

Dynamic Class-Based Queue Management for Scalable Media Servers

A. Striegel and G. Manimaran

Department of Electrical and Computer Engineering

Iowa State University, USA

adstrieg@iastate.edu gmani@iastate.edu

April 13, 2001

Abstract

Real-time media servers are becoming increasingly important due to the rapid transition of the Internet from text and graphics based applications to multimedia-driven environments. In order to meet these ever increasing demands, real-time media servers are responsible for supporting a large number of clients with a heterogeneous mix of Quality of Service (QoS) requirements. In this paper, we propose a dynamic class-based queue management scheme that effectively captures the tradeoff between scalability and QoS granularity in a media server. We examine the adaptiveness of the scheme and its integration with the existing schedulers. Finally, we evaluate the effectiveness of the proposed scheme through extensive simulation studies.

1 Introduction

Real-time media servers are becoming increasingly important as the Internet moves from supporting text and graphic driven applications to supporting 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 hundreds, if not thousands of clients, with a wide range of QoS requirements. The QoS requirements of the clients are usually specified in terms of end-to-end delay, jitter, and loss [1, 2]. To meet the end-to-end QoS requirements of the clients, both the server and the network should provide QoS guarantees. At the network level, QoS support is either guaranteed by establishing real-time channel [3] between the server and the client or attempted in a best-effort manner [4]. Similarly, at the server level, QoS support is either guaranteed [5, 6] or attempted in a best-effort manner [7].

In this paper, we assume a particular type of real-time traffic (known as a *real-time stream*) from the server to a client, which is defined as a sequence of related packets that are transmitted at a regular interval (period) with certain delay constraints (deadline) [4]. From the perspective of the 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 cannot be used or less useful. In

the case of late packets that cannot be used, transmission of these packets only wastes bandwidth unnecessarily. For late packets that have a reduced usefulness, the perceived QoS of the client is reduced by the difference in usefulness between an on-time packet and the tardy packet. Thus, a late packet can be viewed in the same category as a lost packet since late packets will also reduce the QoS perceived by the client. In this paper, *loss-rate* [2, 7] can be defined as the fraction of packets that are discarded (dropped) or transmitted after their deadline.

Although loss-rate can be used as an overall QoS metric, a more applicable QoS metric can be used for streams that can tolerate occasional packet losses. Several streams in the media server may be able to tolerate a loss of a fraction of their packets with little or no degradation of the QoS perceived by the client. Thus, for loss-tolerant streams, loss-rate is not necessarily a meaningful measure of QoS. Instead, loss-rate is replaced by a different QoS requirement that defines when losses are allowed to occur. In order to satisfy this QoS requirement, the loss-rate can be monitored over a finite range, or window [7], of consecutive packets. For example, if a few consecutive audio packets miss their deadline, a vital portion of the talkspurt may be missing and the quality of the reconstructed audio signal may not be satisfactory. However, if the lost packets are adequately spaced, then interpolation techniques can be used to satisfactorily reconstruct the signal [8]. When the number of packet loss over a window packets exceeds the maximum tolerable value, a condition known as *dynamic failure* [4] occurs. In this paper, we evaluate different scheduling schemes for loss-tolerant streams on the basis of *dynamic failure rate*, which is defined as the fraction of packets that caused dynamic failure. In short, dynamic failure rate measures the fraction of packets which negatively impact the perceived QoS of the client.

In summary, a media server must be able to support diverse performance objectives. The media server must be scalable to service a large number of clients but at the same time should meet the individual QoS requirements of those clients. Several schedulers have been proposed to address these issues that are discussed in the next section.

2 Related Work

Scheduling schemes such as Earliest Deadline First (EDF) [9] and Weighted Fair Queuing (WFQ) [10] are not effective for loss-tolerant streams because they do not exploit the ability of such streams to tolerate occasional packet losses. Two schemes, Distance Based Priority (DBP) [4] and Dynamic Window-Constrained Scheduling (DWCS) [7], were proposed to exploit loss-tolerance by allocating bandwidth according to loss-tolerances and delay constraints. These schemes are described below in detail.

