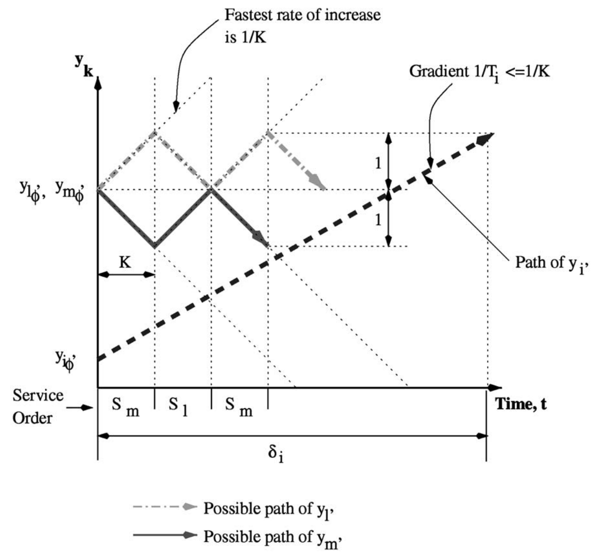


Fig. 10. The change in current window-denominators, y_i', y_l' and y_m' for three streams, S_i, S_l, and S_m, respectively, when all request periods, except possibly T_i, are finite. The initial state, ϕ, is when all current window-constraints first equal 0 and the current window-denominators are all greater than 0.

only when $y_l' = 0$ is W_l' reset. Consequently, $y_i' \geq max[y_l', y_m']$ is true when $y_i' = y_{l_\phi}' + 1 = y_{m_\phi}' + 1$.

Suppose now, another stream, S_o (with $y_{o_\phi}' = y_{l_\phi}' = y_{m_\phi}'$ and $T_o = K$), is serviced before either S_l or S_m when in state ϕ. Then, $y_l' = y_m' = y_{l_\phi}' + 1 = y_{m_\phi}' + 1$ after K time units. If S_l is now serviced, then $y_m' = y_{m_\phi}' + 2$ after a further K time units. In this case, $y_i' \geq max[y_l', y_m', y_o']$ is true when $y_i' = y_{l_\phi}' + 2 = y_{m_\phi}' + 2 = y_{o_\phi}' + 2$. In general, for each of the $n-1$ streams, S_j, other than S_i, each with $T_j = K$ and $y_{j_\phi}' \geq y_{i_\phi}'$, it is the case that $y_i' \geq max[y_1', \cdots, y_{i-1}', y_{i+1}', \cdots, y_n']$ is true when

$$y_i' = y_{1_\phi}' + (n-2) = \cdots = y_{n_\phi}' + (n-2).$$

Therefore, since $dy_i'/dt = 1/T_i$, it follows that

$$\delta_i \leq T_i(y_{j_\phi}' - y_{i_\phi}' + (n-2)).$$

Now, observe that $y_{j_\phi}' \leq y_j$ for each and every stream, $S_j \mid j \neq i$, since state ϕ is the first time W_j' is 0. Furthermore, we have the constraints that

$$y_j = max[y_1, \cdots, y_{i-1}, y_{i+1}, \cdots, y_n],$$

$y_i \leq y_j$, and $y_{i_\phi}' \geq 1$. Therefore,

$$\delta_i \leq T_i(y_j + (n-2)). \quad (2)$$

If $T_j > T_i, \forall j \neq i$ and both T_j and T_i are finite, then y_i' and y_j' converge more quickly than in the case above, when $T_j \leq T_i$. Therefore, if window-constraint violations occur, the maximum delay of service to S_i (from (1) and (2)) is no more than

$$(x_i + 1)T_i + T_i(y_{max} + n - 2) + C_{max},$$

or, equivalently,

$$T_i(x_i + y_{max} + n - 1) + C_{max},$$

where $y_j = y_{max}$ in (2) and C_{max} is the worst-case additional delay due to another packet in service when a packet in S_i reaches the highest priority. □

APPENDIX B

Proof of Theorem 3. When $n \leq q$, it is clear there are never more than q streams with current window-constraint $\frac{0}{y_i'}$. For all nontrivial values of n, it must be that $q < n \leq qy_k$, given that $y_k = max(y_1, y_2, \cdots, y_n)$. From Lemma 1, if $y_1 = y_2 = \cdots = y_n$, and $x_i = y_i - 1, \forall i$, then $n \leq qy_i$. It can be shown that all lower values of n will yield a feasible schedule if one exists for largest n.

