

# Granularity Aware $(m, k)$ Queue Management for Real-time Media Servers

Yingxin Jiang and Aaron Striegel  
Department of Computer Science and Engineering  
University of Notre Dame  
Notre Dame, IN 46556 USA  
{yjiang3,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. While techniques such as aggregation of state information for scalability have been proposed in the literature such as with Differentiated Services, the per-stream effects of such aggregation are poorly understood. In this paper, we explore the effects of aggregated state information and propose a granularity aware  $(m, k)$  queue management (GAQM) which improves control over the tradeoff between scalability/granularity and QoS performance. Specifically, we identify the necessity of balancing aggregation groups according to critical characteristics such as relative deadlines. We present detailed examples of GAQM and evaluate our work through simulation studies.*

## 1 Introduction

With the proliferation of multimedia content on the Internet, a critical aspect of the user experience is the quality of service (QoS) delivered by the content infrastructure. Notably, the multimedia content possesses several common qualities in that data is often timely (must be delivered before in a certain amount of time) [1, 6] and loss tolerant (a loss may not produce a noticeable quality drop). The fundamentals of real-time scheduling offer important insight as QoS characteristics are easily mapped to their real-time counterparts.

As the system load increases, the scheduling mechanism becomes critical to balance the loss between applications as tolerated by their QoS requirements. For instance, an audio signal may miss vital portions of a talkspurt if a block of consecutive packets are dropped but may perform satisfactorily if losses are adequately spaced. A loss tolerant stream is said to experience *dynamic failure* when the number of lost packets over a window exceeds the maximum tolerable value [3]. In its simplest form, the goal of the scheduler is to minimize the dynamic failure or conversely, maximize the number of clients receiving acceptable quality of service.

In the literature, a wide variety of schemes have been proposed to address the problem of scheduling loss tolerant multimedia streams at a real-time server [3, 8–12]. While the work in [10] offered a light weight per-stream stateful mechanism for scheduling, the work in [9] focused on reducing the amount of state required through aggregation into groups. This posits an interesting question: *what is more scalable, reducing the weight of the state or reducing the amount of state required?*

In practice, the issue of scalability is one of satisfying the maximum client load and excess capacity is immaterial if quality is sacrificed. Hence for many real-time media servers, the usage of a light weight state mechanism such as DWCS [10] is sufficient. In contrast, when addressing the core of the network where the scale can be considerably higher (millions or billions of flows), the amount of state dominates the scalability issue. Despite the fact that techniques for reducing state (fixed number of classes, etc.) are well-known, the effects on per-stream quality are not well understood. Rather, such work on scalability has typically focused the average performance of the class or group.

The issue of state aggregation and improving per-stream quality provides the motivation for this paper. While the application of this work to the network with

its considerable intricacies is the end goal, the real-time server provides an excellent controlled environment from which to observe the effects of state aggregation. Furthermore, loss tolerant multimedia streams offer an ideal test case for exploring the impact of aggregated state information.

To that end, our paper investigates the effects of state granularity on a real-time media server. Specifically, we propose a new queue management scheme, Granularity Aware  $(m, k)$  Queue Management (GAQM), that offers significant improvement versus existing work and offers novel insights regarding the importance of group organization versus the underlying stream characteristics.

The rest of the paper is structured as follows. Section 2 details the related work to our paper. Next, Section 3 describes the GAQM model. Then, Section 4 presents simulation studies of our scheduler. Finally, in Section 5, we make several concluding remarks.

## 2 Related Work

### 2.1 Distance Based Priority (DBP)

In [3], the  $(m, k)$  model was proposed by Hamdaoui and Ramanathan 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. When the number of dropped packets over  $k$  exceeds  $k - m$ , a condition known as *dynamic failure* occurs. In order to encapsulate the  $(m, k)$  model with scheduling, the Distance Based Priority (DBP) scheme was proposed in [3]. In the DBP scheme, 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 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.

In [8], the EDBP algorithm was proposed to 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.

### 2.2 Dynamic Window-Constrained Scheduling (DWCS) & Virtual Deadline Scheduling (VDS)

In [10], the DWCS scheme was proposed by West et al. to deal with loss-tolerant streams expressed by the  $(x, y)$  model. In the  $(x, y)$  model, there are no more than  $x$  packets lost for every  $y$  consecutive packets. 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 dynamic window whereby the values of  $x$  and  $y$  are changed according to whether or not a packet of the stream is dropped. In [10], simulation results showed that DBP and DWCS had similar performance characteristics.