2.1 (m, k) model and Distance Based Priority (DBP)

In [4], 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 have to meet their deadlines. In order to encapsulate the (m, k) model with scheduling, the Distance Based Priority (DBP) scheme was proposed in [4]. In the DBP scheme, for each stream, a state is maintained. This

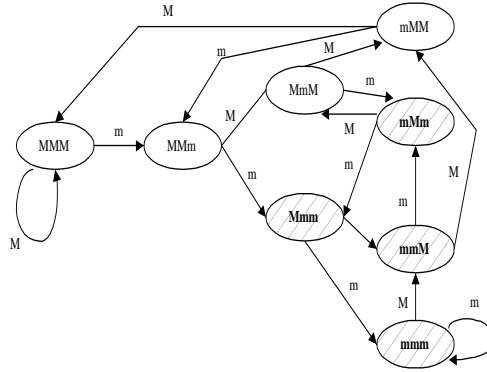


Figure 1: (m,k) state machine for a stream with parameters $(2,3)$

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: DWCS precedence among pairs of packets

state represents the history of the last k packets transmitted and a DBP value is associated with each state. 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 the number of meets is less than m . 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. Figure 1 shows the state diagram for a stream with a $(2,3)$ -firm guarantee wherein M and m are used to represent meeting a deadline and missing a deadline, respectively. Each state is represented by a three-letter string. For example, MMm denotes the state where the most recent packet missed its deadline and the two previous packets met their deadlines. The edges represent the possible state transitions. Starting from a state, the stream makes a transition to one of two states, depending on whether its next packet meets (denoted by M) or misses (denoted by m) its deadline. For example, if a stream is in state MMm (DBP value of 1) and its next packet meets the deadline, then the stream transits to state MmM . In Figure 1, the failing states are mMM , Mmm , mmM , and mmm .

2.2 (x, y) model and Dynamic Window-Constrained Scheduling (DWCS)

In [7], the (x, y) model was proposed for capturing the loss-tolerance of a stream such that x is the maximum acceptable loss for a given window of y consecutive packets. The (x, y) model can be related to the (m, k) model where $(x, y) = (k - m, k)$. DWCS maintains per-stream state information like DBP but the state information differs significantly from DBP. Whereas DBP uses the notion of state transitions and k -bit history tuples to capture the relative priority of streams, DWCS uses the notion of a *dynamic window* in which x and y are allowed to change. In DWCS, each time a packet in stream i is transmitted or dropped, the per-stream state information (x_i, y_i) is adjusted accordingly.

The rules for ordering pairs of packets from different streams for transmission are shown in Table 1. All packets belonging to the same stream are stored in the same queue and stored in their arrival order. In a queue, for given two packets having the same non-zero loss-tolerance, they are ordered based on EDF. If two packets have the same non-zero loss tolerance and deadline, they are ordered according to the lowest loss-numerator x_i , where $\frac{x_i}{y_i}$ 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 the tighter loss constraints, since fewer consecutive packet losses can be tolerated. If two packets have zero loss-tolerance with zero loss-denominator, they are ordered based on EDF, otherwise they are ordered based on highest loss-denominator first.

The rules for DWCS are given below:

- Loss-tolerance adjustment for a stream i that has a packet transmitted before its deadline

$$\text{if } (y'_i > x'_i) \text{ then } y'_i = y'_i - 1;$$

$$\text{if } (x'_i = y'_i = 0) \text{ then } x'_i = x_i; y'_i = y_i;$$

- Loss-tolerance adjustment for a stream i whose packet misses its deadline

$$\text{if } (x'_i > 0) \text{ then}$$

$$x'_i = x'_i - 1; y'_i = y'_i - 1;$$

$$\text{if } (x'_i = y'_i = 0) \text{ then } x'_i = x_i; y'_i = y_i;$$

$$\text{else if } (x'_i = 0) \text{ then } x'_i = x_i; y'_i = y_i;$$

In addition to providing scheduling for loss-tolerant streams, DWCS can also perform static priority and EDF scheduling to support both delay constrained and non-delay constrained traffic. However, although both DWCS and DBP can provide fine QoS granularity, the per-stream state information does not scale well with the number streams, which forms the motivation for our paper.

2.3 Motivation and Objectives

To provide diverse QoS requirements to a large number of clients, a media server must be scalable and must provide appropriate QoS granularity. In the context of media servers, *scalability* is informally defined as the ability to handle

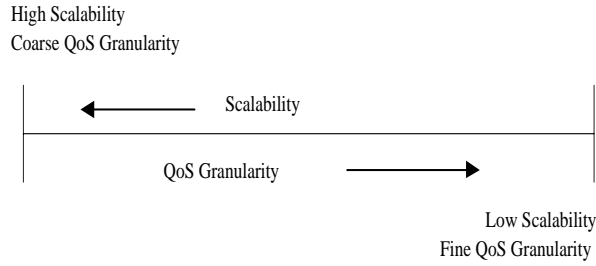


Figure 2: Tradeoff between scalability and QoS granularity

a large number of streams, whereas *QoS granularity* is defined as the variation in QoS experienced by the streams serviced by the media server. However, as Figure 2 shows, a tradeoff exists between scalability and QoS granularity.

