Dynamic Window-Constrained Scheduling for Real-Time Media Streaming

Richard West

Computer Science Department
Boston University
Boston, MA 02215
richwest@cs.bu.edu

Karsten Schwan & Christian Poellabauer

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{schwan,chris}@cc.gatech.edu

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 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 link bandwidth by a set of real-time (i.e., window-constrained) streams can exceed 100% and still ensure all window-constraints are met. In fact, we show how it is possible to guarantee worst-case per-stream bandwidth and delay constraints while utilizing all available link capacity. Finally, we show how best-effort packets can be serviced with fast response time, in the presence of window-constrained traffic.

1. Introduction

Low latency, high bandwidth integrated services networks have introduced opportunities for new applications such as video conferencing, tele-medicine, virtual environments [7, 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 amongst 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 perfor-

mance 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. In contrast to loss-constrained applications, computer data transferred between hosts using a file-transfer protocol cannot tolerate any loss at all. In this case, a more appropriate performance measure is mean delay, to ensure that the delay incurred by packets from this class of applications is minimized. It should be clear then, that integrated services networks must be able to support diverse performance objectives. Therefore, a suitable service discipline at the network access points and, possibly, switches must be able to schedule the transmission of packets from various streams so that the objectives of as many of the most important packets as possible are met.

This paper describes the real-time properties of Dynamic Window-Constrained Scheduling (DWCS) [38, 39], an algorithm that is suitable for packet scheduling in real-time communications. DWCS is designed to explicitly service packet streams in accordance with their loss and delay 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 [4, 5], and those which attempt to guarantee at least m out of k packet deadlines for each and every stream. 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 [3] schedule activities with (m, k)-*hard* deadlines, but their approach requires such hard temporal constraints to be guaranteed by off-line feasibility tests. Moreover, Bernat and Burns work focuses less on the issue of providing a solution to on-line 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, 8, 1] is also similar to DWCS. In essence, the generalized pinwheel scheduling problem is equivalent to determining a schedule for a set of n activities $\{a_i \mid 1 \leq i \leq n\}$, each requiring at least m_i deadlines are met in *any* window of k_i deadlines, given that the time between consecutive deadlines is a multiple of some fixed-size time slot, and resources are allocated at the granularity of one time slot. DWCS can be thought of as a special case of pinwheel scheduling, whereby DWCS guarantees a minimum of m_i deadlines are met every *fixed* (i.e., non-overlapping) window of k_i deadlines, for a given activity a_i. In fact, DWCS is capable of producing a feasible schedule, *independent of an activity's window size* k_i, when 100% of available resources (such as bandwidth) are utilized. By comparison, Baruah and Lin [1] have developed a pinwheel scheduling algorithm, that is capable of producing a feasible schedule when the utilization of resources approaches 100%, given that $k \rightarrow \infty$.

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.

In contrast to the related work described above, the significant contributions of this work are: (1) the description and analysis of an *on-line* version of DWCS that can guarantee m out of k deadlines (or, equivalently, no more than x missed packet deadlines for every fixed window of y consecutive packets in a given stream), (2) an approach to ensure fast response time to best-effort packet streams in the presence of real-time streams, and (3) a proof that DWCS ensures the delay of service to packets in any given stream is bounded, even in overload situations. In fact, DWCS is capable of ensuring the delay bound of any given stream is independent of other streams. As a consequence of these scheduling characteristics, DWCS can guarantee a minimum fraction of link bandwidth to each stream, over finite windows of time, independent of other streams. This is true even in overload, as DWCS strategically misses the deadlines of packets in low-priority streams, so that the more important streams (in jeopardy of violating their window-constraints) are serviced first. A side-effect of this property is that 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 queueing algorithms [10, 41, 12, 2, 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. As will be shown in Section 3.4, DWCS is a flexible algorithm, supporting many modes of operation that in-

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

clude not only window-constrained service but also earliest deadline first, static priority and fair scheduling. We believe DWCS is a suitably efficient algorithm for use in both end hosts, network access points or even programmable switches. In fact, we have shown in related 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 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% 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 precisely defines the scheduling problem, and then describes the operation of Dynamic Window-Constrained Scheduling. Section 3 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 4 describes an approach to guarantee fast response time to best-effort packet streams in the presence of streams with real-time (i.e., window-) constraints. Finally, conclusions are described in Section 5.

2. 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 Figure 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 given stream, at least one packet in a 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.