Virtual Deadline Scheduling (VDS) was proposed in [12]. 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, according to its deadline and the current window-constraint, VDS derives a virtual deadline, which may be some finite time after the corresponding real-time deadline. VDS performs similar scheduling decision as DWCS by using virtual deadlines.

Multi-hop Virtual Deadline Scheduling (MVDS) [11] is an extension of VDS, which deals with the end-to-end window constraints rather than focuses on a single real-time server. When a job arrives at a server, the server derives a local virtual deadline for it based on its local deadline at the previous hop and current window constraints.

### 2.3 Dynamic Class-Based Queue Management (DCQM)

Most scalable schemes (RMS [2], EDF, class-based scheduling [4, 5, 7]) ignore stream-wise information at the cost of dynamic failure performance. On the other hand, schemes maintaining per-stream state information, such as DBP and DWCS, suffer from scalability issues. To balance the scalability/granularity and QoS performance in a media server, Striegel and Manimaran proposed the DCQM (Dynamic Class-Based Queue Management) model in [9].

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

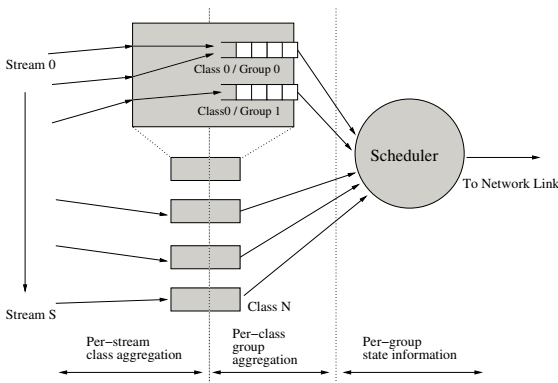


Figure 1. DCQM model proposed in [9]

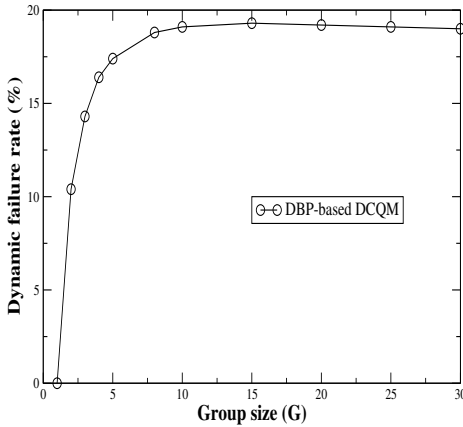


Figure 2. Dynamic failure rate of DCQM based scheduler

of streams that can be multiplexed. Figure 1 shows the DCQM model. 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. This occurs because the QoS received by the group does not represent the QoS received by the individual streams themselves. As a result, the more streams a group holds, the more the stream-wise state information is hidden, hence the higher dynamic failure rate the system receives.

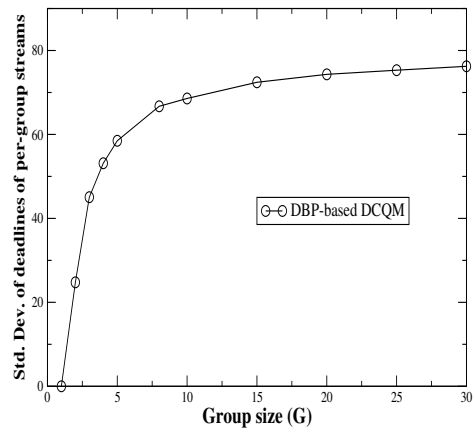


Figure 3. Average standard deviation of the deadlines of groups in the DCQM model

In Figure 2, which shows the effect of  $G$  (the number of streams a group can hold) on the dynamic failure rate for the DCQM scheduler, we can see that the DCQM scheduler offers minimal control on the tradeoff between scalability/granularity and QoS performance, even when  $G$  is small. For example, since the scalability for  $G = 8$  is worse than the scalability for  $G = 30$ , the dynamic failure rate for  $G = 8$  should be lower than the one for  $G = 30$ . However, in the DCQM model, they receive almost the same QoS performance. One of the reasons for this ineffective tradeoff control between scalability/granularity and QoS performance lies in the fact that streams are multiplexed into groups blindly, without consideration for their characteristics.

