Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams

Richard West and Christian Poellabauer

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

Abstract

This paper describes how Dynamic Window-Constrained Scheduling (DWCS) can guarantee real-time service to packets from multiple streams with different performance objectives. We show that: (1) DWCS can guarantee that no more than x packets miss their deadlines for every y consecutive packets requiring service, as long as the minimum *aggregate bandwidth requirement of all real-time packet streams does not exceed the available bandwidth, (2) using DWCS, the delay of service to real-time packet streams is bounded even when the scheduler is overloaded, (3) DWCS can ensure that the delay bound of any given stream is independent of other streams, and (4) a fast response time for best-effort packet streams, in the presence of real-time packet streams, is possible. Furthermore, if a feasible schedule exists, each stream is guaranteed a minimum fraction of available bandwidth over a finite window of time.*

1. Introduction

Many real-time, distributed applications, such as telemedicine, virtual environments, video-on-demand and streaming audio, can tolerate the loss of a certain fraction of all information transferred across a network to one or more clients. Information is lost if it is either received later than desired or not received at all by a client. For many such 'loss-tolerant' applications, there is usually a restriction on the number of *consecutive* packets of information that can be lost. For example, losing too many 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. More precisely, an application might tolerate x packet losses for every y arrivals at the various service points across a network. A suitable service discipline, for scheduling the transmission of packets from 'loss-tolerant' applications, must attempt to guarantee that no more than x packets are serviced late for every y packets requiring service.

This paper analyzes the real-time properties of Dynamic Window-Constrained Scheduling (DWCS) [18, 19], 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 packet stream. Furthermore, DWCS has the desirable property of supporting 'fair' allocation of bandwidth to packet streams, in proportion to their loss-constraints and per-packet deadlines [18]. In fact, DWCS can behave as a static-priority (SP), earliest-deadline-first (EDF), and fair scheduling algorithm [6, 20, 7, 2, 8, 15, 16]. Moreover, DWCS is intended to support multimedia traffic streams in the same manner as the SMART scheduler [14], but DWCS is less complex and requires maintenance of less state information than SMART.

Although DWCS has a lot in common with fair scheduling algorithms, it is more closely related to algorithms which attempt to guarantee service to packet streams, whereby at least m out of k packet deadlines are met for each and every stream. Hamdaoui and Ramanathan [9] were the first to introduce the notion of (m, k)-firm deadlines, which is similar to the concept of 'Skip-Over' by Koren and Shasha [12]. However, in some cases, skip over algorithms unnecessarily skip service to one or more activities (such as periodic tasks or packet streams), even if it is possible to meet the deadlines of those activities.

The (m, k)-firm deadline algorithm of Hamdaoui and Ramanathan guarantees that a statistical number of deadlines will be met, by using 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. 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 [10, 4, 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 $\{a_i \mid 1 \le i \le n\}$, 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 all resources are utilized, given that $k \to \infty$.

Other notable work includes Jeffay and Goddard's Rate-Based Execution (RBE) model [11]. 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,k)-hard 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 packet 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 can ensure the delay bound of any given stream is independent of other streams.

Note that, for the above 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.

The remainder of this paper is organized as follows: Section 2 describes a version of DWCS that provides (m,k)-hard service guarantees. Section 3 analyzes the performance of DWCS, while Section 4 describes an approach to effectively servicing best-effort packet streams without violating the service constraints of real-time packet streams. Finally, conclusions are described in Section 5.

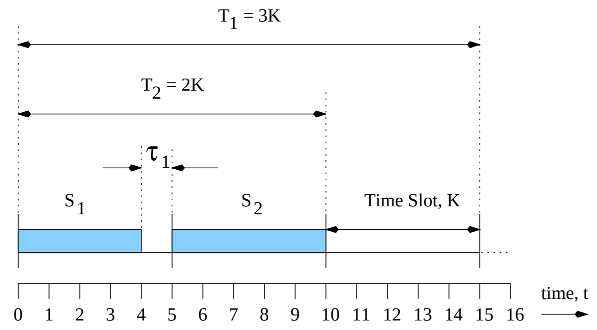

Figure 1. Example of two packets from different streams, S_1 and S_2 being serviced in their respective time slots. Each time slot is of constant size K. Observe that the packet in S_1 requires $K - \tau_1$ service time, thereby wasting τ_1 time units before the packet in S_2 is serviced.

2. Dynamic Window-Constrained Scheduling

We begin this section by defining the problem of guaranteeing a feasible schedule for real-time packet streams, which can tolerate at most x missed deadlines every fixed window of y requests. We then describe how the DWCS algorithm works, so that it can produce a feasible schedule on-line.