Now, consider the set Γ comprised of one stream, S_j, that has window-constraint, x_j/y_j, and $n-1$ other streams, each having window constraint, x_i/y_i. From Lemma 1, it follows that if $x_j/y_j < x_i/y_i$, then $n < qy_i$. In this case, n is maximized if $x_j = y_j - 1$, $x_j + 1 = x_i$, and $x_i = y_i - 1$. Hence, $x_j < x_i$, $y_j < y_i$ and $n < q(x_i + 1)$.

The set Γ is scheduled in the various nonoverlapping intervals of the hyperperiod, resulting in changes to window-constraints, as shown below.

1. *Time interval* $[0, q)$: Stream S_j is scheduled first since $x_j/y_j < x_i/y_i$. The current window-constraints of each stream are adjusted over the time interval (shown above the arrows) as follows:

$$\frac{x_j}{y_j} \xrightarrow{q} \frac{x_j}{y_j - 1} \quad \text{(one stream,} S_j\text{, serviced on time)}$$
$$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i}{y_i - 1} \quad (q-1 \text{ streams serviced on time})$$
$$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i - 1}{y_i - 1} \quad (n-q \text{ streams not serviced on time}).$$

2. *Time interval* $[q, q(x_j + 1))$: It can be shown that $n > q(x_j + 1)$ when n is maximized. Furthermore, in this scenario, DWCS will schedule qx_j streams with the smallest current window-constraints, updated every q time units. As a result, window-constraints now change as follows:

$$\frac{x_j}{y_j - 1} \xrightarrow{qx_j} \frac{0}{y_j - 1 - x_j} \quad \text{(one stream,} S_j\text{, not serviced)}$$
$$\frac{x_i}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - x_j}{y_i - 1 - x_j} \quad (q-1 \text{ streams not serviced on time})$$
$$\frac{x_i - 1}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - 1 - x_j}{y_i - 1 - x_j} \quad (n - q - qx_j \text{ streams not serviced})$$
$$\frac{x_i - 1}{y_i - 1} \xrightarrow{qx_j} \frac{x_i - x_j}{y_i - 1 - x_j} \quad (qx_j \text{ streams serviced on time}).$$

At this point, consider the $n - q - qx_j$ streams in state $\frac{x_i - 1 - x_j}{y_i - 1 - y_j}$ after time $q(x_j + 1)$. We know, in the worst case, $x_j + 1 = x_i$ to maximize n, so we have

$$n - q - qx_j = n - q(x_j + 1) = n - qx_i.$$

We also know $n < q(x_i + 1)$, therefore

$$n - qx_i < q(x_i + 1) - qx_i = q.$$

Consequently, at the time $q(x_j + 1)$, less than q streams other than S_j are in state $\frac{0}{y_i'}$. Even though S_j is in state $\frac{0}{y_j'}$, we can never have more than

q streams with zero-valued numerators as part of their current window-constraints. We know that, by maximizing n, we have

$$x_j + 1 = x_i, x_j + 1 = y_j \Rightarrow y_j = x_i.$$

Therefore, at time $q(x_j + 1)$, all current window-constraints can be derived from their original window-constraints, as follows:

$$\frac{x_j}{y_j} \stackrel{q(x_j+1)}{\longrightarrow} \frac{0}{0} \quad (1 \text{ stream}, S_j, \text{served once; reset } \tfrac{0}{0} \text{ to } \tfrac{x_j}{y_j})$$
$$\frac{x_i}{y_i} \stackrel{q(x_j+1)}{\longrightarrow} \frac{0}{1} \quad (n - qx_i \text{ streams never serviced on time})$$
$$\frac{x_i}{y_i} \stackrel{q(x_j+1)}{\longrightarrow} \frac{1}{1} \quad (q - 1 \text{ streams serviced once on time})$$
$$\frac{x_i}{y_i} \stackrel{q(x_j+1)}{\longrightarrow} \frac{1}{1} \quad (qx_j \text{ streams serviced once on time}).$$

3. *Time interval* $[q(x_j + 1), q(x_j + 2))$: At the end of this interval of size q, the window-constraints change from their original values, as follows:

$$\frac{x_j}{y_j} \stackrel{q(x_j+2)}{\longrightarrow} \frac{x_j}{y_j-1} \quad (1 \text{ stream, } S_j, \text{serviced twice overall})$$
$$\frac{x_i}{y_i} \stackrel{q(x_j+2)}{\longrightarrow} \frac{x_i}{y_i} \quad (n - 1 \text{ streams serviced at least once; reset window-constraints}).$$

4. *Time interval* $[q(x_j + 2), 2q(x_j + 2))$: At the end of this interval of size $q(x_j + 2)$, the window-constraints change from their original values, as follows:

$$\frac{x_j}{y_j} \stackrel{2q(x_j+2)}{\longrightarrow} \frac{x_j}{y_j-2} \quad (1 \text{ stream}, S_j)$$
$$\frac{x_i}{y_i} \stackrel{2q(x_j+2)}{\longrightarrow} \frac{x_i}{y_i} \quad (n - 1 \text{ streams; reset window-constraint}).$$

Over the entire period $[0, y_j q(x_j + 2)]$, the window-constraints change as follows:

$$\frac{x_j}{y_j} \stackrel{y_j q(x_j+2)}{\longrightarrow} \frac{x_j}{y_j} \quad (1 \text{ stream}, S_j)$$
$$\frac{x_i}{y_i} \stackrel{y_j q(x_j+2)}{\longrightarrow} \frac{x_i}{y_i} \quad (n - 1 \text{ streams}).$$

At this point, every stream has been served at least once and no more than q streams ever have zero-valued current window-constraints in any given nonoverlapping interval of size q. Observe that the hyperperiod is $lcm(qy_1, qy_2, \cdots, qy_n)$ which, in this case, is $qy_i y_j$. Since $x_j + 2 = y_i$, $y_j q(x_j + 2) = qy_i y_j$, and we have completed the hyperperiod. All streams have reset their window-constraints to their original values, so we have a feasible schedule. □

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments that have helped improve the quality of this paper.

REFERENCES

[1] A. Atlas and A. Bestavros, "Statistical Rate Monotonic Scheduling," *Proc. 19th IEEE Real-Time Systems Symp.*, pp. 123-132, 1998.

[2] S.K. Baruah and S.-S. Lin, "Pfair Scheduling of Generalized Pinwheel Task Systems," *IEEE Trans. Computers*, vol. 47, no. 7, pp. 812-516, July 1998.

[3] J.C. Bennett and H. Zhang, "WF^2Q: Worst-Case Fair Weighted Fair Queueing," *Proc. IEEE INFOCOM '96*, pp. 120-128, Mar. 1996.

[4] G. Bernat and A. Burns, "Combining (n/m)-Hard Deadlines and Dual Priority Scheduling," *Proc. 18th IEEE Real-Time Systems Symp.*, pp. 46-57, Dec. 1997.

[5] G. Bernat, A. Burns, and A. Llamosi, "Weakly-Hard Real-Time Systems," *IEEE Trans. Computers*, vol. 50, no. 4, pp. 308-321, Apr. 2001.