Figure 3 shows the average standard deviation of the deadlines of groups with the increase of  $G$  in the DCQM model: after  $G$  is greater than 8, the standard deviation of the deadlines of per-group streams is quite high. If the streams multiplexed into the same group have significantly different deadlines, the streams with lax deadlines will receive better performance than the streams with tight deadlines when FIFO is used inside of a group. After dropping the tight-deadline packets, the packets with lax deadlines will stay at the head of the queue. Thus, each time the scheduler chooses the packet from the queue of the group to transmit, the packets with lax deadlines will have a higher probability of being at the head of the queue. Since the state information of the streams is mapped into a single piece of group state information, the group cannot differentiate which kind of performance is received by an individual stream belong-

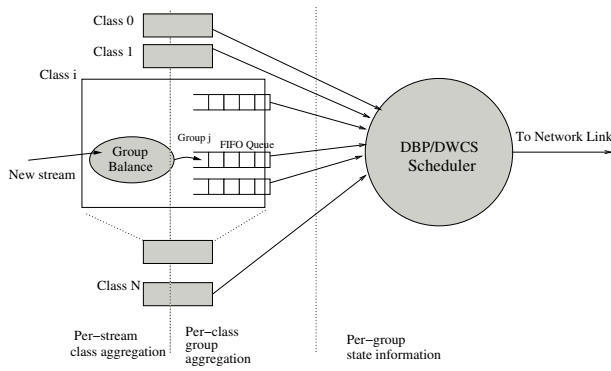


Figure 4. GAQM model

ing to it.

### 3 GAQM (Granularity Aware $(m, k)$ Queue Management) Overview

Figure 4 shows the model of GAQM (Granularity Aware  $(m, k)$  Queue Management). To capture the tradeoff between scalability/granularity and QoS performance more accurately, GAQM incorporates the following aspects:

- **Group balancing:** The group balancing algorithm is invoked on a stream join to correct any unbalanced stream distributions. Unbalanced groups include both cardinality balancing (group size) and characteristic balancing (deadlines).
- **Deadline Tolerance ( $DT$ ):** Deadline Tolerance is used to restrict deviation in deadlines for streams multiplexed into the same group. By restricting the deviation in relative deadlines, a quasi-EDF ordering is achieved despite using a FIFO queue.

#### 3.1 Scheduling

Scheduling in the GAQM is provided similar to the original DCQM model [9] (see Figure 1). 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. Ties in priority are resolved using the deadline.

While GAQM conceptually operates similar to DCQM, the key differences occur in how groups are se-

lected for individual streams. Unlike DCQM, GAQM does not assign streams to groups simply based on similar loss tolerances. Rather, GAQM applies sophisticated stream join and group balance techniques that significantly reduce the penalty associated with group-wise state aggregation.

#### 3.2 Stream Join

When a new stream,  $S$ , begins to send packets in a media server, GAQM will select a group for the stream within the QoS class of  $S$ . If there are *open groups*<sup>1</sup> existing within the class, the GAQM scheduler finds the group,  $G_{open}$ , which has the closest average deadline to the new stream. The new stream,  $S$ , is allocated to  $G_{open}$ , if:

$$|d(G_{open}) - d(S)| \leq DT$$

where  $d(G_{open})$  is the average deadline of group  $G_{open}$ ,  $d(S)$  is the deadline of stream  $S$ , and  $DT$  is defined as the *deadline tolerance*. 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. The number of groups depends on the number of streams,  $G$  (group size), and  $DT$  (deadline tolerance). Figure 5 details the algorithm for stream joining.

For each group in GAQM, a FIFO queue is employed to minimize queue insertion cost. While a more intelligent queue order scheme such as EDF could offer better performance, the insertion cost incurs significant overhead. Notably, schemes such as DBP and DWCS can employ a FIFO scheme without penalty as queues are maintained on a per-stream basis, thus implying a de facto order provided that the relative deadline is a consistent stream characteristic.