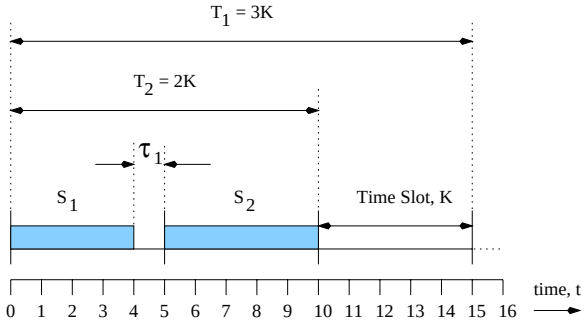

Figure 1. Example of two packets from different streams, S_1 and S_2 being serviced in their respective time slots. Each time slot is of constant size K. Observe that the packet in S_1 requires $K - \tau_1$ service time, thereby wasting τ_1 time units before the packet in S_2 is serviced. In this example, S_1 has a request period of 3 time slots, while S_2 has a request period of 2 time slots.

2.1. 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 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, comprising 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 on-line algorithm. The algorithm should attempt to maximize network bandwidth. In fact, we show in Section 3 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% 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.

2.2. 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 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 under-load 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.

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

Table 1. Precedence amongst pairs of packets in different streams. The precedence rules are applied top-to-bottom in the table above.

The desirable property of EDF, that all deadlines can be met as long as the load does not exceed 100% [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 fine-tune the characteristics of the algorithm.

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. In fact, using a slotted time system, as described earlier, deadlines can be aligned on time slot boundaries. For analysis purposes rather than intrinsic requirements of DWCS, we assume that scheduling decisions are also made on time-slot boundaries.

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 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 Figure 2.

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;

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

At this point in time, the window-constraint, W_j, of any other stream, $S_j \mid j \neq i$, comprising one or more late packets, is adjusted as shown in Figure 3. Note that with DWCS, window-constraints do not change for streams whose packets do not have deadlines. Streams comprising packets without deadlines are *non-time-constrained*, and their window-constraints act as static priorities.

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
    $y'_j = y'_j + \epsilon$;
    Tag S_j with a violation;

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

We can now show the pseudo-code for DWCS in Figure 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 can be serviced in a given request period, and exactly one 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 ensures DWCS is *work-conserving*, in that the jth packet, $p_{i,j}$ in a stream, S_i, can be serviced before the deadline of a prior packet, $p_{i,j-1}$ in the same stream, if $p_{i,j-1}$ has been serviced before the

```
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;
        }
    }
}
```

Figure 4. The DWCS algorithm.

end of its request period. This implies $p_{i,j-1}$ is serviced before its deadline. For the purposes of *real-time*, as opposed to best-effort streams, this paper assumes DWCS works as a non-work-conserving scheduler. However, all best-effort streams can be serviced whenever there is available time to service such streams.

In the absence of a feasibility test, it is possible that window-constraint violations can occur. A violation actually occurs when $W'_i = x'_i/y'_i \mid x'_i = 0$ and another packet in S_i then misses its deadline. Before S_i is serviced, x'_i remains zero, while y'_i is increased by a constant, ϵ, every time a packet in S_i misses a deadline. The exception to this rule is when $y_i = 0$ (and, more specifically, $W_i = 0/0$). This special case allows DWCS to *always* service streams in EDF order, if such a service policy is desired.

If S_i 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. Theorem 2 shows the delay bound for a stream which is tagged with window-constraint violations. Consequently, S_i is assured of service, since it will eventually take precedence over all streams with a zero-valued current window-constraint. 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_j is serviced before S_i, because $y'_j > y'_i$. At some later point in time, S_i will have the highest window-denominator, since its denominator is increased by ϵ every request period, T_i, that a packet in S_i is delayed, while S_j'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 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.

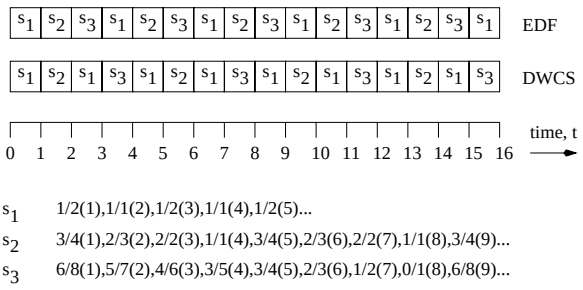

Figure 5. Example showing the scheduling of 3 streams, S_1, S_2, and S_3, using EDF and DWCS. All packets in each stream have unit service times and request periods. The window-constraints for each stream are shown as fractions, x/y, while packet deadlines are shown in brackets.

