Dynamic Window-Constrained Scheduling for Multimedia Applications *

Richard West and Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

This paper describes an algorithm, called Dynamic Window-Constrained Scheduling (DWCS), designed to meet the service constraints on packets from multiple, network-bound media streams with different performance objectives. Using only two attributes, a deadline *and a* loss-tolerance *per packet stream, DWCS: (1) can limit the number of late packets over finite numbers of consecutive packets in loss-tolerant or delay-constrained, heterogeneous traffic streams, (2) does not require a-priori knowledge of the worst-case loading from multiple streams to establish the necessary bandwidth allocations to meet per-stream delay and loss-constraints, and (3) can exhibit both fairness and unfairness properties when necessary. In fact, DWCS can perform fair-bandwidth allocation, static priority (SP) and earliest-deadline first (EDF) scheduling. We show the effectiveness of DWCS using a streaming video application, running over ATM.*

1 Introduction

This paper describes a novel packet scheduling algorithm, called Dynamic Window-Constrained Scheduling (DWCS), that resides at the base of an end-system quality of service (QoS) architecture. DWCS is designed to maximize network bandwidth usage in the presence of multiple packets each with their own delay constraints and loss-tolerances. The algorithm requires two attributes per packet stream, as follows:

- *Deadline* – this is the latest time a packet can *commence* service. The deadline is determined from a specification of the maximum allowable time between servicing consecutive packets in the same stream.
- *Loss-tolerance* – this is specified as a value x_i/y_i, where x_i is the number of packets that can be lost or transmitted late for every *window*, y_i, of consecutive packet arrivals in the same stream, i. Hence, for every y_i packet arrivals in stream 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 have the same loss-tolerance, while each successive packet in a stream has a deadline that is offset by a fixed amount from its predecessor.

DWCS has the ability to share bandwidth among competing clients in strict proportion to their deadlines and loss-tolerances. This is similar to (weighted) fair scheduling[1, 2, 3, 4], which attempts to allocate bandwidth in proportion to stream *weights*. Similar proportional share algorithms have been targeted at CPU scheduling[5, 6]. However, the idea of 'windowing' in DWCS is closer to the work of Hamdaoui and Ramanathan[7] who have simulated an algorithm that services multiple streams, in an attempt to ensure at least m customers (packets or threads) in a stream (or process) meet their deadlines for every k consecutive customers from the same stream (or process). In comparison, DWCS can also perform static priority and earliest-deadline first scheduling, supporting both deadline and non-deadline constrained traffic. Furthermore, DWCS can meet explicit delay and 'windowed' loss constraints, using only two attributes that enable a diverse range of service specifications. We now discuss the DWCS algorithm in more detail.

2 The DWCS Algorithm

Dynamic Window-Constrained Scheduling (DWCS) orders packets for transmission based on the *current* values of their loss-tolerances and deadlines. Precedence is given to the packet at the head of the stream with the lowest loss-tolerance. Packets in the same stream all have the same original and current loss-tolerances, and are scheduled in their order of arrival. Whenever a packet misses its deadline, the loss-tolerance for all packets in the same stream, s, is re-

*This work is supported in part by DARPA through the Honeywell Technology Center under contract numbers B09332478 and B09333218, and by the British Engineering and Physical Sciences Research Council with grant number 92600699.

duced to reflect the increased importance of transmitting a packet from s. This approach avoids starving the service granted to a given packet stream, and attempts to increase the importance of servicing any packet in a stream likely to violate its original loss constraints. Conversely, any packet serviced before its deadline causes the loss-tolerance of other packets (yet to be serviced) in the same stream to be increased, thereby reducing their priority.