On one extreme, a server may be extremely scalable. Examples of this include EDF and class-based scheduling [11]. These schemes either require no or a fixed amount of state information which depends on the number of queues maintained by them. However, the additional scalability is gained at the cost of QoS granularity. As streams are multiplexed onto queues (classes), the granularity of the state information is reduced. Thus, although a given queue may meet its QoS requirements, the individual streams multiplexed onto the queue may not meet their individual QoS requirements. This occurs because the QoS received by the queue represent the average QoS received by the streams, not the QoS received by the individual streams themselves. As a result of this, the dynamic failure rate increases with decreasing number of queues due to the reduced QoS granularity.

In contrast, schemes such as DBP and DWCS emphasize QoS granularity over scalability. State information is maintained on a per-stream basis, thus offering the best dynamic failure rates since there is a one-to-one mapping of streams to state information maintained. However, this fine grain QoS comes at the cost of scalability. Although some techniques can be employed in the implementation of DWCS to increase scalability [7], these techniques do not solve the scalability problem due to per-stream state information.

Therefore, the tradeoff of scalability versus QoS granularity provides the motivation for our paper. In our paper, a dynamic class-based queue management scheme (DCQM) is proposed that balances scalability and QoS granularity in a media server. This scheme can be used with the existing schedulers to provide a highly adaptive and scalable media server.

The objectives of our paper are as follows:

- Show that DCQM balances scalability and QoS granularity.
- Show that DCQM does not require *a priori* knowledge of streams and can respond to dynamic creation (stream join) and termination (stream leave) of streams.
- Show that DCQM can adapt appropriately to server dynamics by dynamic group sizing.
- Show that DCQM can be varied between the extremes of high scalability/coarse QoS granularity and low scal-

ability/fine QoS granularity through simulation studies.

The rest of the paper is structured as follows. Section 3 describes the DCQM model and basis for our scheduler. Then, Section 4 details the experimental studies of our scheduler. Finally, in Section 6, we make some concluding remarks.

3 Dynamic Class-Based Queue Management (DCQM)

In this section, we introduce our Dynamic Class-Based Queue Management (DCQM) model and then describe the algorithms for creation and termination of streams under our model. Finally, we describe how the DBP and DWCS schedulers can be used together with our model.

3.1 DCQM Model

In order to achieve scalability in a media server, the issues which reduce scalability must first be examined. In the DBP and DWCS schedulers, state information was maintained on a per-stream basis. This per-stream state information is not scalable as the number of streams increases. Therefore, we propose an abstraction similar to one used in the IETF Differentiated Services model [11], such that streams are aggregated into a fixed number of classes. Each class has an associated queue and consists of streams with similar loss tolerances that are aggregated together under a single class state. Thus, as the number of streams increases, the solution is scalable since the number of classes is fixed.

Although using per-class state information improves scalability, it also introduces a QoS granularity tradeoff. Prioritization is done on a per-class basis rather than on a per-stream basis reducing the fine grain QoS delivered by a per-stream solution. Thus, although the class itself may receive the appropriate average loss level, individual streams within that class may violate their individual loss-tolerances. In the context of media servers, where the amount of loss that a stream experiences can be just as critical as the end-to-end delay, a pure class-based scheme cannot deliver the per-stream loss tolerances required.

3.2 DCQM Groups

Therefore, we propose a scheme to allow flexibility between the two extremes of per-stream state information and per-class state information. In order to accomplish this, the notion of a group must first be defined. A *group* is defined to be a set of streams, S , with similar loss characteristics under the constraint of $|L| < G$, where $|L|$ is the cardinality of set S and G is the maximum group size. State information is maintained on a per-group basis. Thus, by varying G , it is possible to achieve per-stream QoS ($G = 1$) or per-class QoS ($G = \infty$).