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

2.3. 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 queueing, pinwheel and weakly-hard algorithms. Observe that with fair queueing 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 Queueing (SFQ) [13, 30], Self-Clocked Fair Queueing (SCFQ) [12], or Frame-Based Fair Queueing (FFQ) and Starting-Potential Fair Queueing (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 queueing 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, 8, 1] and weakly-hard [4, 5, 15, 19] real-time schedulers than fair queueing 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 im-

proving 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 simulated experiments in Section 3.5.

3. Analysis of DWCS

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

- If a feasible schedule is known to exist, DWCS ensures that the maximum delay of service to a real-time packet stream is bounded. The exact value of this maximum delay is characterized below.
- If window-constraint violations occur (because the scheduler is overloaded), the maximum queueing delay of a stream (and, hence, packet) is still bounded. Again, 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 on-line 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 Figure 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.

3.1. Delay Characteristics

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

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 $x_i T_i$ 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 amongst all queued packets.*

Proof. 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 amongst all packets queued for service, (2) the time taken for W'_i to become the minimum amongst 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, $y'_j \mid j \neq i, 1 \leq j \leq n$, amongst 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 $x_i T_i$.

[2]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.

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 comprise 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, $y'_j \mid j \neq i, 1 \leq j \leq n$, amongst 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, $S_j \mid j \neq i$, 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 $1/K$.

Figure 6 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, only when $y'_l = 0$ is W'_l reset. Consequently, $y'_i \geq max[y'_l, y'_m]$ is true when $y'_i = y'_{l_\phi} + 1 = y'_{m_\phi} + 1$.

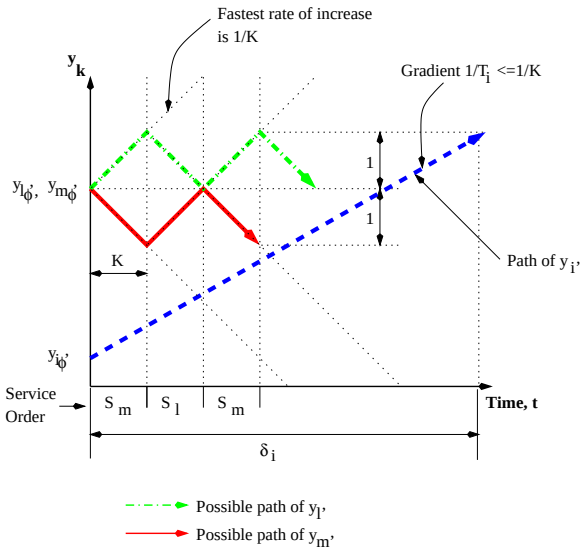

Figure 6. The change in current window-denominators, y'_i, y'_l and y'_m for three streams, S_i, S_l and S_m, respectively, when all request periods, except possibly T_i, are finite. The initial state, ϕ, is when all current window-constraints first equal 0, and the current window-denominators are all greater than 0.

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 Equations 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 Equation 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. □

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 $dy_i'/dt = \epsilon/T_i \mid \epsilon = 1$ (see Figure 6). 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.

3.2. 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 'on time' 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 $y_i T_i$ 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 $y_i T_i$ time units. For a stream, S_i, with request period, T_i, the *minimum* utilization factor is $U_i = \frac{(y_i - x_i)C_i}{y_i T_i}$, 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 $q \in Z^+$ [3], K is a constant, and U is the minimum utilization factor for a feasible schedule.[4]

This implies a feasible schedule is possible even when the server capacity, or link bandwidth, is 100% utilized, given: (a) all packets have constant, or some known worst-case, service time and, (b) 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 amongst 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 queueing algorithms that (a) attempt to share resources over the smallest window of time possible (thereby approximating the fluid-flow model) and, (b) 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 queueing 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 3. *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 non-trivial 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:

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

$$\Rightarrow n \leq qy_k$$

[3] Z^+ is the set of positive integers.