Pairwise Packet Ordering
Lowest loss-tolerance first
Same non-zero loss-tolerance, order EDF
Same non-zero loss-tolerance & deadlines, order lowest loss-numerator first
Zero loss-tolerance & denominators, order EDF
Zero loss-tolerance, order highest loss-denominator first
All other cases: first-come-first-serve

Table 1: Precedence amongst pairs of packets

Table 1 shows the rules for ordering pairs of packets in different streams. Observe that, if two packets have the same non-zero loss-tolerance and deadline they are ordered lowest loss-numerator x_i first, where x_i/y_i is the current loss-tolerance for all packets in stream i. By ordering on the lowest loss-numerator, precedence is given to the packet in the stream with *tighter* loss constraints, since fewer consecutive packet losses can be tolerated. If two packets have zero loss-tolerance and their loss-denominators are both zero, they are ordered EDF, otherwise they are ordered highest loss-denominator first. In such a circumstance, it is possible that a stream may lose more packets than its loss-tolerance specification allows. By increasing the denominator in this case, the algorithm attempts to favor the adversely affected packet stream, bringing the amortized loss for packets in that stream back to the original loss-tolerant value.

Every time a packet in stream i is transmitted, the loss-tolerance of i is adjusted. Likewise, other streams' loss-tolerances are adjusted *only if* any of the packets in those streams miss their deadlines as a result of queueing delay.

For streams that can lose packets, any packets in these streams that have missed their deadlines are simply discarded. For a stream that cannot lose packets, the deadline serves to minimize queueing delay before eventual transmission of all packets in that stream. The loss-tolerance value for such streams serves to avoid transmitting too many late packets.

We now describe how loss-tolerances are adjusted. Let x_i/y_i denote the original loss-tolerance for all packets in stream i. Let x_i'/y_i' denote the current loss-tolerance for all queued packets in stream i. Let x_i' denote the current loss-numerator, while x_i is the original loss-numerator for packets in stream i. y_i' and y_i denote current and original loss-denominators, respectively. Before a packet stream is serviced, its current and original loss-tolerances are equal. For all buffered packets in the same stream i as the packet most recently transmitted before its deadline, adjust the loss numerators and denominators as follows:

if $(y_i' > x_i')$ then $y_i' = y_i' - 1$;
if $(x_i' = y_i' = 0)$ then $x_i' = x_i; y_i' = y_i$;

Observe that loss-tolerances do not change for streams without deadlines. However, for all buffered packets, if any packet in stream $j | j \neq i$ 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 $(x_j' = 0)$ then
  if $(x_j > 0)$ then $y_j' = y_j' + \lceil \frac{y_j - x_j}{x_j} \rceil$;
  if $(x_j = 0)$ then $y_j' = y_j' + y_j$;

As an example, consider $n = 3$ streams of packets, s_1, s_2 and s_3 (see Figure 1). Let the original loss-tolerances of each stream be 1/2, 3/4 and 6/8, respectively. Let the deadlines of the first packets in each stream be 0 and let each successive packet p in stream i, have a deadline one time unit later than predecessor $(p-1)$ in the same stream i. That is, $deadline_{1_i} = 0$ and $deadline_{p_i} = deadline_{(p-1)_i} + 1$, $\forall i, 1 \leq i \leq n, p \in Z^+$, where Z^+ is the set of positive integers. Assume that the service time of each packet is one time unit. If each stream has a packet arrive for service once every time unit, the total load on the scheduler is 3.0 from all three streams. However, due to the loss-tolerances of each stream, the minimum demand from all streams is $\sum_{i=1}^{n} \frac{(1-l_i)C_i}{T_i}$, where l_i is the loss-tolerance, C_i is the service time (or transmission delay) of each and every packet in stream i, and T_i is the inter-arrival time for packets in stream i (which is also the time between successive packet deadlines). For this example, with the three streams having the above loss-tolerances, the effective scheduler load can be as low as 1.0 if we carefully discard (or service late) appropriate late packets from each stream. Thus, it may still be possible to service all three streams while meeting the appropriate losses from each stream.