As a result of this, DCQM is a two-tiered system as shown by Figure 3. First, streams are aggregated into classes according to the individual stream's loss requirement. Next, the streams are then mapped into groups inside a class. This results in a loss class having one or more groups associated with it. The number of groups depends on G and the

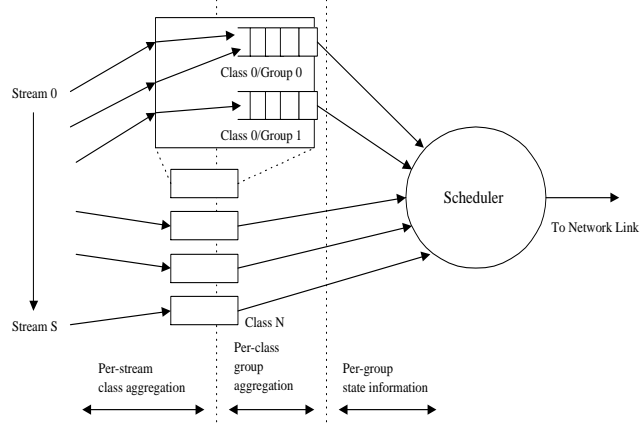


Figure 3: DCQM model

number of streams aggregated into the class. Prioritization is conducted on a per-group basis with each group having its own queue, which can offer finer grain QoS than the per-class prioritization.

3.3 Scalability vs. QoS Granularity

The goal of DCQM is to provide a scheme that adapts appropriately to system dynamics to balance scalability with QoS granularity. For a given media server, let G be the maximum group size, S be the number of streams being scheduled by the server, N be the number of classes, and K be a multiplicative factor that represents the additional amount of state information beyond N that can be maintained. Therefore, the general constraint on our system is as follows:

$$\frac{S}{G} \leq NK$$

Rearranging the equation yield the maximum group size G that is available to our system as:

$$G \geq \frac{S}{NK}$$

In our model, N is a fixed quantity and K and S vary upon server load and stream creation or termination, respectively. Therefore, DCQM should adapt G appropriately to the respective server loads in order to operate successfully. However, the above equation is only a general case where streams are aggregated uniformly between classes. To be more precise, the proper constraint for our system is as follows:

$$\sum_{i=1}^N \frac{S_i}{G} \leq NK$$

where S_i is the number of streams in class i . A further extension would be to examine the effect of variable group sizes between classes to offer QoS granularity as a class behavior. In this paper, G is considered to be constant across all classes.

Stream Join (Stream S)

begin

G: Maximum group size.

ClassGroups: set of groups whose class is Class(S).

LowGroup: a group in *ClassGroups* with the least members.

HighGroup: the highest priority group in *ClassGroups* with the most members.

if Count(*LowGroup*) + 1 > *G*

 Create *NewGroup*

 Add S to *NewGroup*

 Transfer *G*/2 streams to *NewGroup*

 Add *NewGroup* to *ClassGroups*

else

 Add S to *LowGroup*

end if

end.

Figure 4: DCQM Stream Join Algorithm

3.4 Group Membership

When dealing with group membership at the media server, several key issues arise. First, the stream join policy is examined for when a new stream is created at the server. Next, the stream leave policy is examined for when a stream is terminated. Finally, the adaptive control algorithm is examined for dynamic restructuring of the group size.

Stream Join

Figure 4 details the algorithm for when a new stream is created in the media server. First, the stream must be aggregated into an appropriate class. The class selection is dependent upon two factors, namely the loss tolerance of the stream and respective methods for aggregating streams into classes. The mapping of streams into loss classes is beyond the scope of this paper. For this paper, we will assume that a stream selects an appropriate loss class.

After a class has been selected for the stream, the next step is to select a group within the class for the stream. If an *open group*¹ exists within the class, the stream will join the group which has the least members. If all existing groups are *closed*², a new group must be created. In the case where $G > 2$, an additional step must take place. As a result

¹Open group is a group having less than *G* members.

²Closed group is a group having *G* members.

Stream Leave (Stream S)

begin

MemberGroup: Group that S belongs to.

ValidGroups: set of groups whose class is Class(S) &
whose member count $\leq G - \text{Count}(\text{MemberGroup}) - 1$

LeastGroup: the least priority group in ValidGroups
with the least members.

Remove S from MemberGroup

if ValidGroups is not empty then

 Transfer all streams in MemberGroup to

 LeastGroup

 Delete MemberGroup

end if

end.

Figure 5: DCQM Stream Leave Algorithm

of creating the new group with only a single member, the stream distribution between the groups is now unbalanced. Thus, due to the imbalanced stream distribution, the new stream will receive a finer grain QoS than is being received by the other groups in the class. This disparity becomes more pronounced as G increases. One method to correct this disparity is to move half of the streams of an existing group to the new group in order to more appropriately balance out the groups. Due to this, each of the two groups (the new group with $\lfloor \frac{G}{2} \rfloor + 1$ members and the old group with $\lceil \frac{G}{2} \rceil$ members) now has a finer grain QoS than all of the other groups in the class. Here, the group to be split (old group) should not be selected randomly, rather it should be the highest priority group, i.e., the group that has experienced the most losses. An alternate method to correct this disparity is to uniformly balance the load across groups by moving one or more streams from one or more groups to the new group. In this paper, we use the first approach. Note that the packets for moved streams are not copied, only the queue routing (pointer) information for those streams is changed.