To balance the effect of the FIFO queue, the *deadline tolerance* ( $DT$ ) is used to control the deviation of deadlines for streams that share the same group and hence share the same queue. If  $DT$  is too large, stream deadlines can vary considerably, resulting in the potential for significant order inaccuracies, i.e. packets with a later deadline are served first. The net effect as observed with DCQM is a significant increase in dynamic failure rate as additional streams are multiplexed into the group.

<sup>1</sup>A 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  $G$ .

```

Stream Join (Stream S)
begin
  ClassGroups: set of groups whose class is Class(S)
  ClosestGroup: a group having less than G members
    and the closest deadline to S in ClassGroup
  SimilarGroup: a group can do group balancing with
    Group(S)
  DT: deadline tolerance

  if ( (ClosestGroup == NULL) or
    (diff( deadline(ClosestGroup), deadline(S)) > DT) )
    Create NewGroup
    Add S to NewGroup
    BalanceGroup(NewGroup, SimilarGroup)
  else
    Add S to ClosestGroup
    BalanceGroup(ClosestGroup, SimilarGroup)
end

```

**Figure 5. GAQM stream join algorithm**

By limiting  $DT$ , a quasi-EDF ordering is imposed on the queue such that arrival time ordering of packets will have a strong probability of implying deadline ordering. Stated formally, given two packets  $P_x$  and  $P_y$  belonging to a stream or streams that share the same group and given that  $P_y$  arrives after  $P_x$ , the following relationship should hold:

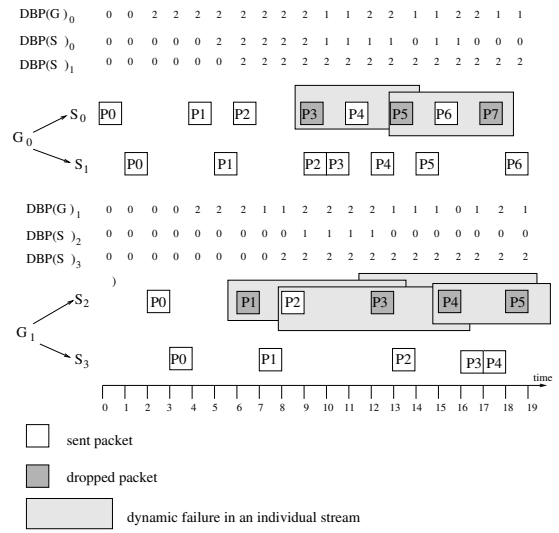
$$\lim_{DT \rightarrow 0} Prob(d(P_x) < d(P_y)) = 1$$

where  $d(P_x)$  is the deadline of packet  $P_x$  and  $d(P_y)$  is the deadline of packet  $P_y$ . In short, as  $DT$  is tightened, the ordering of the queue will approach that of an EDF scheme. An appropriate  $DT$  value will ensure that if a group is selected for scheduling, the packet at the head of the queue will be the packet with the lowest deadline.

However, while a low  $DT$  value will ensure an implicit EDF ordering without the EDF insertion cost, a value of  $DT$  that is too low will prevent stream aggregation from occurring, thus defeating the entire purpose of stream aggregation. From our simulation studies, we recommend a value of the standard deviation for  $DT$ . The derivation of the optimal value for  $DT$  is a topic for future research.

### 3.3 Group Balancing

As explained in section 3.2, if the streams multiplexed into one group have significantly different deadlines, the packets in the group-wise FIFO queue have an increased chance of being out of deadline order. The



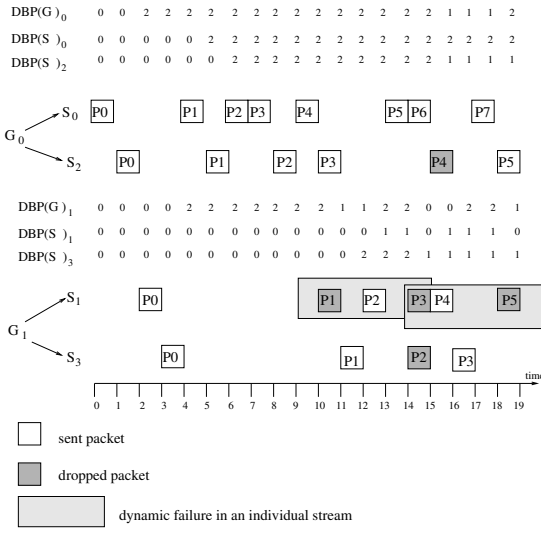
**Figure 6. Schedule for Example 1**

following shows how a DBP-based DCQM scheduler works when the streams with different deadlines are multiplexed into one group.