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

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 3, 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 *hyper-period* 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 hyper-period essentially defines a period in which a repeating schedule of service to all streams occurs. Let the hyper-period, H, be $lcm(qy_1, qy_2, \cdots, qy_n)$. The following theorem can now be stated:

Theorem 4. *In each* non-overlapping *window of size q in the hyper-period, 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. When $n \leq q$, it is clear there are never more than q streams with current window-constraint $\frac{0}{y_i'}$. For all non-trivial 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 3, 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 comprising 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 3, 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 non-overlapping intervals of the hyper-period, 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:

$\frac{x_j}{y_j} \xrightarrow{q} \frac{x_j}{y_j - 1}$ (one stream, S_j, serviced on time)
$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i}{y_i - 1}$ ($q - 1$ streams serviced on time)
$\frac{x_i}{y_i} \xrightarrow{q} \frac{x_i - 1}{y_i - 1}$ ($n - q$ 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:

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

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

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

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

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

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

$\frac{x_j}{y_j} \xrightarrow{q(x_j+2)} \frac{x_j}{y_j - 1}$ (1 stream, S_j, serviced twice overall)
$\frac{x_i}{y_i} \xrightarrow{q(x_j+2)} \frac{x_i}{y_i}$ ($n - 1$ 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:

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

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

$\frac{x_j}{y_j} \xrightarrow{y_j q(x_j+2)} \frac{x_j}{y_j}$ (1 stream, S_j)
$\frac{x_i}{y_i} \xrightarrow{y_j q(x_j+2)} \frac{x_i}{y_i}$ ($n-1$ 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 non-overlapping interval of size q. Observe that the hyper-period is $lcm(qy_1, qy_2, \cdots, qy_n)$ which, in this case is qy_iy_j. Since $x_j + 2 = y_i$, $y_j q(x_j + 2) = qy_iy_j$, and we have completed the hyper-period. All streams have reset their window-constraints to their original values, so we have a feasible schedule. □

Having proved Theorem 4, we can 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 5. *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 4 (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 4 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 non-zero 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.

3.3. 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 Figure 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: $(C_1 = 3, T_1 = 5, W_1 = 2/3)$, $(C_2 = 4, T_2 = 6, W_2 = 23/35)$ 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 non-preemptive 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: $(C_1^* = 1, T_1^* = 1, , W_1^* = 4/5)$, $(C_2^* = 1, T_2^* = 1, W_2^* = 27/35)$ 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.

3.4. 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 [6]. 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 Queueing 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, Figure 7 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% of the available bandwidth. The service times for each and every packet in streams S_1 and S_2 are 5 time units and 3 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 $y_i.T_i$.

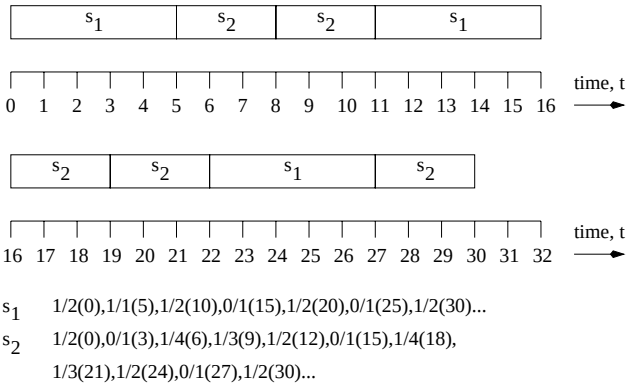

Figure 7. Example DWCS scheduling of 2 streams, s_1, s_2, using the original table of precedence rules [38]. Packets in s_1 take 5 time units to be serviced, while those in s_2 take 3 time units. Note that the fine-grained loss-constraints of each stream are no longer met but each stream gets 50% of the bandwidth every 30 time units.

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 amongst 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 $\Delta_{min} = lcm(\gamma_1, ..., \gamma_n)$, 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 $\eta'_i = \frac{\eta_i \Delta_{min}}{\Delta} = \frac{w_i \Delta_{min}}{\omega C_i}$.

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, having the ability to guarantee window-constraints in a slotted time system. 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, DWCS guarantees that a stream will never suffer starvation. Furthermore, the least upper bound on bandwidth utilization using DWCS can be as high as 100%. Finally, DWCS be configured to operate as an EDF, static priority or fair scheduling algorithm.