Stream Leave

Figure 5 shows the algorithm for when a stream is terminated. When the member count in a group drops to a point where it can be combined with an existing open group, it is merged with such an open group in order to minimize the number of groups in a class and balance the stream distribution as much as possible. Similar to the stream join algorithm, there is a choice for selecting this open group. In the leave algorithm, the group with the lowest priority (i.e., the group that has experienced the lowest loss) should be the ideal candidate for combination. There could be other possibilities as well for the selection.

Adaptive Group Size

An adaptive group sizing algorithm can be invoked either periodically or following stream join/leave requests. In an increasing only model, the algorithm would increase G until steady state occurs. For a server that can experience a dynamic load, it may be more desirable to allow the algorithm to decrease G (better QoS) during low loads and increase G (better scalability) during high loads. However, when selecting the frequency of group rearrangement, one must also consider the cost of group rearrangement which depends on the change in G as well as the relative distribution of the groups.

In a DCQM media server, two sets of information are maintained. First, there is a limited amount of per-state information. This per-state information is the appropriate routing (queue pointer) information detailing which group a stream belongs to. This information is fairly static throughout the duration of a stream and thus presents scalability concerns only in terms of storage capacity, not CPU bandwidth. Second, the prioritization information is maintained on a per-group basis. The amount of prioritization information can range from per-stream (variable) to per-class (fixed). Thus, an increase in scalability occurs due to a reduction in both required storage capacity and required CPU bandwidth. The decrease in required storage capacity occurs because state information is maintained on only a per-group basis rather than on a per-stream basis. The decrease in required CPU bandwidth arises from the decrease in the number of queues being checked for deadline expiration and being considered for scheduling. For a non-adaptive server, the server administrator must determine the appropriate tradeoff between scalability and fine grain QoS. However, an adaptive server will appropriately adjust this tradeoff at a cost of scalability (CPU bandwidth involved in invoking the adaptive algorithm).

3.5 Proposed Schedulers

We propose two schedulers based on the DCQM model, a scheduler based on the DBP scheduler (DCQM-DBP) and another based on the DWCS scheduler (DCQM-DWCS). These schedulers run on top of the DCQM model via a two tiered approach.

The first level is handled by the DCQM model. The DCQM model is responsible for mapping streams into classes and then grouping them inside those classes. The creation and termination of streams is conducted according to the DCQM stream join and stream leave algorithms, respectively. State information is maintained on a per-group basis according to either the DBP or DWCS prioritization scheme.

The second level is then handled by the respective schedulers. In both DBP and DWCS, a set of queues is considered for scheduling based on the state information for each queue. Whereas both the original DBP and DWCS used a one-to-one mapping of streams to queues, DCQM simply abstracts the schedulers to deal with only queues such that each queue is now a group of streams rather than an individual stream. In DCQM, all groups are considered as candidates for scheduling with the state information of each group used for prioritization. Thus, class does not imply priority in DCQM, rather it only defines how the streams will be aggregated into groups.

In addition, groups must also be examined for removing packets that have missed their deadlines. As a result of this deadline checking, a tradeoff arises between scalability and QoS granularity. One extreme is to examine all groups for deadline expirations each time a packet is selected for scheduling. This gives the most accurate state information for each group when the scheduler is executed. However, this consumes a large CPU bandwidth, thus making the scheduler less scalable. In the case of per-stream groups, $G = 1$, this can represent significant scalability problems. Therefore, in [7], the notion of variable checking for deadline expiration was introduced. The scalability could be improved at a small cost to QoS by increasing the number of packets scheduled between two consecutive deadline expiration checks. In this paper, we will assume that all groups will be checked for deadline expiration each time a packet is scheduled.

4 Simulation Studies

We have evaluated the performance of the proposed DCQM schedulers through simulation studies. In our simulation studies, each stream is characterized by a loss class, stream period, and stream duration. As each packet is transmitted or dropped, the statistics such as packet loss and dynamic failure are recorded.

The schedulers were evaluated using dynamic failure rate as the performance metric. Dynamic failure rate can be used to evaluate the QoS granularity since a finer grain QoS will yield a lower dynamic failure rate. In our simulation, one millisecond (ms) is represented by one simulation clock tick. The streams for the simulation are generated as follows:

- The arrival of the streams follow Poisson distribution with a mean inter-arrival time between 14 ms and 22 ms.
- The period of a stream is exponentially distributed with a mean of 70 ms.
- The duration of a stream is exponentially distributed with a mean of 20,000 ms.
- Packets are assumed to be of fixed length.
- The server is assumed to have enough buffer space and hence packet dropping due to buffer overflow does not occur.