**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 6 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 0<sup>2</sup>. In group  $G_0$ , since  $S_1$  has a more lax deadline than  $S_0$ , more packets belonging  $S_0$  are dropped. In addition, 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.

<sup>2</sup>For a stream or a group, the dynamic failure rate is not counted until there are at least  $m$  packets transmitted.



**Figure 7. Schedule for Example 2**

In GAQM, 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.

GAQM 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:

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

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

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 GAQM scheduler works with the same scenario as Example 1.

**Example 2:** In contrast to DCQM, the GAQM 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 7 shows the resulting schedule from a DBP-based GAQM 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 similar deadlines, the state information of a group reflects the state information of its streams more accurately. Because 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 the better QoS performance.

## 4 Simulation Studies

We evaluated the performance of the proposed GAQM scheme through simulation studies. In our simulation studies, we compared the DBP-based DCQM scheme to the DBP-based GAQM 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, all streams in the same group had the same  $(m, k)$  value which was set as the group's value for  $(m, k)$  as well<sup>3</sup>. One millisecond (ms) was represented by one simulation clock tick. The streams for the simulations were generated as follows:

- The  $(m, k)$  value of each stream equaled (4,6) or (7,10) with the ratio of 1:1.
- Streams were Constant Bit Rate (CBR). Each stream was characterized by the stream period ( $p_i$ ), relative deadline ( $d_i$ ), and stream duration. The arrival of the streams followed Poisson distribution with a mean inter-arrival time of 110.
- For an individual stream, both the period and the relative deadline of generated packets were exponentially distributed with a mean of 120. The duration of a stream followed exponential distribution with a mean of 11,000.
- Packets were assumed to be of fixed length.
- The server was assumed to have sufficient buffer space such that dropping due to buffer overflow does not occur.

### 4.1 Effect of Group Balancing in GAQM

Figure 8 shows the dynamic failure rate of the DBP-based DCQM scheduler and the DBP-based GAQM

<sup>3</sup>Since we mainly focus on improving the server's performance when  $G$  has different values, for this paper, we assume that all streams belong to one class.

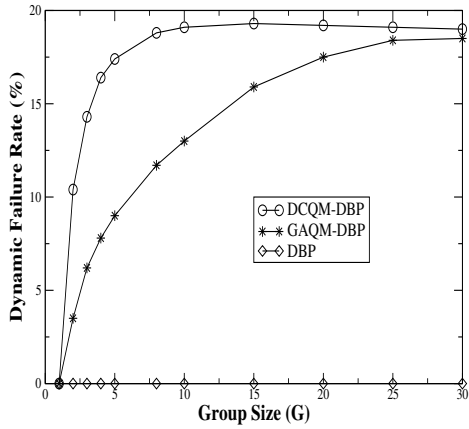


Figure 8. Dynamic failure rate for the DBP-based DCQM scheduler and the DBP-based GAQM scheduler

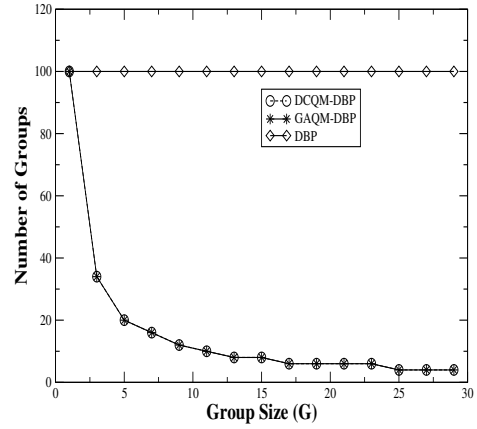


Figure 9. Effect of group size on number of groups

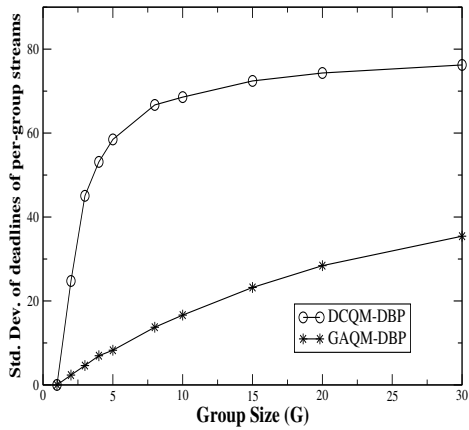


Figure 10. Average standard deviation of deadlines of groups in the DBP-based DCQM scheduler and the DBP-based GAQM scheduler