3.5. Simulated Results

To show that it is possible to feasibly schedule a set of packet streams when the demand for bandwidth is no more than 100% of available bandwidth, we simulated the number of missed deadlines and window-constraint violations for a number of streams, comprising fixed (unit) length packets, with different request periods and original window-constraints. All simulations assume the scheduler is invoked at fixed intervals and the table of precedence rules shown in Table 1 is used. The following three scenarios were considered:

- *Scenario 1*: There were 8 scheduling classes for all streams. The original 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 4 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 4 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.

Tables 2(a), 2(b) and 2(c) show the results of scenarios 1, 2, and 3, respectively. n is the total number of streams, D is the number of missed deadlines, V is the number of window-constraint violations, and U is the *minimum* total utilization factor. U is the value defined in Section 3.2 and signifies the minimum demand from a stream when it experiences its worst-case delay. This occurs when a maximum of x_i deadlines are missed every y_i consecutive deadlines in S_i.

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 clear from the tables that, although some packets miss their deadlines when U is less than 1.0, there are no window-constraint violations for any streams until U exceeds 1.0. It is also interesting to note that Senarios 2 and 3 show scheduling results for classes having different request periods. While we showed U could rise to 1.0 in Corollary 5 and still guarantee a feasible schedule, the assumption was that all stream request periods were equal. Scenarios 2 and 3 relax this constraint and a feasible schedule is still possible, as seen by the absence of window-constraint violations, until U exceeds 1.0. This shows that in practice, DWCS is capable of providing guaranteed service in less restrictive scenarios than those described in Section 3.2.

Having described some important characteristics of DWCS for real-time streams, we now describe how best-effort streams can be serviced with low average delay, while still guaranteeing service to real-time streams.

4. Heterogeneous Packet Streams

In many situations, it is desirable, or even necessary, to service a mixture of both real-time and best-effort packet streams. Many researchers have proposed that best-effort, or non-time-constrained streams are only scheduled when all real-time streams have been serviced. Other researchers, in real-time systems research [9, 3], have attempted to reduce the mean delay of non-time-constrained activities (such as threads or packets) by giving them precedence over real-time activities until it is essential to service the real-time activities.

(a) Scenario 1			
n	*D*	*V*	*U*
240	0	0	0.4830
320	0	0	0.6440
400	0	0	0.8050
480	0	0	0.9660
488	16664	0	0.9821
496	33328	0	0.9982
504	49992	14344	1.0143
512	66656	30295	1.0304
520	83320	44753	1.0465

(b) Scenario 2			
n	*D*	*V*	*U*
80	0	0	0.2810
160	0	0	0.5620
240	0	0	0.8430
256	0	0	0.8921
272	0	0	0.9554
280	20820	0	0.9835
288	49968	11215	1.0116
304	108264	54444	1.0678
320	166560	88544	1.1240

(c) Scenario 3			
n	*D*	*V*	*U*
480	0	0	0.9156
496	0	0	0.9461
504	0	0	0.9613
512	15152	0	0.9766
520	30990	0	0.9919
528	46828	7038	1.0071
544	78528	31873	1.0376
560	110240	53455	1.0681
640	268800	148143	1.2207

Table 2. Number of missed deadlines, D, and window-constraint violations, V, for increasing numbers of streams, n, and increasing utilization factors, U. In all scenarios, the number of window-constraint violations remains zero until the total utilization factor, U, exceeds 1.0.

For combined best-effort and window-constrained real-time streams, our approach is to service best-effort streams only when one packet from each and every real-time, window-constrained stream has been serviced in its current request period. Note that we only allow multiple packets in real-time streams to be serviced in the same request period if such streams have been *marked* as eligible for scheduling multiple times in one request period. For all normal cases, only one packet in a real-time stream can be serviced in any given request period. Furthermore, the next packet in a real-time stream has a deadline that is offset from the previous packet's deadline by the stream's request period.

$U_{min,WC}$	U_{BE}	$U_{min,WC} + U_{BE}$
0.0000	1.00	1.0
0.1526	0.84	0.9926
0.3052	0.68	0.9852
0.4578	0.52	0.9778
0.6104	0.37	0.9804
0.7630	0.21	0.9730
0.9156	0.05	0.9656
0.9461	0.02	0.9661
0.9613	0.0001	0.9614
0.9766	0.0	0.9766
0.9919	0.0	0.9919
1.0071	0.0	1.0071

Table 3. $U_{min,WC}$ is the minimum utilization factor of all the window-constrained (WC) streams when there are no window-constraint violations. $U_{min,WC}$ is *calculated* from *Scenario 3* in Section 3.2. In this scenario, U_{BE} is the *measured* utilization factor of best-effort streams when they are serviced. A best-effort stream is serviced when one packet from each and every real-time, window-constrained stream has been serviced in its current request period.