[6] G. Bernat and R. Cayssials, "Guaranteed On-Line Weakly-Hard Real-Time Systems," *Proc. 22nd IEEE Real-Time Systems Symp.*, Dec. 2001.

[7] P.P. Bhattacharya and A. Ephremides, "Optimal Scheduling with Strict Deadlines," *IEEE Trans. Automatic Control*, vol. 34, no. 7, pp. 721-728, July 1989.

[8] C. Carlsson and O. Hagsand, "DIVE—A Platform for Multi-User Virtual Environments," *Computers and Graphics*, vol. 17, no. 6, pp. 663-669, Nov./Dec. 1993.

[9] M. Chan and F. Chin, "Schedulers for the Pinwheel Problem Based on Double-Integer Reduction," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 755-768, June 1992.

[10] A. Demers, S. Keshav, and S. Schenker, "Analysis and Simulation of a Fair-Queueing Algorithm," *J. Internetworking Research and Experience*, pp. 3-26, Oct. 1990.

[11] D. Ferrari, "Client Requirements for Real-Time Communication Services," *IEEE Comm. Magazine*, vol. 28, no. 11, pp. 76-90, Nov. 1990.

[12] S. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. INFOCOM '94*, pp. 636-646, Apr. 1994.

[13] P. Goyal, H.M. Vin, and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *Proc. IEEE SIGCOMM '96*, 1996.

[14] S. Greenberg and D. Marwood, "Real-Time Groupware as a Distributed System: Concurrency Control and Its Effect on the Interface," *Proc. ACM Conf. Cooperative Support for Cooperative Work*, pp. 207-217, 1994.

[15] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines," *IEEE Trans. Computers*, vol. 44, no. 11, pp. 1443-1454, Nov. 1995.

[16] J.M. Harrison, "Dynamic Scheduling of a Multiclass Queue: Discount Optimality," *Operations Research*, vol. 23, no. 2, pp. 370-382, Mar.-Apr. 1975.

[17] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem," *Proc. 22nd Hawaii Int'l Conf. System Science*, pp. 693-702, Jan. 1989.

[18] K. Jeffay and S. Goddard, "A Theory of Rate-Based Execution," *Proc. 20th IEEE Real-Time Systems Symp. (RTSS)*, Dec. 1999.

[19] G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," *Proc. 16th IEEE Real-Time Systems Symp.*, pp. 110-117, Dec. 1995.

[20] R. Kravets, K. Calvert, P. Krishnan, and K. Schwan, "Adaptive Variation of Reliability," *Proc. High-Performance Networking Conf. (HPN-97)*, Apr. 1997.

[21] R. Krishnamurthy, K. Schwan, R. West, and M. Rosu, "On Network Coprocessors for Scalable, Predictable Media Services," *IEEE Trans. Parallel and Distributed Systems*, to appear.

[22] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West, "Leveraging Block Decisions and Aggregation in the Sharestreams QoS Architecture," *Proc. Int'l Conf. Parallel and Distributed Systems (IPDPS)*, 2003.

[23] Linux DWCS, http://www.cs.bu.edu/fac/richwest/dwcs.html, 2004.

[24] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, Jan. 1973.

[25] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks," *Proc. Operating Systems Design and Implementation*, Dec. 2002.

[26] A.K. Mok and W. Wang, "Window-Constrained Real-Time Periodic Task Scheduling," *Proc. 22nd IEEE Real-Time Systems Symp.*, 2001.

[27] J. Nieh and M.S. Lam, "The Design, Implementatin and Evaluation of SMART: A Scheduler for Multimedia Applications," *Proc. 16th ACM Symp. Operating Systems Principles,* Oct. 1997.

[28] S.S. Panwar, D. Towsley, and J.K. Wolf, "Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service," *J. ACM,* vol. 35, no. 4, pp. 832-844, Oct. 1988.

[29] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE/ACM Trans. Networking,* vol. 1, no. 3, pp. 344-357, June 1993.