2.1. Problem Definition

In order to define the real-time scheduling problem addressed as part of this paper, we introduce the following definitions:

Bandwidth Utilization. This is a measure of the fraction (or percentage) of available bandwidth used by packet streams to meet their service constraints. A series of packet streams is said to *fully utilize* [13] available bandwidth, B, if all packet streams using B satisfy their service constraints, and any increase in the use of B violates the service constraints of one or more packet streams.

Dynamic Window-Constrained Scheduling (DWCS). DWCS is an algorithm for scheduling packet streams, each having the following pair of service attributes, which are used to define each stream's delay and loss constraints:

- *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. In this paper, all request periods are assumed to be multiples of a time slot, as shown in Figure 1.
- *Window-Constraint* – this is specified as a value 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. 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*.

Feasibility. A schedule, comprising a sequence of packet streams, is feasible if no original window-constraint of any packet 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 packet streams does not exceed 100% of available bandwidth. Before we analyze DWCS, we first describe the algorithm in more detail.

2.2. The DWCS Algorithm

This section describes a revised version of Dynamic Window-Constrained Scheduling (DWCS) from that described in [18, 19], so that *deterministic* real-time guarantees can be made for packet streams tolerating x missed deadlines every y requests. The original algorithm is work-conserving and only guarantees a statistical number of real-time service constraints. Moreover, the work-conserving nature of the original algorithm can sometimes cause a stream to be serviced earlier than necessary. This has the effect of reducing the likelihood of servicing the same stream at some time in the future, when it is actually more important to be serviced. However, the revised algorithm can operate in a non-work-conserving mode, to guarantee that no stream will be granted more than its minimum required service, when it is otherwise impossible to produce a feasible schedule (see Section 3.2).

The revised DWCS algorithm works as follows: packets are ordered 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. Whenever a packet in S_i misses its deadline, the window-constraint for S_i is modified in a way that reflects the increased importance of servicing S_i in the future. Conversely, any packet in a stream serviced before its deadline causes the corresponding stream's window-constraint to be modified in a manner that effectively reduces the priority of servicing subsequent packets in the same stream.

The window-constraint of a packet 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.

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.

Table 1 shows the rules for ordering pairs of packets. It differs slightly from the precedence rules used in the original design of DWCS [18]. It should be noted that DWCS is more than merely an earliest deadline first (or earliest due date) algorithm. Observe that earliest deadline first scheduling (EDF) considers that 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. 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. Observe that packets are ordinarily serviced in earliest deadline first order. However, if at least two packet 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 $d_i = d_j$ 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 from the stream with *tighter* window-constraints, since fewer consecutive late or lost packets from that stream can be tolerated. Likewise, if two packet streams have zero 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 follows:

(A) Window-constraint adjustment when a packet in S_i is serviced before its deadline:

```
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;
```

At this point in time, the window-constraint, W_j, of any other packet stream, $S_j \mid j{\neq}i$, comprising one or more late packets, is adjusted as follows:

(B) Window-constraint adjustment when a packet in $S_j \mid j \neq i$ misses its deadline:

```
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;
```

Observe 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. Consequently, the pseudo-code for DWCS is shown in Figure 2.

```
Let Si = Stream i
    di = deadline of the next packet in Si
    Ti = request period of Si
    Wi'= current window-constraint of Si

while (TRUE) {
  for (each packet in all streams eligible
       for service at the current time)
    find the next packet in stream, Si,
    with the highest priority,
    according to the rules in Table 1;
  service next packet in Si;
  adjust Wi' according to rules in (A);
  /* Adjust deadline of next
     packet in Si. */
  di = di + Ti;
  for (each packet in Sj, other than Si,
       missing its deadline) {
    while (deadline missed) {
      adjust Wj' according to rules in (B);
      if (current packet can be dropped)
        drop current packet in Sj;
      /* Adjust deadline of head packet
         in Sj. */
      dj = dj + Tj;
    }
  }
}
```

Figure 2. The DWCS algorithm.

Usually, a packet 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.

To support work-conservation, DWCS also allows packet streams to be *marked* as eligible for scheduling multiple times in the same request period. That is, 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 end of its request period. This implies $p_{i,j-1}$ is also 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 packet 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 packet streams with zero-valued current window-constraints. 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 and d_j, are equal, precedence is given to the packet 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.