Table 3 shows the measured utilization factor, U_{BE}, of best-effort streams in the presence of real-time streams having the same service constraints as in *Scenario 3*, in Section 3.2. The number of real-time packet streams is increased, thereby increasing their utilization factor. $U_{min,WC}$ shows the minimum utilization factor of all the real-time, window-constrained (WC) streams when there are no window-constraint violations. If $U_{min,WC}$ were the measured utilization factor of real-time streams, then the best-effort streams could use the remaining fraction of bandwidth to minimize their delay. One way to minimize the delay of best-effort streams is to calculate a *pseudo* request period, T_{BE}, and window-constraint, W_{BE}, so that $1 - \sum_{i=1}^{n} \frac{(1-W_i)C_i}{T_i} = \frac{(1-W_{BE})C_{BE}}{T_{BE}}$, when there are n real-time, window-constrained streams. However, with this approach, there can be cases where real-time streams miss deadlines due to best-effort streams being serviced. In some cases, this may be acceptable, since each real-time stream only violates a tolerable number of packet deadlines, and does not violate its window-constraint. In other cases, we want to ensure real-time packet streams *never* miss deadlines when best-effort streams are serviced. Hence, our alternative approach is to service best-effort streams only when a packet from each and every window-constrained stream has been serviced in each real-time stream's current request period. This guarantees packets in real-time streams do not miss any deadlines due to servicing best-effort packet streams. From Table 3, the sum, $U_{min,WC} + U_{BE}$, is still close to 1.0. Since the *actual* utilization factor of both real-time and best-effort streams is 1.0, in this scenario, the real-time streams are actually experiencing a utilization factor above their minimum required utilization factor, $U_{min,WC}$. Consequently, there are some cases when a real-time, window-constrained stream is being serviced when it need not be serviced. This means the delay of best-effort packet streams is greater than the minimum possible delay. However, since $U_{min,WC} + U_{BE}$ is close to 1.0, the

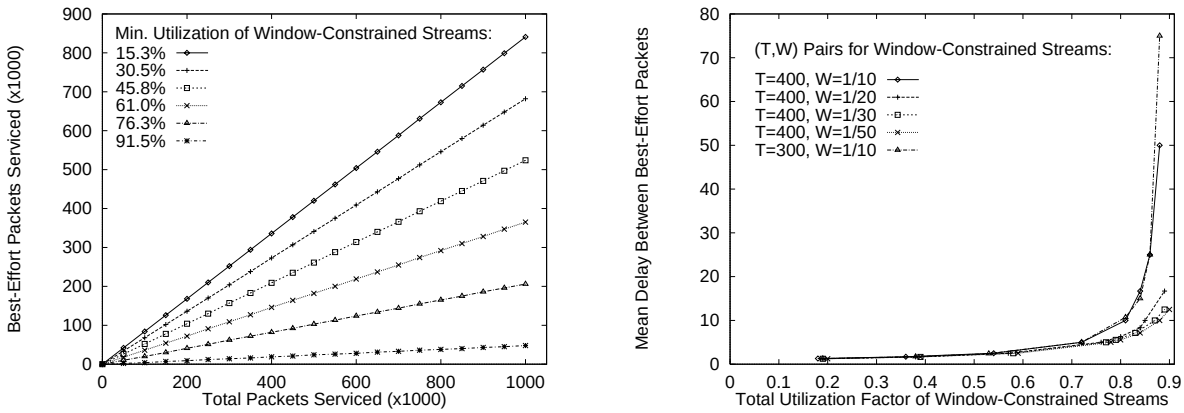

Figure 8. (a) The number of best-effort packets serviced, as a function of all packets serviced from both best-effort and real-time streams. The results are shown when real-time streams have different *minimum* utilization factors; (b) the mean delay (as a function of packet service times) between servicing consecutive packets in best-effort streams under different loads from real-time streams.

best-effort streams are experiencing close to the minimum possible delay.

