

Experimental Studies of Granularity Aware (m, k) Scheduling for Real-time Media Servers

Yingxin Jiang, Xiaolong Li and Aaron Striegel
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
{yjiang3,xli5,striegel}@nd.edu

Abstract—Real-time media servers are becoming increasingly important as the Internet supports more and more multimedia applications. In order to meet these ever increasing demands, real-time media servers will be responsible for supporting a large number of clients with a wide range of QoS requirements. In this paper, we propose a granularity aware (m, k) scheduler (GAS) which successfully controls the tradeoff between QoS granularity and scalability. We propose algorithms for group balance, stream join, burst sending, and feedback that exploit individual stream characteristics while avoiding per-stream state cost. Besides presenting detailed examples of GAS, we evaluate our work through simulation studies and experiments.

I. INTRODUCTION

Real-time media servers are becoming increasingly important as the Internet supports more and more multimedia applications such as video-on-demand and teleconferencing. In order to meet these ever increasing demands, real-time media servers will be responsible for supporting a large number of clients with a wide range of quality of service (QoS) requirements.

From the perspective of the real-time media server, the *deadline* of a packet can be viewed as the latest time by which the packet can be scheduled for transmission in order to reach the client on time. Packets that do not reach the client within their respective deadlines contain stale information that can be useless [1]. Thus, a late packet can be viewed in the same category as a lost packet. Although loss rate can be used as an overall QoS metric [2], it cannot express the information about how the sent or dropped packets are distributed. For example, if a few consecutive audio packets miss their deadlines, a vital portion of the talkspurt may be missing and the quality of the audio signal may not be satisfactory even though the total loss rate of this audio stream may not be high at all. However, if the lost packets are adequately spaced, the quality of this audio signal may be satisfactory. Several streams in the media

server may be able to tolerate a loss of a fraction of their packets in a specific duration with little or no degradation of the QoS received by the client. For such *loss-tolerant* applications, their performances are usually sensitive to the dropout patterns and loss rate is not necessarily a meaningful QoS metric.

In the literature, a variety of schemes have been proposed to minimize dynamic failures¹ [3]–[5] and a variety of schemes [6]–[8] have been implemented based on Linux. In the DBP (Distance Based Priority) [3] and DWCS (Dynamic Window-Constrained Scheduling) [5] schemes, dynamic failure rates are kept low through scheduling based on the distance or tolerance of a stream for additional losses before dynamic failure occurs. However, since a one-to-one mapping of stream to state information is maintained, the performance increase comes at the cost of scalability. In contrast, most scalable schemes (RMS [9], EDF, class-based scheduling [10]) ignore stream-wise information at the cost of dynamic failure performance. In [4], the DCQM (Dynamic Class-Based Queue Management) model tried to balance scalability and QoS granularity by adding the concept of a *group* to aggregate state information. Since the DCQM model did not exploit the service constraints of individual streams when multiplexing streams into groups, the performance of DCQM was not satisfying.

This tradeoff between the scalability and QoS granularity introduces the motivation for our project, namely can one preserve per-stream-state performance without sacrificing scalability?

The rest of the paper is structured as follows. Section II details the related work to our paper. Section III describes the GAS model. Next, Section IV details the implementation of the GAS model in Linux. Then, Section V and Section VI present simulation and experiment

¹When the number of lost packets over a window exceeds the maximum tolerable value, a dynamic failure occurs.

studies of our scheduler. Finally, in Section VII, we make several concluding remarks.

II. RELATED WORK

In [3], the (m, k) model was proposed for capturing the loss constraints of a real-time stream. A stream in the (m, k) model with m and k parameters states the following QoS requirement: for every k consecutive packets in the stream, at least m packets must meet their deadlines. In the DBP (Distance Based Priority) scheme [3], a state is maintained for each stream that represents the history of the last k packets transmitted or dropped. The DBP value of a stream, which is associated with the state, is the number of transitions required to reach a failing state, where failing states are those states in which dynamic failure has occurred. The lower the DBP value, the higher the priority. At any time, the packet from the stream with the highest priority is selected for transmission.

[11] proposed an analytical model for computing the probability of dynamic failure in both the single priority scheme and the DBP scheme. By comparing the predicted dynamic failure to the one obtained through simulation, the analytical model works for low and moderate loads. In addition, the model shows again that for loss-tolerant applications, the DBP scheme provides a higher performance than the single priority scheme.

The EDBP algorithm in [12] let the scheduler discern the levels of dynamic failure between different streams. The EDBP algorithm uses a modification of the DBP state calculation that allows the DBP value to be negative: When the stream reaches a failing state, EDBP expands upon the initial DBP algorithm by setting the EDBP value equal to one minus the number of transitions to return to a non-failing state. In the initial DBP model, however, as long as the stream has reached a failing state, the DBP value is equal to zero.