From Figure 1, the first packet to be scheduled in this case will be from s_1, because s_1 has the lowest loss-tolerance. Since the serviced packet does not miss its deadline, the new (current) loss-tolerance of s_1 will be set to 1/1. As a result, we can still allow the loss of the next packet in s_1 and not violate the original loss tolerance. Hence, the rationale for adjusting the loss-tolerance in this way. At time $t = 1$, the first packet in s_1 has been serviced but the first packets in s_2 and s_3 have each missed their deadlines. As a result, the first packet in each of these streams is dropped and the new loss-tolerances for s_2 and s_3 are set to $2/3$ and $5/7$, respectively. This change in loss-tolerance compensates for one less allowable packet loss over a range of one fewer packets than in the original loss-tolerance specification. At time $t = 1$, the packet at the head of s_2 with $deadline = 1$ has the highest priority, so it is serviced next. This causes the packets with $deadline = 1$ from s_1 and s_3 to miss their deadlines. Observe that s_1's loss-tolerance is set back to its original value at this point, because it was temporarily set to $0/0$, which is meaningless. Notice that at time $t = 5$, a packet in s_2 gets serviced. When two packets have the same non-zero loss-tolerance and deadline, and their loss-numerators are the same, ties can be broken arbitrarily. Every four time units, the schedule repeats itself. Observe that over the first eight packets serviced, s_1 transmits four packets and loses four, consuming 50% of the bandwidth, and both s_2 and s_3 transmit two packets and lose six, each consuming 25% of the bandwidth. Furthermore, one packet from s_1 is serviced every two time units (or packet service times), one packet from s_2 is serviced every four time units, and two packets from s_3 are serviced every eight time units. Hence, the loss-tolerances from all three streams are met.

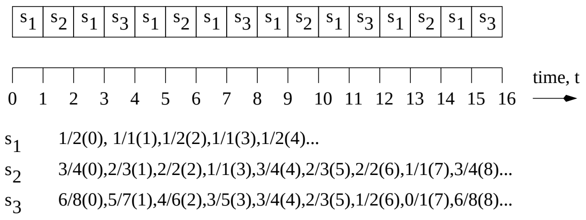

Figure 1: Example DWCS scheduling of 3 streams, s_1, s_2 and s_3. Deadlines are shown in brackets and loss-tolerances are shown as x/y.

We now describe some features that show the flexibility of DWCS.

Earliest-Deadline First Scheduling using DWCS

When the loss-tolerances of all packets in each stream are set to $0/0$, packets are scheduled in EDF order. 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. 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[8].

Static Priority Scheduling using DWCS

If no packets in any streams have deadlines (ie., 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[9]. In DWCS, the current loss-tolerances associated with each packet in every stream are always equal to their original loss-tolerances, and each packet's loss-tolerance serves as its static priority. Namely, for packets with infinite deadlines, the term "loss-tolerance" is really a misnomer, since no packets are actually lost. As expected, precedence is given to the packet with the lowest "loss-tolerance" (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 such as SFQ[4] have the ability to share bandwidth among n message streams such that each stream receives a weighted fair-share of available bandwidth. Specifically, let w_i be the weight of message stream i and $B_i(t_1, t_2)$ be the aggregate service (in bits) of i in the interval $[t_1, t_2]$. If we consider two message streams, i and 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, as shown in Figure 1. Given stream weights, w_i, in a fair bandwidth-allocating algorithm, we can calculate the loss-tolerances 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 $w_i | 1 \leq i \leq n, w_i \in Z^+$: First, let C_i be the service time of each packet in stream i. (This assumes all packets in any one stream are the same length). Let $\omega = \sum_{i=1}^{n} w_i$ and let η_i be the number of packets from stream i serviced in some arbitrary time window Δ. (Note that $\eta_i C_i$ is the total service time of stream 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 $deadline_{1_i} = 0$, and $deadline_{p_i} = deadline_{(p-1)_i} + C_i$, for each packet p_i in stream i, where $p \in Z^+$.
3. To calculate the loss-tolerance, l_i, of packets in stream i, let $l_i = \frac{x_i}{y_i}$, where: $y_i = \frac{\Delta_{min}}{C_i}$, $x_i = \frac{\Delta_{min}}{C_i} - \eta_i'$, and $\eta_i' = \frac{\eta_i \Delta_{min}}{\Delta} = \frac{w_i \Delta_{min}}{\omega C_i}$.

If deadlines are assigned as in step 2, we can translate packet loss-tolerances back into stream *weights*, w_i, as follows: $w_i = \frac{y_n(y_i - x_i)}{y_i(y_n - x_n)}$, where $0 < \frac{x_i}{y_i} < 1$.