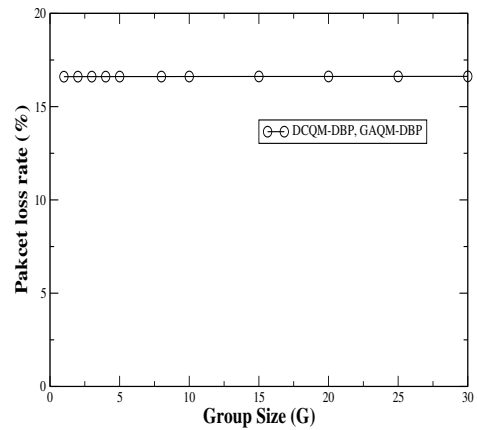
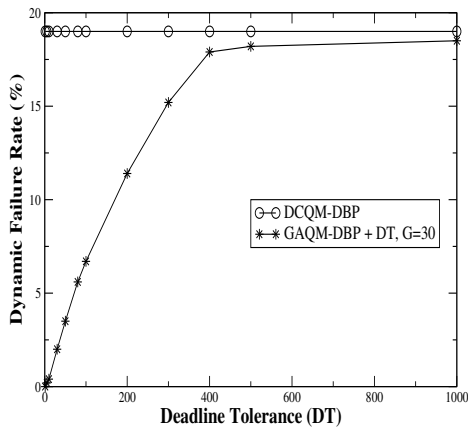
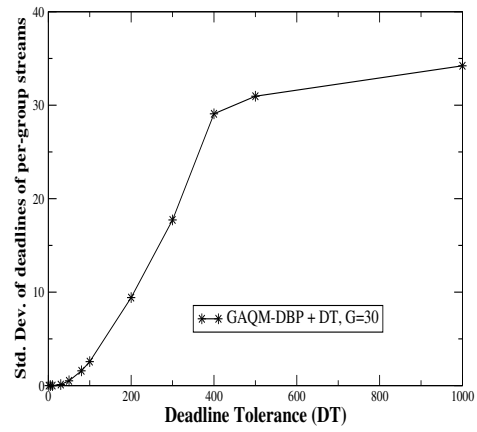


Figure 11. Packet loss rate in the DBP-based DCQM scheduler and the DBP-based GAQM scheduler



**Figure 12. Effect of deadline tolerance on dynamic failure rate**



**Figure 13. Effect of deadline tolerance on standard deviation of deadlines for per-group streams**

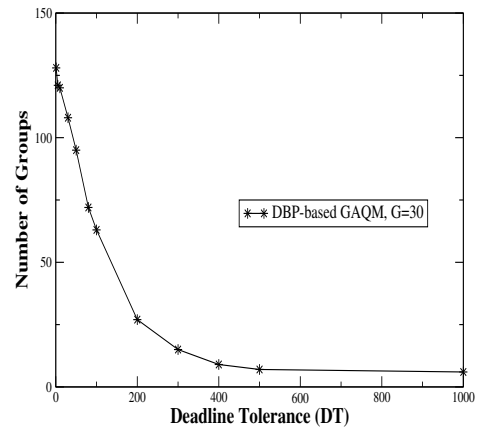
scheduler. To isolate the effect of group balancing, in this GAQM scheduler,  $DT = \infty$  (Deadline Tolerance). With an increase of the group size in Figure 8, the dynamic failure rate increased in both the DCQM and the GAQM models. The change in dynamic failure rate as the group size,  $G$ , increases can be explained by the variation in the number of groups as shown in Figure 9. As  $G$  increases, the number of groups decreases, thus yielding a decrease in QoS granularity and hence an increase in the dynamic failure rate. When  $G$  equals 1, both DCQM and GAQM performs as a DBP scheduler since the history for a group is the same as the history for a stream.