In [5], the DWCS (Dynamic Window-Constrained Scheduling) scheme was proposed to deal with loss-tolerant streams expressed by the (x, y) model. In addition, DWCS can behave as a static-priority (SP), earliest-deadline-first (EDF), and fair scheduling algorithm [13], [14]. In the (x, y) model, there are no more than x packets lost for every y consecutive packets. The relationship between the (m, k) model and the (x, y) model can be stated as $(x, y) = (k - m, k)$. Although both DBP and DWCS maintain per-stream state information, maintenance of the state information in the two schemes is considerably different. While DBP uses the notion of state transitions where every state represents the condition of k consecutive packets belonging to one stream by using k bits; DWCS uses the notion of a

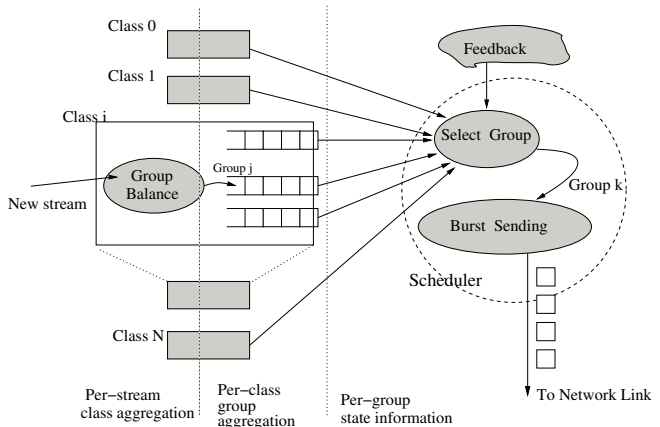


Fig. 1. GAS model

dynamic window whereby the values of x and y are changed according to whether or not a packet of the stream is dropped.

A new algorithm, Virtual Deadline Scheduling (VDS), to window-constrained scheduling was presented in [15]. VDS is suitable for weakly-hard real-time systems which means jobs can be serviced by the time later than their deadlines for a certain degree. For each job, VDS derives a virtual deadline according to the job's real-time deadline and the current window-constraint. The virtual deadline may be some finite time after the corresponding real-time deadline. Then VDS does the similar scheduling decision as DWCS by using virtual deadlines. Simulation shows that VDS can provide better window-constrained guarantees than other related algorithms.

To balance the scalability and QoS granularity in a media server, the DCQM (Dynamic Class-Based Queue Management) model was proposed in [4]. The DCQM model uses the concept of a *group* to allow flexibility between the two extremes of per-stream QoS and per-class QoS by varying the number of streams that can be multiplexed. State information is maintained on a per-group basis with the DBP or DWCS scheduler applied on the basis of group state. At any time, the packet from the group with either the highest DBP or DWCS priority is selected for transmission. As the number of streams multiplexed into the same group increases, the accuracy of the per-group state decreases. Thus, although a given group may meet its QoS requirements, the individual streams multiplexed into this group may not meet their QoS requirements.

III. GAS(GRANULARITY AWARE (m, k) SCHEDULING) OVERVIEW

Figure 1 shows the model of GAS (Granularity Aware (m, k) Scheduler). Scheduling in the GAS model is similar to the original DCQM model [4]. Streams are aggregated into groups and history (DWCS or DBP) is maintained on a group-wise basis. Scheduling is performed from the perspective of groups, selecting the first packet from the group queue. As packets are scheduled or dropped from a group, the group-wise history is updated appropriately. To capture the tradeoff between QoS granularity and scalability more accurately, GAS incorporates the following aspects:

- *Group balancing*: The group balancing algorithm is invoked on a stream join to correct any unbalanced stream distributions.
- *Burst sending*: Rather than selecting the “best” group at each possible scheduling point, GAS uses the notion of burst sending for B (Burst Window Size) packets from the same group.
- *Client feedback*: The feedback from the clients can be used to improve the performance of the system.

A. Group Balancing

If the streams multiplexed into one group have significantly different deadlines, the streams with lax deadlines will receive better performance than the streams with tight deadlines inside of a group-wise FIFO queue since the packets with lax deadlines will have a higher probability of being at the head of the queue. The following shows how a DBP-based DCQM scheduler works when the streams with different deadlines are multiplexed into one group. When a group is selected to transmit a packet, the packet with the earliest arrival time will be chosen.