Figure 8(a) shows the number of best-effort packet streams serviced, as a function of all packets serviced from both best-effort and real-time streams. Each set of real-time packet streams has a different utilization factor (hence, the six different lines in the graph). In all cases, the service constraints of real-time streams were the same as *Scenario 3*, in Section 3.2. The utilization factor of these real-time streams was increased, by increasing the number of streams in each of 8 different scheduling classes, from 10 to 60 streams per class. From the figure, it can be seen that there is a constant rate of service to best-effort streams at each of the different loads from real-time streams. This is useful, in that best-effort streams will not experience large variations in delay (and, hence, jitter) in the presence of real-time streams.

Figure 8(b) shows the mean delay between servicing consecutive packets in best-effort streams under different loads from real-time streams. The mean delay is a multiple of packet service times, which are assumed constant. Observe that, when real-time streams impose the same load but have different request periods, T, and window-constraints, W, then best-effort streams experience different delays. In fact, for loads above about 55%, there are different delays for best-effort packet streams that are dependent upon the service constraints of real-time streams. It can be seen that, for a given real-time load, increasing the request period, T, of real-time streams, can reduce the delay of best-effort streams. This is because only one packet in a real-time stream is serviced in its request period, T. By increasing T, for any given real-time load, there is a larger window of time to service best-effort streams, since real-time streams are not eligible for service again until their next request periods. Likewise, reducing the magnitude of the window-constraints (in this example, from $1/10$ to $1/50$), also reduces the delay incurred by packets in best effort streams. For any given real-time load, there will be fewer real-time streams if the utilization factor of each stream is higher. By reducing the window-constraints from $1/10$ to $1/50$, we are increasing the minimum required utilization factor of real-time packet streams. Consequently, there are fewer real-time streams that must be serviced in any request period, T. This means that more best-effort streams can be serviced in a given window of time. As a result, for higher loads, the mean delay of packets in best-effort streams is reduced, when real-time window-constraints are reduced.

Summary. For combined best-effort and window-constrained, real-time streams, our approach is to service best-effort streams only when one packet from each and every real-time, window-constrained stream has been serviced in its current request period. This is conservative, in that it ensures no packet in any window-constrained stream misses its deadline as a result of servicing best-effort streams. However, there may be cases when we want to allow some packets to miss their deadlines in real-time streams, as long as the window-constraints of these packet streams are not violated. In the latter case, we can calculate a *pseudo* request period, T_{BE}, and window-constraint, W_{BE} for best-effort streams. Furthermore, these best-effort packet streams can be prioritized to ensure precedence is given to the highest priority, non-time-constrained stream when it is safe to service such a stream. Finally, best-effort streams can be serviced at a constant rate. This minimizes the variations in delay of service to consecutive packets in

best-effort streams, in the presence of real-time streams.

5. Conclusions

This paper describes a modified version of Dynamic Window-Constrained Scheduling (DWCS) [38, 39]. DWCS was originally designed as a packet scheduler to provide (m, k)-*firm* deadline guarantees [15] and fair queueing [10, 41, 12, 2, 13, 30, 34] properties, for loss and delay constrained traffic streams such as multimedia audio and video streams. In this paper, we have shown a version of DWCS that 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. While the algorithm is capable of behaving as an EDF, static priority, or fair scheduler, it is able to guarantee window-constraints to real-time streams while ensuring fast response time to non-time-constrained best-effort streams.

DWCS can be thought of as a special case of pinwheel scheduling [17], in which a minimum of m deadlines are guaranteed to be met, every *fixed* (as opposed to *sliding*) window of k deadlines. However, DWCS has a least upper bound on resource (such as bandwidth) utilization [24] of 100%, independent of the window size, k. By comparison, Baruah and Lin [1] have developed an algorithm for pinwheel systems with a least upper bound on utilization that *asymptotically approaches* 100%, only when $k \rightarrow \infty$. Like Baruah and Lin's algorithm, DWCS is computationally efficient, with a cost linear in the number of streams, to determine the next packet for service. An important quality of DWCS is its ability to strategically miss packet deadlines *in overload conditions* in favor of more time-critical packets, thereby bounding the number of consecutive deadlines missed in a given stream.

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. More recently, algorithms such as DWCS are being used as part of work in sensor-based distributed computing projects at Boston University.

References

[1] S. K. Baruah and S.-S. Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47(7), July 1998.

[2] J. C. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *IEEE INFOCOMM'96*, pages 120–128. IEEE, March 1996.

[3] G. Bernat and A.Burns. Combining (n/m)-hard deadlines and dual priority scheduling. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, pages 46–57, San Francisco, December 1997. IEEE.