To complete this section, Figure 3 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. Observe that, using DWCS, all window-constraints are met over non-overlapping windows of y_i deadlines (for each stream, S_i), and no time slots are unused in this example. 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 [13]. 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 [17]. By considering window-constraints when deadlines are tied, DWCS is able to make guarantees that EDF cannot.

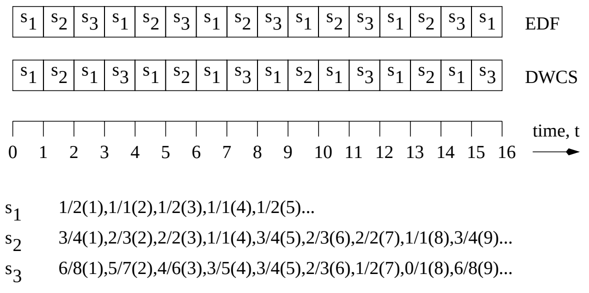
s1 s2 s3 s1 s2 s3 s1 s2 s3 s1 s2 s3 s1 s2 s3 s1
EDF
s1 s2 s1 s3 s1 s2 s1 s3 s1 s2 s1 s3 s1 s2 s1 s3
DWCS
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
time, t
s1 1/2(1),1/1(2),1/2(3),1/1(4),1/2(5)...
s2 3/4(1),2/3(2),2/2(3),1/1(4),3/4(5),2/3(6),2/2(7),1/1(8),3/4(9)...
s3 6/8(1),5/7(2),4/6(3),3/5(4),3/4(5),2/3(6),1/2(7),0/1(8),6/8(9)...

Figure 3. 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.

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 deadlines. This can lead to a worst-case complexity for DWCS of $O(n)$. However, 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 [19], requires $O(log(n))$ operations. Observe that there is an $O(1)$ cost per stream to update the corresponding service constraints, *after servicing a packet*.

An earlier paper [19] shows how DWCS can be approximated, to further reduce its scheduling latency, thereby improving service scalability 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.

Having described DWCS, we analyze the algorithm's performance in the following section.

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 packet stream (and, hence, packet) is still bounded. Again, the exact value of this maximum delay is characterized below.
- If the *minimum* aggregate bandwidth requirement of all real-time packet streams does not exceed the total available bandwidth, then a feasible schedule is guaranteed using DWCS. Moreover, 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.

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*[1].

[1]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 packet 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 details of this proof are shown in the Appendix.

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 packet stream will always be serviced before a lower priority packet 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 5). 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_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 packet 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 packet 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 packet streams that fully utilize all available bandwidth [13]. 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.

We now characterize the least upper bound on bandwidth utilization, assuming that at most one packet from any given stream is serviced in a single, fixed-sized time slot of size K, and all request periods are multiples of such a time slot. That is, $C_i \leq K, T_i = q_i K, x_i \geq 0, y_i > 0, \forall i \mid 1 \leq i \leq n$, K is a constant, and q_i is a positive integer.

Theorem 3 *Using DWCS, the least upper bound on the utilization factor is* 1.0*, if all streams comprise packets with the same service times, and all request periods are multiples of the packet service times. That is, DWCS is optimal in the sense that a feasible schedule exists if* $\sum_{i=1}^{n} \frac{(1-W_i)C_i}{T_i} \leq 1.0$*, given* $C_i = K$ *and* $T_i = q_i K$ *for* $q_i \in Z^+$*, where* Z^+ *is the set of positive integers.*

Proof The details of this proof are given in a full-length Technical Report [17]. Observe that, in Theorem 3, each packet is serviced for exactly K time units, which is the size of one time slot. This is a necessary condition, because packets are indivisible entities and, hence, cannot be preempted.

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 CPU scheduling, in which variable-length threads (or processes) can be preempted at fixed intervals (e.g., every $10mS$ timeslice). Moreover, ATM networks have fixed-length (53 byte) cells and the SAR component of the ATM Adaptation Layer segments application-level packets into cells, which 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, the per-stream service constraints are translated as follows. Let C_i be the *worst-case* service time of a packet in stream S_i before fragmentation, and let $c_i = K$ be the constant service time of each and every fragment. Likewise, let W_i and T_i be the window-constraint and request period, respectively, for a stream before fragmentation, while w_i and t_i are the translated window-constraint and request period, respectively, after fragmentation. Then:

$c_i = K, t_i = \lfloor \frac{T_i}{C_i} \rfloor K$ and $w_i = a_i/b_i$, where a_i and b_i are the smallest values satisfying $a_i/b_i = \frac{T_i c_i - C_i(1-W_i)t_i}{T_i c_i}$.