Example 1: Consider a real-time application with two groups, G_0 and G_1 ($G = 2$) and four periodic streams, $S_0, S_1, S_2,$ and S_3 . The streams and the groups have the same (m, k) value, $(2, 3)$. The periods of S_0, S_1, S_2, S_3 are 2, 2, 3, and 3, respectively. The deadlines of S_0, S_1, S_2, S_3 are 3, 8, 3, and 8, respectively. Stream S_0 and S_1 belong to G_0 ; S_2 and S_3 belong to G_1 .

Figure 2 shows the resulting schedule from a DBP-based DCQM scheme. At $t = 0$, the DBP values of G_0 and G_1 are equal to 0. Until $t = 19$, the number of dynamic failures in G_0 and G_1 are 0 and 1 respectively. However, the actual number of dynamic failures in $S_0, S_1, S_2,$ and S_3 are 2, 0, 4, and 0². In group G_0 , since S_1 has a more lax deadline than S_0 , more packets belonging to S_0 are dropped. In addition,

²For a stream or a group the dynamic failure rate is not counted until there are at least m packets transmitted.

there are several transmitted packets of S_1 which are unnecessarily scheduled since the service requirement of S_1 is $(2, 3)$. Most importantly, the example illustrates how the state information of group G_0 represents the *average* service received by its members, not the actual services received by the individual streams. The cause of the intra-group unbalance can be directly attributed to the differences in deadlines of S_0 and S_1 . Similarly, in group G_1 , because S_3 has a more lax deadline than S_2 , S_3 receives a better performance than S_2 as well.

In GAS, the group balancing algorithm is used to balance groups that belong to the same class. We consider two groups, G_0 and G_1 , to do the group balancing. G_0 contains n streams, $S_{01}, S_{02}, \dots, S_{0n}$ with deadlines $d_{01}, d_{02}, \dots, d_{0n}$, respectively; G_1 contains m streams, $S_{11}, S_{12}, \dots, S_{1m}$ with deadlines $d_{11}, d_{12}, \dots, d_{1m}$, respectively. GAS first increasingly sorts all streams belonging to G_0 and G_1 by their deadlines and produces d_1, d_2, \dots, d_{m+n} , where $d_i \leq d_j$ if $i < j$. Next, streams $S(d_1), \dots, S(d_{\lfloor \frac{m+n}{2} \rfloor})$ are allocated to G_0 ; streams $S(d_{\lfloor \frac{m+n}{2} \rfloor + 1}), \dots, S(d_{m+n})$ are allocated to G_1 . The deadlines of G_0 and G_1 can be computed as:

$$d(G_0) = (\sum_{i=1}^{\lfloor \frac{m+n}{2} \rfloor} d_i) / (\lfloor \frac{m+n}{2} \rfloor);$$

$$d(G_1) = (\sum_{i=\lfloor \frac{m+n}{2} \rfloor + 1}^{m+n} d_i) / (m + n - \lfloor \frac{m+n}{2} \rfloor)$$

Note that the packets for moved streams are not copied, only the queue routing information for those streams is changed. Example 2 shows how the DBP-based GAS scheduler works with the same scenario as Example 1.

Example 2: To isolate the group balancing effect, B (Burst Window Size) is equal to 1. In contrast to DCQM, the GAS model attempts to put the streams with similar deadlines into the same group through the group balancing algorithm when $S_0, S_1, S_2,$ and S_3 are added into the server: S_0 and S_2 are multiplexed into G_0 ; S_1 and S_3 are multiplexed into G_1 .

Figure 3 shows the resulting schedule from a DBP-based GAS scheme. Until $t = 19$, the number of dynamic failures in G_0 and G_1 are 0 and 2, respectively. The number of dynamic failures in $S_0, S_1, S_2,$ and S_3 are 0, 2, 0, and 0, respectively. Since the streams multiplexed into one group have the similar deadlines, the state information of a group reflects the state information of its streams more accurately. Since the packets waiting in the queue of a group to be transmitted have similar deadlines no matter which streams they belong to, every stream in one group has a similar opportunity to transmit packets which results in a finer QoS granularity.