3 Experimental Evaluation

All experiments were performed on a cluster of SparcStation Ultra II Model 2148s. Streams of MPEG-1 video frames are placed into shared memory queues, ready for packetization and scheduling. At any point in time there are between one and n active streams $s_i | 1 \le i \le n$, each with their own service attributes.

Fair-Bandwidth Allocation

The first experiment compares DWCS to SFQ in its ability to achieve fair-bandwidth allocation (link sharing) amongst n streams in the shortest time-frame possible. Weights are assigned to the streams in SFQ, and the corresponding loss-tolerances and deadlines, for DWCS, are computed using the method in Section 2.

Four streams requiring service, comprise bursts of 150 frames with an average arrival rate of 30 frames per second, followed by idle periods with a mean inter-burst gap of 1 second. The burst periods average 5 seconds and these bursts are repeated 10 times per stream, for a total of 1500 frame (or packet) arrivals per stream. All packets are eventually transmitted, even if they are late. The scheduler interval is $40mS$, so 25 frames can be serviced in 1 second. For these experiments, the packet service times are assumed to be equal to the scheduler period, since the scheduler services at most one packetized frame each time it executes. This scenario overloads the scheduler and forces a build-up of back-logged arrivals.

Figure 2 shows the bandwidths (bit service rates) of the four streams over a 50 second period, for SFQ and DWCS. The weights for streams $s_1, ..., s_4$ are 1, 1, 2 and 4, respectively. The corresponding loss-tolerances are 7/8, 14/16, 6/8 and 4/8. In the steady-state, SFQ and DWCS behave almost identically, each servicing s_1 and s_2 at about 110 Kbps, s_3 at about 220 Kbps, and s_4 at about 440 Kbps.

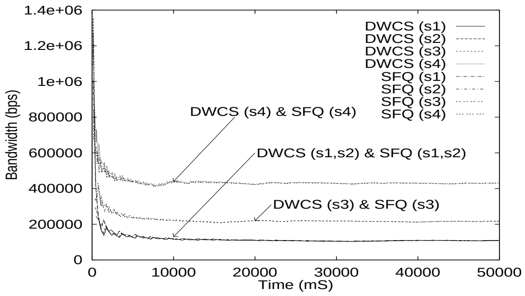

Figure 2: Bandwidth allocations for 4 streams, $s_1, ..., s_4$, with weights, 1, 1, 2 and 4, respectively. The corresponding loss-tolerances are 7/8, 14/16, 6/8 and 4/8.

By minimizing the window of time over which bandwidth is allocated in proportion to weights of streams in SFQ, SFQ actually manages to meet the loss-tolerances and deadlines of packets almost as successfully as DWCS. However, DWCS explicitly uses deadlines and loss-tolerances for packet streams. Hence, bandwidth can be allocated to streams to meet these constraints, independent of the constraints on other streams, assuming enough bandwidth is available, which is the case when $\sum_{i=1}^{n} \frac{(1-l_i)C_i}{T_i} \le 1.0$. Note that in the experiments above, at no time did DWCS violate the loss-tolerances on any of its streams even though the scheduler was overloaded and had to service some packets late.

Out-of-Band Traffic

Consider the scenario where stream s_1 requires twice as much bandwidth as s_2 when both streams are active. However, suppose stream s_3 is carrying time-critical (out-of-band) traffic that must be delivered to its destination with the shortest possible delay. In this case, we always want to grant service to s_3 when s_3 has packets for service, but when s_3 is not active, the bandwidth must be shared between s_1 and s_2.

Fair-scheduling algorithms cannot handle the above scenario but DWCS can, because for the duration of a burst of packets from s_3, that burst must be serviced exclusively. This violates the fairness properties of fair schedulers. In contrast, with DWCS, by carefully choosing a loss-tolerance that reflects the highest priority for s_3, and by setting the packet deadlines to infinity, DWCS gives exclusive service to s_3 when it is active. However, as s_1 and s_2 are starved of service, their loss-tolerances *dynamically* decrease, thereby raising their priorities until it is possible that they have precedence over s_3. The loss-tolerance of