Example. Consider three streams, S_1, S_2 and S_3 with the following constraints: $(C_1 = 3, W_1 = 2/3, T_1 = 5)$, $(C_2 = 4, W_2 = 23/35, T_2 = 6)$ and $(C_3 = 5, W_3 = 1/5, T_3 = 7)$. 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 to be $(c_1 = 1, w_1 = 4/5, t_1 = 1)$, $(c_2 = 1, w_2 = 27/35, t_2 = 1)$ and $(c_3 = 1, w_3 = 3/7, t_3 = 1)$, 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. This fragment could be a single cell in an ATM network but, more realistically, it makes sense for one fragment to map to multiple ATM 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. 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. The following scenario was considered (other scenarios are described in a detailed Technical Report [17]):

There were 8 scheduling classes, $\rho_1 \cdots \rho_8$, for packet streams. The original window-constraints for the classes of packet streams, from ρ_1 to ρ_8, were $1/10, 1/20, 1/30, 1/40, 1/50, 1/60, 1/70,$ and $1/80$, respectively. Packets in streams belonging to ρ_1 and ρ_2 had request periods of 400 time units, while those in ρ_3 and ρ_4 had request periods of 480 time units. Remaining packets in streams belonging to ρ_5 and ρ_6 had request periods of 560 time units, while those in ρ_7 and ρ_8 had request periods of 640 time units. The number of packet streams in each case was uniformly distributed between each scheduling class, and a total of a million packets across all streams were serviced.

Table 2 shows the results of the above scenario. n is the total number of packet streams, D is the number of missed deadlines, V is the number of window-constraint violations, U is the *minimum* total utilization factor (as defined in Section 3.2) and $\frac{n}{8} \cdot \sum_{i=1}^{8} \frac{C_i}{T_i}$ is the utilization factor for all 8 classes *when all window-constraints are zero*. Observe that some packets miss their deadlines when U is less than 1.0, but only when $\frac{n}{8} \cdot \sum_{i=1}^{8} \frac{C_i}{T_i}$ is greater than 1.0. However, there are no window-constraint violations for any streams until U exceeds 1.0.

n	D	V	U	$\frac{n}{8} \cdot \sum_{i=1}^{8} \frac{C_i}{T_i}$
480	0	0	0.9156	0.9518
496	0	0	0.9461	0.9835
504	0	0	0.9613	0.9994
512	15152	0	0.9766	1.0152
520	30990	0	0.9919	1.0311
528	46828	7038	1.0071	1.0470
544	78528	31873	1.0376	1.0787
560	110240	53455	1.0681	1.1104
640	268800	148143	1.2207	1.2690

Table 2. Simulated results for 8 scheduling classes.

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 packet streams are only scheduled when all real-time packet streams have been serviced. Other researchers [5, 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.

One way to minimize the delay of best-effort packet 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 packet streams. However, with this approach, there can be cases where real-time packet streams miss deadlines due to best-effort packet 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 packet streams are serviced. Hence, our alternative approach is to service best-effort packet streams only when a packet from each and every window-constrained packet 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. Results have shown that best-effort packet streams typically experience close to their minimum possible delay with this latter approach [17].

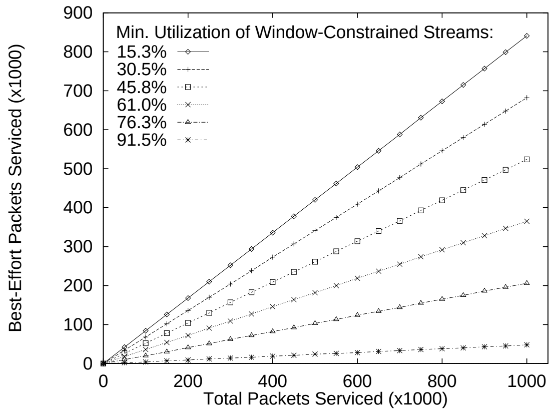

Figure 4. The number of best-effort packets serviced, as a function of all packets serviced from both best-effort and real-time streams.

Figure 4 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 *minimum* utilization factor (hence, the six different lines in the graph). In all cases, the service constraints of real-time packet streams were the same as in the simulated scenario in Section 3.4. The utilization factor of these real-time packet streams was increased, by increasing the number of packet streams in each of the 8 different scheduling classes, from 10 to 60 packet streams per class. From the figure, it can be seen that there is a constant rate of service to best-effort packet streams at each of the different loads from real-time packet streams. This is useful, in that best-effort packet streams will not experience large variations in delay (and, hence, jitter) in the presence of real-time packet streams.