In addition, the comparison of Example 1 to Example

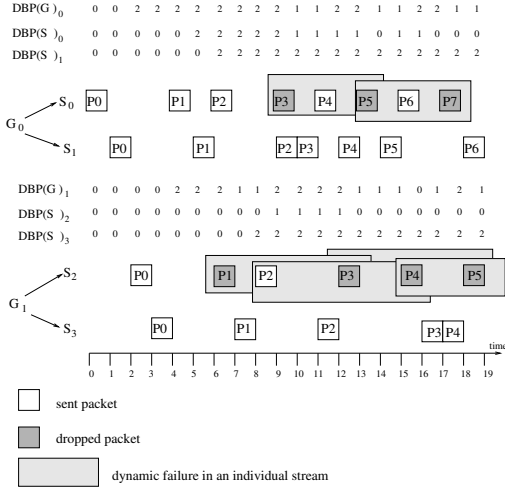


Fig. 2. Schedule of the packets in Example 1

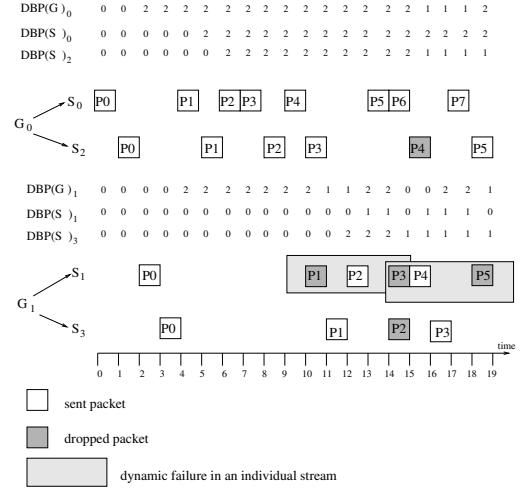


Fig. 3. Schedule of the packets in Example 2

2 also shows the inefficiency of period-based aggregation: in Example 1, streams with the same periods are aggregated into one group; in Example 2, streams with the same deadlines are aggregated into the same group. As deadline-based aggregation treats streams more fairly, the performance of Example 2, where the number of dynamic failures is 2, is better than the one in Example 1, where the number of dynamic failures is 6.

B. Stream Join

When a new stream, S , begins to send packets in a media server, the stream join operation happens. For the newly coming stream, S , the GAS scheme will select a group within the QoS class³ of S . If there are *open groups*⁴ existing within the class, the GAS scheduler finds the group, G_{open} , which has the closest average deadline to the new stream. Otherwise, a new group will be created for stream S . After S joins a group, to decrease the inter-group unbalance (group size) and the intra-group unbalance (deadline), the group balancing algorithm is invoked between the group holding S and the group with the most similar deadline to S in the same class.

C. Burst Sending

When several streams are multiplexed into the same group, the DBP/DWCS state of the group is the aggregation of the states of those streams. This aggregation introduces inaccurate evaluations of the actual received

³The mapping of streams into classes is beyond the scope of this paper.

⁴A group is an open group if the number of streams in this group is less than G . A group is a closed group if the number of streams that this group is holding equals G .

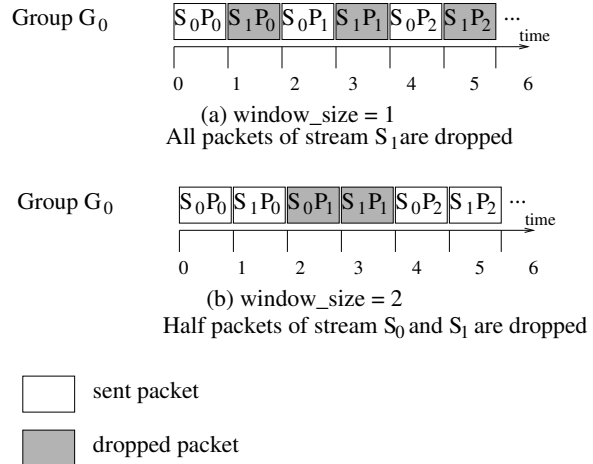


Fig. 4. The effect of burst sending

services by individual streams. Thus, even though selecting the “best” group at each possible scheduling point (at each packet) will minimize the perceived group dynamic failure rate, the aggregation masks the actual stream dynamic performance. Burst sending for B (Burst Window Size) packets can decrease this intra-group unbalance which is introduced by the distribution of packets of per-group streams in the queue of the group. Figure 4 shows the effect of burst sending when the QoS requirement is (1,2). In figure 4 (a), group G_0 satisfies the (1,2) service requirement. But the service is not distributed similarly between its two identical streams S_0 and S_1 : all of S_0 's packets are sent; all of S_1 's packets are dropped. Burst sending two packets from the sending group can correct

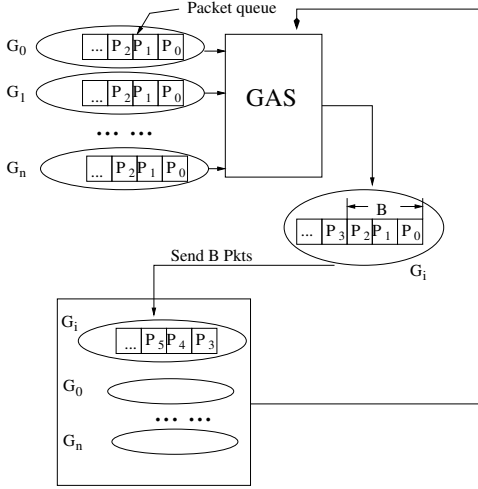


Fig. 5. The model of burst sending

this unbalanced service. In Figure 4 (b), though G_0 does not fully satisfy the (1,2) QoS requirement, both stream S_0 and S_1 satisfy the (1,2) QoS requirement.