[4] G. Bernat, A. Burns, and A. Llamosi. Weakly-hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, April 2001.

[5] G. Bernat and R. Cayssials. Guaranteed on-line weakly-hard real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, December 2001.

[6] P. P. Bhattacharya and A. Ephremides. Optimal scheduling with strict deadlines. *IEEE Transactions on Automatic Control*, 34(7):721–728, July 1989.

[7] C. Carlsson and O. Hagsand. DIVE-a platform for multi-user virtual environments. *Computers and Graphics*, 17(6):663–669, November-December 1993.

[8] M. Chan and F. Chin. Schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, June 1992.

[9] R. Davis and A.Wellings. Dual priority scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 100–109. IEEE, 1995.

[10] A. Demers, S. Keshav, and S. Schenker. Analysis and simulation of a fair-queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990.

[11] D. Ferrari. Client requirements for real-time communication services. *IEEE Communications Magazine*, 28(11):76–90, November 1990.

[12] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOMM'94*, pages 636–646. IEEE, April 1994.

[13] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *IEEE SIGCOMM'96*. IEEE, 1996.

[14] S. Greenberg and D. Marwood. Real-time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the ACM Conference on Cooperative Support for Cooperative Work, ACM press*, pages 207–217. ACM, 1994.

[15] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, April 1995.

[16] J. M. Harrison. Dynamic scheduling of a multiclass queue: Discount optimality. *Operations Research*, 23(2):370–382, March-April 1975.

[17] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: A a real-time scheduling problem. In *Proceedings of the 22nd Hawaii International Conference of System Science*, pages 693–702, Jan 1989.

[18] K. Jeffay and S. Goddard. A theory of rate-based execution. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS)*, December 1999.

[19] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 110–117. IEEE, December 1995.

[20] R. Kravets, K. Calvert, P. Krishnan, and K. Schwan. Adaptive variation of reliability. In *HPN-97*. IEEE, April 1997.

[21] R. Krishnamurthy, K. Schwan, R. West, and M. Rosu. On network coprocessors for scalable, predictable media services. *IEEE Transactions on Parallel and Distributed Systems(TPDS) – to appear*, 2003.

[22] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West. Leveraging block decisions and aggregation in the sharestreams QoS architecture. In *Proceedings of the International Conference of Parallel and Distributed Systems (IPDPS)*, 2003.

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

[24] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[25] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of Operating Systems Design and Implementation*. USENIX, December 2002.

[26] A. K. Mok and W. Wang. Window-constrained real-time periodic task scheduling. In *Proceedings of the 22st IEEE Real-Time Systems Symposium*, 2001.

[27] J. Nieh and M. S. Lam. The design, implementatin and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*. ACM, October 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. *Journal of the ACM*, 35(4):832–844, October 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 Transactions on Networking*, 1(3):344–357, June 1993.

[30] X. G. Pawan Goyal and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *2nd Symposium on Operating Systems Design and Implementation*, pages 107–121. USENIX, 1996.

[31] J. M. Peha and F. A. Tobagi. A cost-based scheduling algorithm to support integrated services. In *IEEE INFOCOMM'91*, pages 741–753. IEEE, 1991.

[32] S. Singhal. *Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments*. PhD thesis, Stanford University, August 1996.

[33] D. Stiliadis and A. Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM Transactions on Networking*, 6(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. In *Real-Time Systems Symposium*. IEEE, December 1996.

[35] A. Striegel and G. Manimaran. Dynamic class-based queue management for scalable media servers. *Journal of Systems and Software*, 2003.

[36] R. West, I. Ganev, and K. Schwan. Window-constrained process scheduling for linux systems. In *Proceedings of the 3rd Real-Time Linux Workshop*, November 2001.

[37] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, December 2000.

[38] R. West and K. Schwan. Dynamic window-constrained scheduling for multimedia applications. In *6th International Conference on Multimedia Computing and Systems, ICMCS'99*. IEEE, June 1999. Also available as a Technical Report: GIT-CC-98-18, Georgia Institute of Technology.

[39] R. West, K. Schwan, and C. Poellabauer. Scalable scheduling support for loss and delay constrained media streams. In *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*. IEEE, June 1999.

[40] J. Xu and R. J. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *SICOMM 2002*, Pittsburgh, PA, August 2002. ACM.

[41] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM*, pages 113–121. ACM, August 1991.