Each scheduler was evaluated using 4 classes of loss-tolerance with (m, k) values of $(5, 7)$, $(10, 14)$, $(15, 21)$, and $(20, 28)$ to yield approximate loss-tolerances of 30% (20/33 frames per second video). The schedulers were evaluated for loads whose total packet loss varied between 1% and 40%.

4.1 Packet Loss Rate vs. Dynamic Failure Rate

Figure 6 shows the packet loss-rate of the EDF, DBP, and DWCS schedulers. However, as is shown in Figure 7, loss-rate is not necessarily correlated with the dynamic failure rate. Whereas the loss-rate contains all packet losses

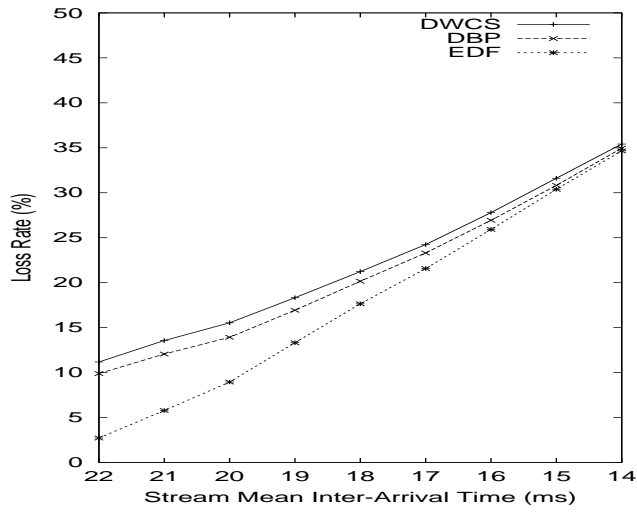


Figure 6: Effect of load on loss-rate

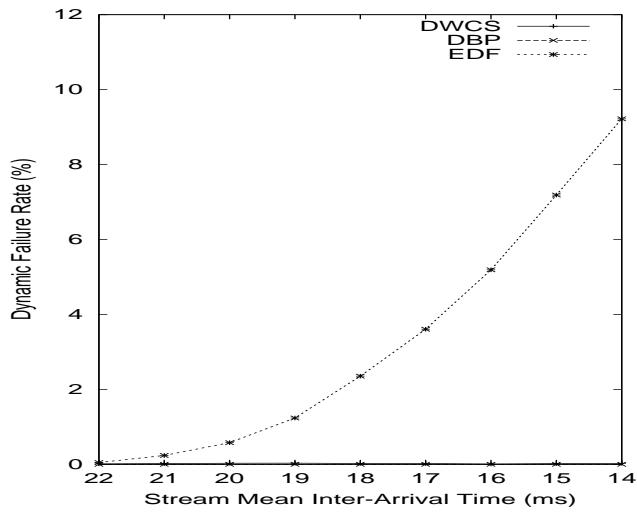


Figure 7: Effect of load on dynamic failure rate

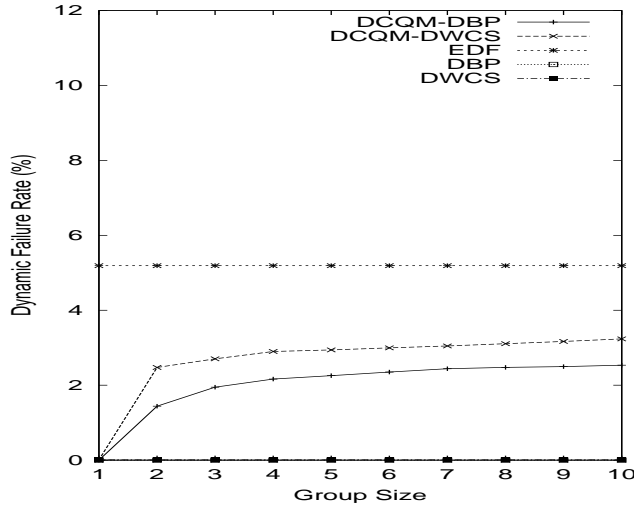


Figure 8: Dynamic failure rate of DCQM based schedulers

experienced by the schedulers, dynamic failure rate measures only those packets that had a negative impact (violation of loss window) on the perceived QoS of the clients. From the graphs, it can be seen that the DBP and DWCS are better than EDF in terms of dynamic failure rate and worse in terms of loss-rate.