Figure 5 details how the GAS scheduler works when $B = 3$. First, the GAS scheduler chooses one group to send packets called the *sending group*. Here, the sending group is G_i . Since the number of packets in the queue of G_i is greater than B , the scheduler sends the first 3 packets from the queue of G_i , regardless of state changes in other groups. Late packets are dropped from each group and state information is updated to reflect the new group-wise DBP/DWCS values. The scheduling process is repeatedly using the highest priority group. Example 3 shows how the DBP-based GAS scheduler works for the same scenario as Example 1 and Example 2 when $B = 4$.

Example 3: Since the GAS model attempts to put the streams with similar deadlines into one group by applying the group balancing algorithm, when S_0 , S_1 , S_2 , and S_3 are added into the server, S_0 and S_2 are multiplexed into G_0 ; S_1 and S_3 are multiplexed into G_1 . In this DBP-based GAS scheduler, B (Burst Window Size) is equal to 4. Thus, when a group is selected by the GAS scheduler as the sending group to transmit packets, at most 4 packets can be transmitted from this group before the GAS scheduler re-prioritizes the sending group. Packets are serviced by a FIFO policy from the queue of a group.

Figure 6 shows the schedule procedure. Until $t = 19$, the number of dynamic failures in G_0 and G_1 are 2 and 1, respectively. Because the GAS scheduler will not use group state information during the burst sending

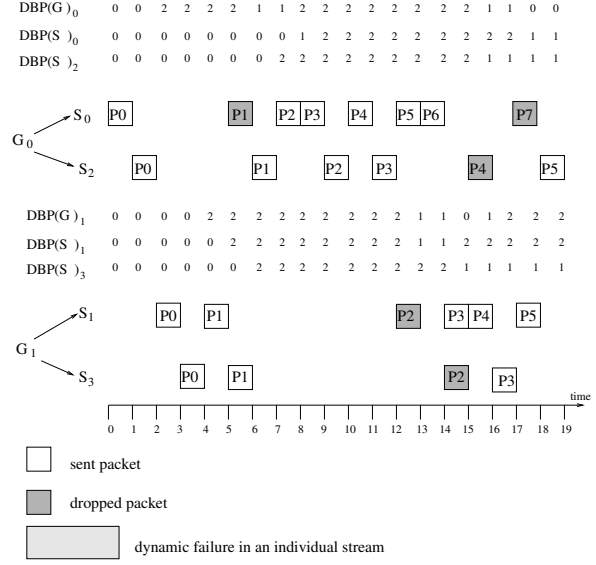


Fig. 6. Schedule of the packets in Example 3

procedure regardless of state information changes, the state information of each group is not consulted thoroughly for each scheduling decision. Thus, the perceived group-wise dynamic failure rate with burst sending is most likely higher than the one without burst sending. However, worse performance for groups does not mean the performances for streams are worse as well. In Example 3, the number of dynamic failures in S_0 , S_1 , S_2 , and S_3 are 0, 0, 0, and 0, respectively, which is a better performance than the one in Example 2 (where burst sending was not used). By burst sending multiple packets from the same group, the effect of aggregation is corrected further.

For a certain group size, G , there is no method to decide which burst window size can ensure the scheduler to achieve the lowest dynamic failure rate. It seems that when the burst window size is close to G , the low dynamic failure rate can be achieved.

D. Client Feedback

By using feedback coming from clients, the server can improve the performance (lower the average of dynamic failure rate received by individual streams/clients). A client always knows the quality of service (QoS) it receives from the server by computing the dynamic failure rate. Then, the client sends the feedback in terms of dynamic failure rate back to the server periodically or controlled by a threshold. The server can exploit the feedback information in two ways: periodic balance and aperiodic balance. However, we haven't considered the

cost to get the feedback, which means that we assume the cost to get and handle the feedback equals zero (the feedback information is sent back to the server without delay).

Periodic Balance: For every certain time (controlled by a timer), according to the feedback information (dynamic failure rates of individual streams) the scheduler chooses the group that holds the stream with the highest dynamic failure rate as the object to improve; then sends the packets from this group by burst sending; finally balances this group with the group whose deadline is close to the deadline of this group.