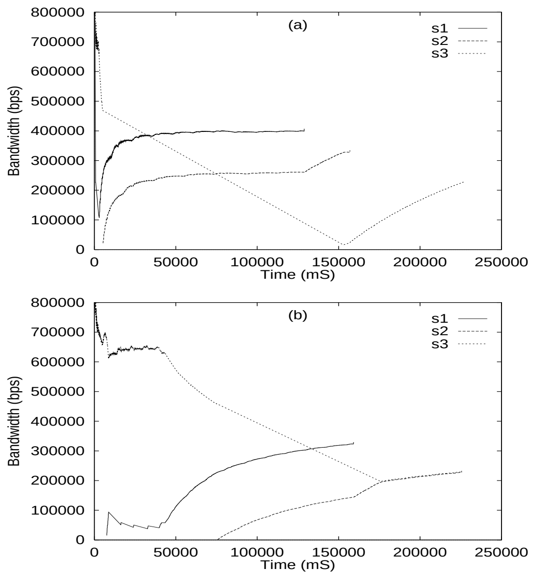

Figure 3: Bandwidth versus time for two dynamic priority streams, s_1 and s_2, and a static priority stream, s_3. (a) Loss-tolerances for s_1, s_2 and s_3 are 1/3, 2/3 and 0/100, respectively. (b) Loss-tolerances for s_1, s_2 and s_3 are 1/3, 2/3 and 0/1500, respectively.

s_3 traffic must be set so that the maximum burst size from s_3 does not require a longer uninterrupted service duration than the time to raise the priority of either s_1 or s_2 above that of s_3.

Figure 3(a) shows the results of servicing 3 streams, in which the first two, s_1 and s_2, are assigned dynamic priorities (ie., loss-tolerances of 1/3 and 2/3, respectively) that reflect their bandwidth shares, while s_3 is assigned a static priority of 0/100. Initially, s_3 receives a greater service rate than either s_1 or s_2, but as s_1 and s_2 are neglected, their loss-tolerances decrease.

If the loss-tolerance for s_3 is reduced even further, by increasing its denominator, larger durations of service time are granted to consecutive packet arrivals from s_3 (as shown in Figure 3(b) where the loss-tolerance of s_3 is 0/1500). Observe that $0/y_1$ is higher priority than $0/y_2$ if $y_1 > y_2$. This is one of the precedence rules described in Table 1. Hence, we can fine-tune stream attributes, so that the loss-tolerance of out-of-band data reflects the time to service the largest traffic burst without interruption. Observe that in Figures 3(a) and (b), when the bandwidth curve for s_3 decreases, it is actually not being serviced. Meanwhile, s_1 and s_2 approach their steady-states, close to 2 : 1 bandwidth shares.

4 Conclusions

DWCS has the ability to limit the number of late packets over finite numbers of consecutive packets in loss-tolerant or delay-constrained, heterogeneous traffic streams. DWCS can support a combination of static priority and dynamic priority (bandwidth-allocated) traffic. In fact, DWCS can perform fair-bandwidth allocation, static priority (SP) and earliest-deadline first (EDF) scheduling. However, unlike fair-scheduling algorithms, DWCS *can be unfair when necessary*, as well as showing all the fairness characteristics of fair-schedulers such as SFQ. By being unfair when necessary, DWCS can schedule out-of-band data in the presence of other loss-tolerant and delay-constrained traffic.

DWCS can meet explicit delay and 'windowed' loss constraints, using only two attributes that enable a diverse range of service specifications. Further details of the algorithm are described in an accompanying paper[10].

References

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, pp. 3–26, October 1990.

[2] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *INFOCOMM'94*, pp. 636–646, IEEE, April 1994.

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

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

[5] 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.

[6] C. A. Waldspurger and W. E. Weihl, "Stride scheduling: Deterministic proportional-share resource management," Tech. Rep. MIT/LCS/TM-528, MIT, 1995.

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

[8] 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*, vol. 35, pp. 832–844, October 1988.

[9] J. M. Harrison, "Dynamic scheduling of a multiclass queue: Discount optimality," *Operations Research*, vol. 23, pp. 370–382, March-April 1975.

[10] R. West and K. Schwan, "Dynamic window-constrained scheduling for multimedia applications," Tech. Rep. GIT-CC-98-18, Georgia Institute of Technology, 1998.