The primary reason behind EDF having a lower loss-rate is due to the fact that the packets are scheduled independent of their stream history. Thus, once a packet is inside the queue, its priority is fixed relative to the deadlines of other packets. However, in DBP and DWCS, the priority of a given packet depends not only on the deadline, but also on the past transmission history of that queue/stream. Because of this, groups that have packets with tight deadlines may not be favored for scheduling compared to groups that encountered more losses within their loss windows. Thus, the chance of dropping packets for groups with low loss increases in order to decrease the chance of dropping packets for groups that are close to encountering dynamic failure.

4.2 Effect of group size on QoS in DCQM

Figure 8 shows the effect of G on dynamic failure rate for the DCQM based DBP and DWCS schedulers (DCQM-DBP and DCQM-DWCS) with a constant load (Stream Mean Inter-Arrival Time = 16 ms). The dynamic failure rates of EDF and DBP are also plotted as reference points on the graph. DCQM is able to successfully vary the dynamic failure rate in a predictable manner between the two extremes of $G = 1$ with per-stream state information to a much coarser grained QoS when $G = 10$. Even at worst case, when $G = \infty$ (i.e., one group per class), DCQM still subdivides streams into similar loss tolerance classes and maintains state information on a class-basis which will result in a better dynamic failure rate than EDF.

The change in dynamic failure rate (in Figure 8) as 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

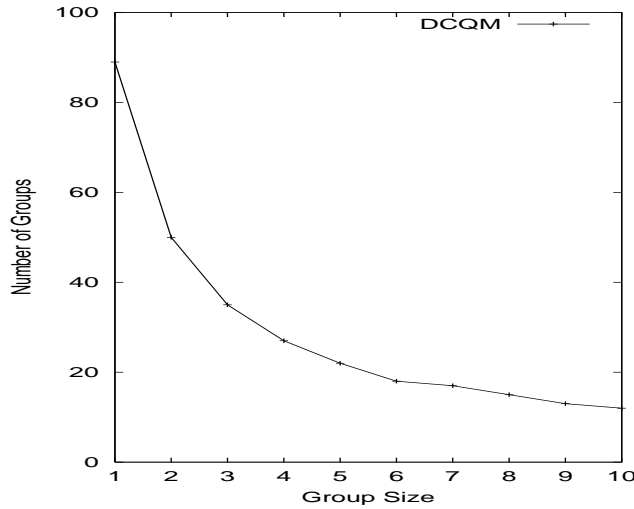


Figure 9: Number of groups in DCQM based schedulers

granularity and hence an increase in dynamic failure rate. The largest increase in dynamic failure rate occurs when stream aggregation is first introduced (when $G = 1$ and $G = 2$). Following this increase, the effect of G is not significant due to the fact that the variation in number of groups is significantly lower beyond $G = 3$.

4.3 Performance of DCQM based schedulers

Figure 10 shows the performance of the DCQM based schedulers versus the existing schedulers for varying stream loads. Four versions of the DCQM scheduler are used with DCQM-DWCS using $G = 2$ and $G = 6$; and DCQM-DBP using $G = 2$ and $G = 6$. From the figure, it can be seen that the dynamic failure rate of the DCQM schedulers increases with the increasing G . Also, it can be seen that the DCQM-DBP offers a slightly better performance than the DCQM-DWCS. Note that the increase in G represents an improvement in scalability.

5 Future Work

In order to improve the performance of the DCQM architecture, several enhancements could be investigated to improve performance and QoS granularity. These improvements include *class-wise history* and the notion of a *victim cache*.

5.1 Class-wise History

The first improvement, class-wise history, relates to the performance of the DCQM model. The class-wise history attempts to further aggregate the loss history scanning of the scheduler. Rather than evaluating each group present at the server, the history of all groups for a class could be consolidated into a single history for each group. Thus, this scheme incorporates a hierarchical scheme whereby the class history is scanned first followed by the history for the

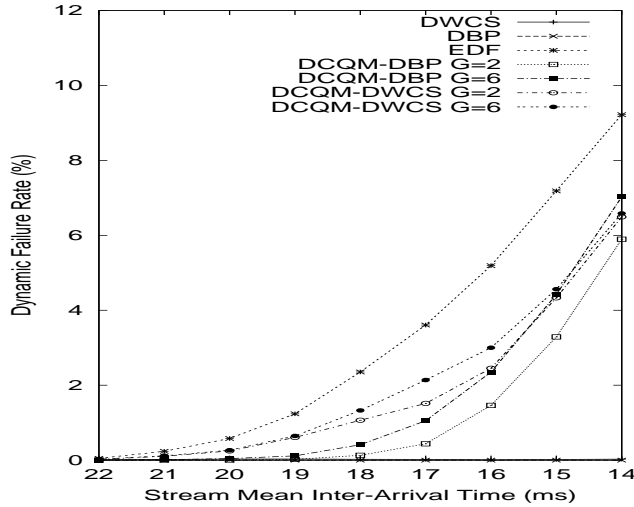


Figure 10: Dynamic failure rate of DCQM schedulers

groups within in a class.