Aperiodic Balance: After the scheduler finishes the burst sending from a group, by checking the feedback information coming from clients, the scheduler knows if streams aggregated into this group having dynamic failures. If the answer is yes, the scheduler does the same job as periodic balance: burst sends packets from this group and balances this group with another group; otherwise, the scheduler does nothing.

IV. IMPLEMENTATION

Based on the DWCS packet scheduler in Linux kernel 2.4, we implemented the DWCS and GAS packet scheduler in Red Hat Fedora 2 with the kernel version being 2.6.5-1.358. Since the structure of the original DWCS implementation does not allow to add group balancing of GAS without significant changes to the design, in this paper, we only implemented the burst sending part of GAS. Furthermore, in order to provide users with friendly user-interface, we implemented an extension to Iproute2 [16] to offer same user-interfaces as other Linux packet schedulers. For the reason of space, we omit the detail description of Linux Packet scheduler [17] and the introduction to Iproute2.

V. SIMULATION STUDIES

We evaluated the performance of the proposed GAS scheme through simulation studies. In our simulation studies, we compared the DWCS-based DCQM scheme to the DWCS-based GAS scheme. The *dynamic failure rate* is used as the performance metric with the goal to decrease the dynamic failure rate as low as possible while maximizing scalability. In our simulations, each stream was characterized by the stream period (p_i), relative deadline (d_i), and stream duration. One millisecond (ms) was represented by one simulation clock tick. The streams for the simulations were generated as follows:

- the (m,k) value equals (20,33).
- The arrival of the streams follow Poisson distribution with a mean inter-arrival time 50ms. The

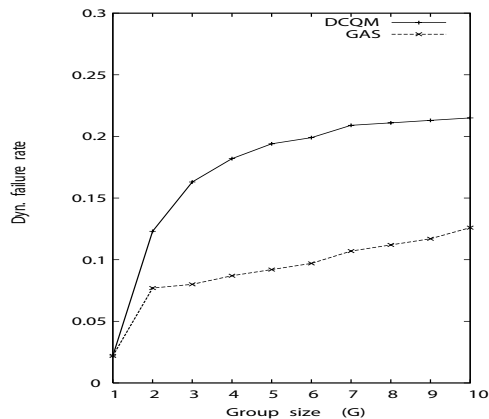


Fig. 7. GAS vs. DCQM

period and deadline of a stream are exponentially distributed with a mean of 70.

- Packets are assumed to be of fixed length.
- The server is assumed to have enough buffer space and hence dropping due to buffer overflow does not occur.

A. Effect of Group Balancing in GAS

Figure 7 shows the dynamic failure rate of the DCQM and GAS schedulers. Here, we do not use the client feedback to improve the GAS scheduler's performance, which we will discuss later. In GAS scheduler, the burst window size (B) equals to the group size (G). With an increase of the group size, in Figure 7, the dynamic failure rate increased in both the DCQM and the GAS models. The change in dynamic failure rate as the group size, G , increases can be explained by the variation in the number of groups (not shown). As G increases, the number of groups decreases, thus yielding a decrease in QoS granularity and hence an increase in the dynamic failure rate.

We also can see in Figure 7 that with an increase of scalability, the GAS model had a better QoS granularity than the DCQM model. The reason lies in the fact that GAS employs group balancing to decrease the intra-group unbalance.

B. Effect of The Burst Window Size in GAS

Figure 8 shows the dynamic failure rates of the DWCS-based GAS scheduler with different values of B (Burst Window Size), when G (Group Size) was equal to 6, 8, and 10. No client feedback is employed to improve the performance. We can see that burst sending effectively reduced the dynamic failure rate for the