5. Conclusions

This paper describes a modified version of Dynamic Window-Constrained Scheduling (DWCS) [18, 19]. DWCS was originally designed as a packet scheduler to provide (m,k)-*firm* deadline guarantees [9] and fair queueing [6, 20, 7, 2, 8, 15, 16] properties, for loss and delay constrained traffic streams such as multimedia audio and video streams. In this paper, we have shown: (1) a version of DWCS that can guarantee (m,k)-*hard* deadlines (or, equivalently, no more than x missed packet deadlines for every fixed window of y consecutive packets in a given stream), (2) using DWCS, the delay of service to real-time packet streams is bounded even when the scheduler is overloaded, (3) DWCS can ensure the delay bound of any given stream is independent of other streams, and (4) a fast response time for best-effort packet streams, in the presence of real-time packet streams, is possible.

A Appendix

A.1. 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 amongst all packets queued for service, (2) the time taken for W_i' to become the minimum amongst all current window-constraints, $W_k' \mid 1 \le k \le n$, 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, $y_j' \mid j{\neq}i, 1 \le j \le n$, amongst each packet 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_iT_i.

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

$$(x_i+1)T_i \tag{1}$$

Part (3): Assuming all packet 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 packet streams have the same or higher current window-denominators than S_i and every time another stream, S_j is serviced, deadline $d_j{\le}d_i$. To show that $d_j{\le}d_i$ 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 that are earlier than $t+C_j$ are incremented by a multiple of the corresponding request periods, $T_k \mid 1 \le k \le n$, depending on how many request periods have elapsed while servicing S_j. The worst-case is that $T_j{\le}T_i, \forall j \ne i$. 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, $y_j' \mid j{\neq}i, 1 \le j \le n$, amongst all packet 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 $T_j{\le}T_i, \forall j \ne i$ and T_j is finite. For n packet 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 packet stream, $S_j \mid j \ne i$, has a request period, $T_j = K$, then S_i will experience its worst delay before $y_i' \ge y_j'$. 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 5 shows the worst-case situation for three packet 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, $y_i'{\ge}max[y_l', y_m']$ 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, $y_i'{\ge}max[y_l', y_m', y_o']$ is true when $y_i' = y_{l_\phi}' + 2 = y_{m_\phi}' + 2 = y_{o_\phi}' + 2$. By induction, for each of the $n-1$ packet streams, $S_j \mid j{\neq}i, 1{\le}j{\le}n$, other than S_i, each with $T_j = K$ and $y_{j_\phi}'{\ge}y_{i_\phi}'$, $y_i'{\ge}max[y_1', y_{i-1}', \cdots, y_{i+1}', y_n']$ 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 $\delta_i{\le}T_i(y_{j_\phi}' - y_{i_\phi}' + (n-2))$.

Now observe that $y_{j_\phi}'{\le}y_j$ 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 $y_j = max[y_1, y_{i-1}, y_{i+1}, y_n]$,

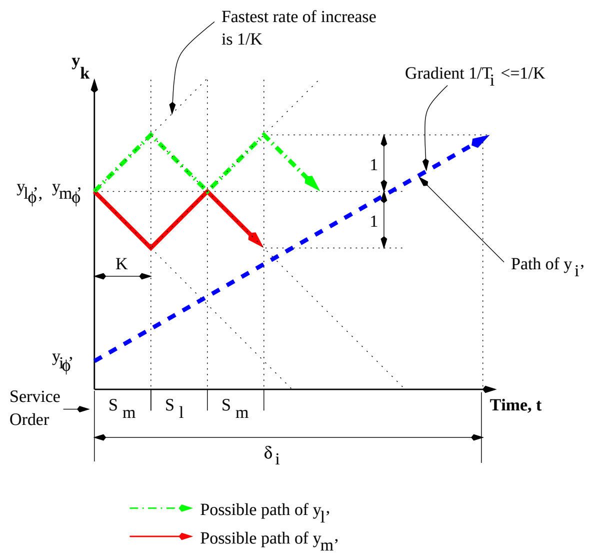

Figure 5. The change in current window-denominators, y_i', y_l' and y_m' for three packet streams, S_i, S_l and S_m, respectively, when all request periods, except possibly T_i, are finite. The initial state, ϕ, is when all current window-constraints first equal 0, and the current window-denominators are all greater than 0.

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

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

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

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] 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.

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] J. Nieh and M. S. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*. ACM, October 1997.

[15] 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.

[16] 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.

[17] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. Technical Report GIT-CC-00-20, Georgia Institute of Technology, College of Computing, 2000.

[18] 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.

[19] 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.

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