We also can see in Figure 8 that with an increase of  $G$ , the GAQM model had a better QoS performance than the DCQM model. The reason lies in the fact that GAQM uses group balancing to decrease the intra-group imbalance. Figure 10 shows the average standard deviation of deadlines of groups. As explained in section 3.2, if the deadlines of per-group streams vary significantly, the QoS performance decreases.

Though the different values of the group size,  $G$ , caused different dynamic failure rates, the packet loss rate of the system was unchanged (shown in Figure 11), since the system load was constant, and the dispatching ability of the server was constant as well.

#### 4.2 Effect of Deadline Tolerance Parameter

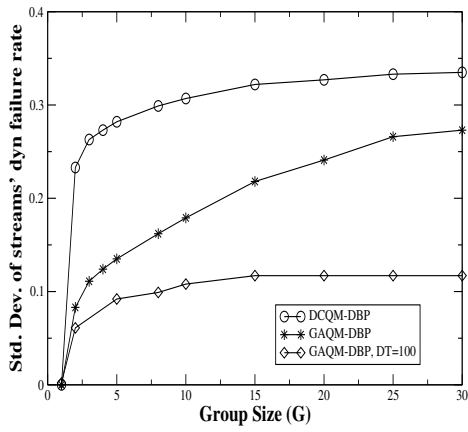
Figure 12, 13, and 14 show the simulation results of the DBP-based GAQM scheduler with different values for the deadline tolerance parameter,  $DT$ , when  $G$



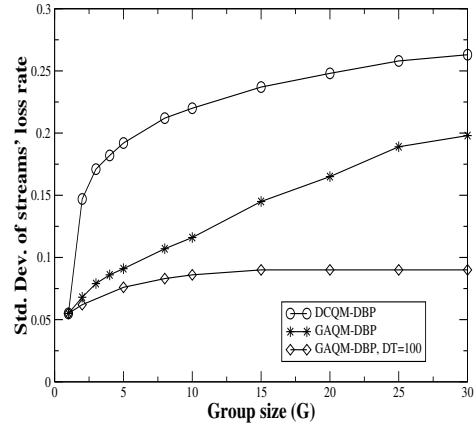
**Figure 14. Effect of deadline tolerance on number of groups**

was equal to 30. The deadline tolerance parameter is used to control the deviation of deadlines for per-group streams. As shown in Figure 13, with an increase of the value of the deadline tolerance parameter, streams with highly different deadlines could be multiplexed into the same group. Thus the probability of packets being out of deadline order in a group's FIFO queue increased, which caused the dynamic failure rate increased, see Figure 12.

The change in the dynamic failure rate as the value of the deadline tolerance increased can also be explained by the variation in the number of groups as shown in Figure 14. As the value of the deadline tolerance parameter increased, the number of groups decreased, thus



**Figure 15. The standard deviation of streams' dynamic failure rate in DCQM and GAQM**



**Figure 16. The standard deviation of streams' loss rate in DCQM and GAQM**

yielding a decrease in QoS granularity and hence an increase in the dynamic failure rate.

### 4.3 Services Received by Individual Streams

Figure 15 shows the standard deviation of streams' dynamic failure rates in the DBP-based DCQM model and the DBP-based GAQM model. Figure 16 shows the standard deviation of streams' loss rates in the DBP-based DCQM model and the DBP-based GAQM model. This GAQM scheduler had two settings for the deadline tolerance parameter,  $DT = \infty$  and  $DT = 100$ . From Figure 15 and Figure 16, we can see that no matter whether or not  $DT$  had an effect on GAQM, the GAQM scheduler treated individual streams in the same class more similarly. When the deadline tolerance parameter was included with the GAQM scheduler ( $DT = 100$ ), with the increase of the group size, the standard deviation for both streams' dynamic failure rates and streams' loss rates were stable. Under such condition,  $DT$  restricted the deviation of deadlines for per-group streams such that streams with significantly different deadlines cannot be multiplexed into the same group, which decreased the effect of group size ( $G$ ).