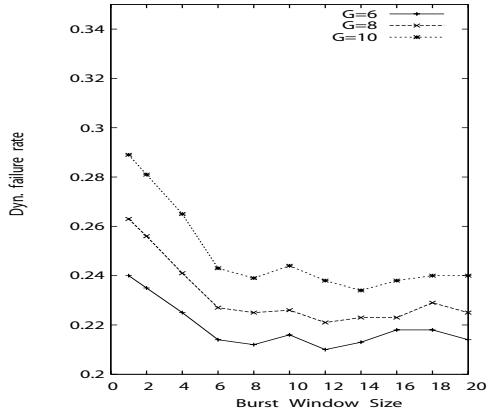


Fig. 8. Effect of burst window size

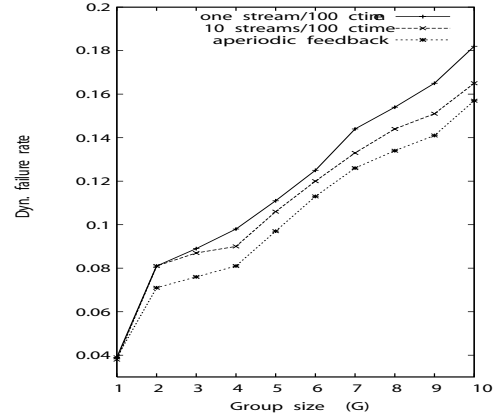


Fig. 9. Periodic and aperiodic balance

system. As explained in section III-C, selecting the group with the highest DWCS priority to send a single packet at each possible scheduling point can only minimize the perceived *group-wise* dynamic failure rate, not for the actual streams since group-wise state information cannot reflect stream-wise state situation accurately. Burst sending B packets consecutively from the sending group without considering the state information of other groups can correct the intra-group unbalance which is caused by the packet distribution of per-group streams and achieve a lower dynamic failure rate. Currently, our initial work has not yielded an formal method for calculating the optimal value for B . Based on our simulation results, it appears that setting B either equal to or close to G yields optimal results.

C. Effect of Periodic And Aperiodic Balance

Figure 9 shows the effects of periodic balance and aperiodic balance on the dynamic failure rate for the GAS scheduler under the condition of B is equal to G .

For the periodic balance, every certain time (100 ms), the scheduler chooses a number of streams which have the highest dynamic failure rate to improve. The less the number of streams chosen to improve, the worse the system performance. In other words, to get a better result, the scheduler must use more feedbacks and more time to deal with them.

Aperiodic balance achieves a better performance than periodic balance does. This is because that the scheduler uses the client feedbacks more frequently and more on time.

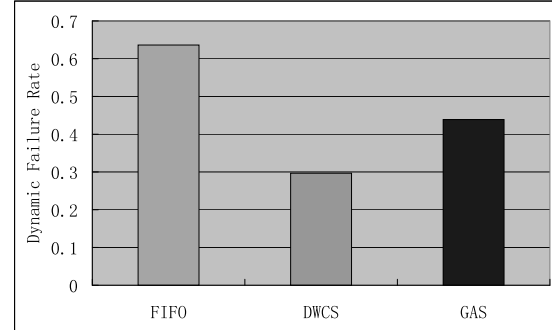


Fig. 10. Average Dynamic Failure Rate on FIFO, DWCS and GAS

VI. EXPERIMENTAL EVALUATION

For validating our simulation results in a real application environment, we conducted experiments with 2 Linux machines. One functions as a stream server that keeps sending packets out to another one, which functions as a client receiving steams. Our experiments were conducted based on our implementations of the DWCS scheduler and the GAS scheduler. We also implemented an udp stream server program and an udp stream client program for the experiments.

A. Experiment Methodology

Similar to our simulation studies, we compared the performance of FIFO, DWCS, DWCS-based GAS in terms of the dynamic failure rate. Furthermore, in order to validate the argument that GAS is more scalable than DWCS, we compared the performance of FIFO, DWCS, and DWCS-based GAS in terms of the single

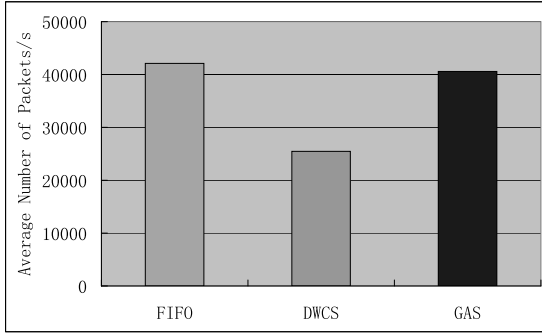


Fig. 11. Average Throughput on FIFO, DWCS and GAS (pkt size = 512 bytes)

stream throughput. Basically, we created 120 packet streams on the server. The server kept sending packets to its downstream client. We ran 120 client processes on the client machine to simulate 120 clients. In each packet there were a stream id and a sequence number for statistics. The client machine would log the stream id and sequence number respectively for each stream. The experiments were conducted multiple times with the resulting averages presented in the figures. Each experiment lasts for 5 minutes. The base parameters of DWCS were set as follows:

- Deadline: 2ms.
- Loss Numerator: 2.
- Loss Denominator: 10.