The benefit to this scheme is that the scheduler would only be responsible for scanning a subset of the total groups available. Since the number of classes is fixed, the impact of the class history is a fixed amount as well. The main weakness of this scheme lies in the additional level of aggregation that is added on top of the group history. Although an individual group may suffer from poor QoS, the average QoS of the group may not be sufficient to promote the group for scheduling, thus potentially suffering from the same problems as outlined earlier in the paper. The effect of the additional level of aggregation would provide an interesting topic for examining the tradeoffs and benefits of such a scheme.

5.2 Victim Cache

The main problem suffered by a stream which suffers a poor QoS is the *anonymity effect* introduced by the aggregation of multiple streams. As was outlined in the simulation studies, the effect of additional aggregation significantly drops off once the group size moves beyond 3 or 4. Thus, in order to maintain a more linear QoS granularity curve, a scheme is necessary which would essentially undo the effect of the aggregation anonymity.

One such solution would be to employ a technique known as a victim cache [12]. In the context of QoS granularity, the victim cache would work as follows:

For a packet P_i which is dropped, add the parent stream (S_x) of the packet to the victim cache. All future packets from S_x are promoted ahead of other packets if the group containing S_x is selected for service until the QoS (loss) of S_x has been corrected.

The victim cache would have a finite length and be filled on a LIFO basis. The victim cache could be placed on a group-wise, class-wise, or global perspective where the perspective would affect the competing streams against which

the victim stream could be promoted. Existing packets for the stream could be searched for in an existing group buffer up to a fixed length and then appropriately promoted. Alternatively, a window of k packets could be examined for promotion each time the group is scheduled. The topic of the victim cache is an extremely interesting topic and has many subproblems that merit further investigation.

6 Conclusions

In this paper, we proposed a dynamic class-based queue management (DCQM) scheme that captures the tradeoff between scalability and QoS granularity in a media server by using the concept of groups inside the classes. To handle the dynamic creation and termination of streams in the server, we have proposed algorithms for stream join and stream leave under the DCQM scheme that maintain the balance of the group membership. Our simulation studies have shown that DCQM can be parameterized to operate between the two extremes of high scalability/coarse QoS granularity and low scalability/fine QoS granularity. Moreover, DCQM can be adapted to work with many existing schedulers. Thus, DCQM is a highly flexible model that provides the necessary scalability and QoS granularity required by media servers.

References

- [1] C.M. Aras, J.F. Kurose, D.S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proc. IEEE*, vol.82, no.1, 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] D. Ferrari and D.C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal of Selected Areas in Communications*, vol.8, no.3, pp.368-379, Apr. 1990.
- [4] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm guarantees," *IEEE Trans. Computers*, vol.44, no.12, pp.1443-1451, Dec. 1995.
- [5] P.V. Rangan and H.M. Vin, "Designing a multiuser HDTV storage server," *IEEE Journal of Selected Areas in Communications*, vol.11, no.1, pp.153-164, Jan. 1993.
- [6] A. Reddy and J. Wyllie, "I/O issues in multimedia systems," *IEEE Computer*, vol.27, no.3, pp.69-74, 1994.
- [7] R. West, K. Schwan, C. Poellabauer, "Scalable Scheduling Support for Loss and Delay Constrained Media Streams," *RTAS'99*.
- [8] Y.-J. Cho and C.-K. Un, "Performance analysis of reconstruction algorithms for packet voice communications," *Computer Networks and ISDN Systems*, vol. 26, pp. 1385-1408, 1994.

- [9] C. Liu, and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of ACM*, vol.20, no.1, pp.45-61, Jan. 1973.
- [10] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair-queuing algorithm," *Journal of Inter-networking Research and Experience*, pp. 3-26, Oct. 1990.
- [11] K.Nichols, S.Blake, F.Baker, and D.L.Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", *RFC 2474*, IETF, Dec. 1998.
- [12] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fullyassociative cache and prefetch buffers," *In Int. Symp. on Computer Architecture*, pp. 364-373, May 1990.