### 4.4 Effect of System Load

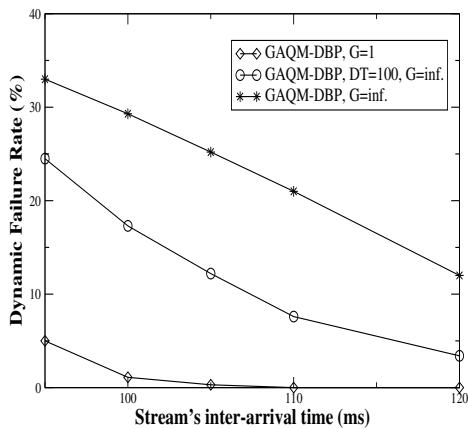
Figure 17 shows the system's dynamic failure rate when the mean of inter-arrival time for streams is varied. Figure 18 shows the system's dynamic failure rate when the mean of stream's period is varied. By changing stream's inter-arrival time or period, the system can

have different loads: the shorter the inter-arrival time or the period, the heavier the load of the system. Both Figure 17 and Figure 18 show that with the increase of the system's load, the level of the quality of service kept decreasing. The reason lies in the fact that with the increase of system load, the scheduling mechanism becomes critical to balance the loss between streams.

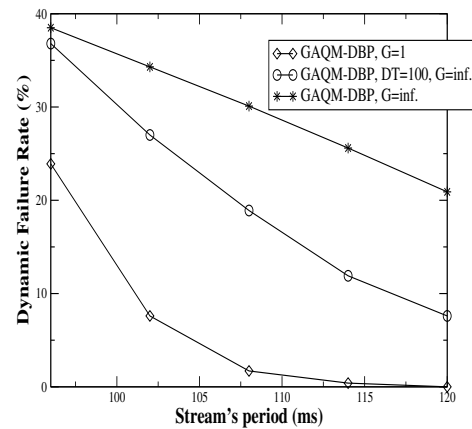
For the GAQM scheme where the deadline tolerance parameter had no effect ( $DT = \infty$ ), any streams with the same  $(m, k)$  QoS requirement could be multiplexed into one group such that packets were out of deadline order in the queue. Thus, when packets were serviced in FIFO order, the schedule utility was low, which caused high dynamic failure rates. When the deadline tolerance parameter was included in the GAQM scheme ( $DT = 100$ ), the deviation of deadlines for per-group streams was restricted, which offered a quasi-EDF order to a degree for FIFO queues and decreased the dynamic failure rates.

## 5 Conclusions

In this paper, we proposed a granularity aware  $(m, k)$  queue management (GAQM) which consists of the algorithms for group balancing and stream join. 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. Our simulation studies have shown that GAQM can more effectively control the tradeoff between scalability/granularity and QoS performance.



**Figure 17. Effect of inter-arrival time for streams on dynamic failure rate**



**Figure 18. Effect of period for streams on dynamic failure rate**

Our future work includes analytically evaluating the tradeoff between scalability/granularity and QoS performance, improving GAQM further, implementing GAQM in Linux, and applying GAQM to Differentiated Services.

## 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*, 82:122–139, Jan. 1994.
- [2] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. *Proc. 19th IEEE Real-Time Systems Symposium*, pages 123–132, 1998.
- [3] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm guarantees. *IEEE Transactions on Computers*, 44(4):1443–1451, Dec. 1995.
- [4] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB group. *IETF RFC 2598*, June 1999.
- [5] J. Mao, W. M. Moh, and B. Wei. PQWRR scheduling algorithm in supporting of DiffServ. *Proc. ICC 2001*, 3:679–684, 2001.
- [6] J. M. Peha and F. A. Tobagi. A cost-based scheduling algorithm to support Integrated Services. *Proc. IEEE INFOCOMM'91*, pages 741–753, 1991.
- [7] M. Song and M. Alam. Two scheduling algorithms for input-queued switches guaranteeing voice QoS. *Proc. IEEE GLOBECOM 2001*, 2001.
- [8] A. Striegel and G. Manimaran. Best-effort scheduling of (m, k)-firm real-time streams in multihop networks. *Computer Communications*, 23(13):1292–1300, July 2000.
- [9] A. Striegel and G. Manimaran. Dynamic class-based queue management for scalable media servers. *Journal of Systems and Software*, 66(2):119–128, May 2003.
- [10] R. West, Y. Zhang, K. Schwan, and C. Poellabauer. Dynamic window-constrained scheduling of real-time streams in media servers. *IEEE Transactions on Computers*, 53(6):744–759, June 2004.
- [11] Y. Zhang and R. West. End-to-end window-constrained scheduling for real-time communication. *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'04)*, Aug. 2004.
- [12] 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.