B. Experiments Results

1) *Average Dynamic Failure Rate on Different Schedulers:* Figure 10 and Figure 11 show the performance of FIFO, DWCS and GAS. To isolate the effect of stream aggregation, no burst sending was exploited in the GAS scheduler ($B = 1$). In Figure 10, on one extreme, DWCS provided the lowest dynamic failure rate as per-stream state information is maintained in DWCS. However, the low dynamic failure rate of DWCS was achieved at the cost of scalability as the throughput of the DWCS scheme was low (shown in Figure 11). On the other extreme, even though FIFO was scalable in terms of high throughput, its dynamic failure rate was high. The GAS scheme was proposed to balance the scalability and QoS granularity by aggregating multiple streams into one group. The experiment results followed the same trend as simulations. Comparing to DWCS, GAS compromised the stream-wise QoS in terms of dynamic failure rate (as shown in Figures 10) while increases the throughput.

2) *Effect of Burst Sending:* Figure 12 shows the effect of the burst window (B) when G (Group Size) was equal

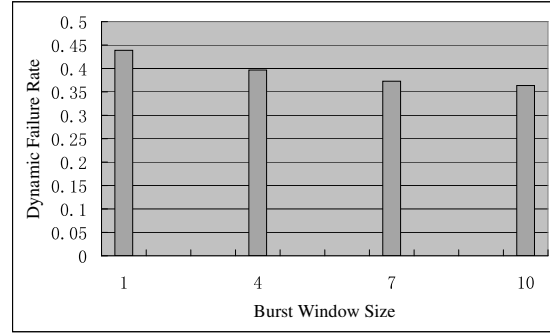


Fig. 12. Effect of burst window (B)

to 8. As the burst window increased, the dynamic failure rate for the system decreased which is similar to the simulation results shown in section V-B. Thus, by burst sending multiple packets successively from one group, the intra-group balance can be improved.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a granularity aware (m, k) scheduler (GAS) which consists of the algorithms for group balancing, stream join, burst sending, and feedback. By coupling the group balancing algorithm with the stream join algorithm, we successfully decreased the intra-group unbalance which is introduced by the significantly different deadlines of per-group streams. In addition, the burst sending algorithm was proposed to further decrease the intra-group unbalance due to the packet distribution of per-group streams. At last, exploiting feedback improved the performance further. Our simulation and experiment studies have shown that GAS can effectively control the tradeoff between QoS granularity and scalability.

Our future work include the implementation of group balancing in the GAS scheme and test the scheduler with real media stream data.

REFERENCES

- [1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, pp. 122–139, Jan. 1994.
- [2] D. Ferrari, "Client requirements for real-time communication services," *IEEE Communications Magazine*, vol. 28, no. 11, pp. 76–90, Nov. 1990.
- [3] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k) -firm guarantees," *IEEE Transactions on Computers*, vol. 44, no. 4, pp. 1443–1451, Dec. 1995.
- [4] A. Striegel and G. Manimaran, "Dynamic class-based queue management for scalable media servers," *Journal of Systems and Software*, vol. 66, no. 2, pp. 119–128, May 2003.

- [5] R. West, Y. Zhang, K. Schwan, and C. Poellabauer, "Dynamic window-constrained scheduling of real-time streams in media servers," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 744–759, Jun. 2004.
- [6] Y. H. P. Gupta and R. Guerin, "Supporting excess real-time traffic with active drop queue," Dept. of ESE, University of Pennsylvania, Tech. Rep., 2002.
- [7] K. Li, J. Walpole, D. McNamee, C. Pu, and D. Steere, "A rate-matching packet scheduler for real-rate applications," *Proceedings of Multimedia Computing and Networking 2001*, Jan. 2001.
- [8] M. Harchol-Balter, N. Bansal, and B. Schroeder, "Implementation of srpt scheduling in web servers," no. CMU-CS-00-170, October 2000.
- [9] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," *Proc. 19th IEEE Real-Time Systems Symposium*, pp. 123–132, 1998.
- [10] V. Jacobson, K. Nichols, and K. Poduri, "An expedited forwarding PHB group," *IETF RFC 2598*, Jun. 1999.
- [11] M. Hamdaoui and P. Ramanathan, "Evaluating dynamic failure probability for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1325–1337, Dec. 1997.
- [12] A. Striegel and G. Manimaran, "Best-effort scheduling of (m, k)-firm real-time streams in multihop networks," *Computer Communications*, vol. 23, no. 13, pp. 1292–1300, Jul. 2000.
- [13] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, Oct. 1990.
- [14] J. C. R. Bennett and H. Zhang, "WF²Q: Worst-case fair weighted fair queueing," *Proc. IEEE INFOCOM'96*, 1996.
- [15] Y. Zhang, R. West, and X. Qi, "A virtual deadline scheduler for window-constrained service guarantees," *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2004.
- [16] "Iproute2 utility suite howto," <http://www.policyrouting.org/iproute2-toc.html>.
- [17] "Traffic control howto," <http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/>.