[30] X.G. Pawan Goyal and H.M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," *Proc. Second Symp. Operating Systems Design and Implementation,* pp. 107-121, 1996.

[31] J.M. Peha and F.A. Tobagi, "A Cost-Based Scheduling Algorithm to Support Integrated Services," *Proc. IEEE INFOCOM '91,* pp. 741-753, 1991.

[32] S. Singhal, "Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments," PhD thesis, Stanford Univ., Aug. 1996.

[33] D. Stiliadis and A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks," *IEEE/ACM Trans. Networking,* vol. 6, no. 2, 1998.

[34] I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton, "A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems," *Proc. Real-Time Systems Symp.,* Dec. 1996.

[35] A. Striegel and G. Manimaran, "Dynamic Class-Based Queue Management for Scalable Media Servers," *J. Systems and Software,* 2003.

[36] R. West, I. Ganev, and K. Schwan, "Window-Constrained Process Scheduling for Linux Systems," *Proc. Third Real-Time Linux Workshop,* Nov. 2001.

[37] R. West and C. Poellabauer, "Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams," *Proc. 21st IEEE Real-Time Systems Symp.,* Dec. 2000.

[38] R. West and K. Schwan, " Dynamic Window-Constrained Scheduling for Multimedia Applications," *Proc. Sixth Int'l Conf. Multimedia Computing and Systems (ICMCS '99),* June 1999. Also available as Technical Report: GIT-CC-98-18, Georgia Inst. of Technology.

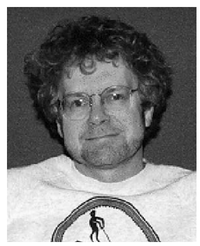
[39] R. West, K. Schwan, and C. Poellabauer, "Scalable Scheduling Support for Loss and Delay Constrained Media Streams," *Proc Fifth IEEE Real-Time Technology and Applications Symp.,* June 1999.

[40] J. Xu and R.J. Lipton, "On Fundamental Tradeoffs between Delay Bounds and Computational Complexity in Packet Scheduling Algorithms," *Proc. SIGCOMM 2002,* Aug. 2002.

[41] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *Proc. ACM SIGCOMM,* pp. 113-121, Aug. 1991.

Richard West received the MEng degree in microelectronics and software engineering in 1991 from the University of Newcastle-upon-Tyne, England. He later received both the MS (1998) and PhD (2000) degrees in computer science from the Georgia Institute of Technology. He is currently an assistant professor in the Computer Science Department at Boston University, where his research interests include operating systems, real-time systems, distributed computing, and QoS management. He is a member of the IEEE.

Yuting Zhang received the Bachelor's and Master's degrees in 1997 and 2000, respectively, from the University of Science and Technology, Beijing, China. She is currently a PhD student in computer science at Boston University. Her research interests include operating systems, computer networks, resource management, and real-time computing.



Karsten Schwan received the MSc and PhD degrees from Carnegie-Mellon University, Pittsburgh, Pennsylvania, where he initiated his research in high-performance computing, addressing operating and programming systems support for the Cm* multiprocessor. He is a professor in the College of Computing at the Georgia Institute of Technology and the director of the Center for Experimental Research in Computer Systems (CERCS), which spans both the College of Computing and the School of Electrical and Computer Engineering. At Georgia Tech, his work concerns middleware and systems support for computational grids and for pervasive systems, focusing on the interactive nature of modern distributed and parallel applications (online monitoring and program steering), the adaptive nature of pervasive applications (system support for predictable operation in dynamic environments), and the online configuration and adaptation of distributed application or system components (network-aware communication). He is a senior member of the IEEE.

Christian Poellabauer received the Master's degree in computer science from the Vienna University of Technology in 1998. Currently, he is a PhD candidate in computer science at the Georgia Institute of Technology. He is a recipient of an IBM PhD Research Fellowship. His research interests are in the area of experimental computing systems, including real-time systems, operating systems, and embedded and pervasive computing. He is a student member of the IEEE.