Best-effort Scheduling of (m,k)-firm Real-time Streams in Multihop Networks

A. Striegel and G. Manimaran

Dept. of Electrical and Computer Engineering
Iowa State University, USA
{adstrieg,gmani}@iastate.edu

Abstract. In this paper, we address the problem of best-effort scheduling of (m,k)-firm real-time streams in multihop networks. The existing solutions for the problem ignore scalability considerations because the solutions maintain a separate queue for each stream. In this context, we propose a scheduling algorithm, EDBP, which is scalable (fixed scheduling cost) with little degradation in performance. The proposed EDBP algorithm achieves this by allowing multiplexing of streams onto a fixed number of queues and by using the notion of a look-ahead window. In the EDBP algorithm, at any point of time, the best packet for transmission is selected based on the *state* of the stream combined together with the laxity of the packet. Our simulation studies show that the performance of EDBP is very close to that of DBP-M (a known algorithm for the problem) with a significant reduction in scheduling cost.

1 Introduction

Packet switched networks are increasingly being utilized for carrying real-time traffic which often require quality of service (QoS) in terms of end-to-end delay, jitter, and loss. A particular type of real-time traffic is a real-time stream, in which a sequence of related packets arrive at a regular interval with certain common timing constraints [1]. Real-time streams occur in many applications such as real-time video conferencing, remote medical imaging, and distributed real-time applications. Unlike non-real-time streams, packets in a real-time stream have deadlines by which they are expected to reach their destination.

Packets that do not reach the destination on time contain stale information that cannot be used. There have been many schemes in the literature to deterministically guarantee the meeting of deadlines of all packets in a stream [2, 3]. The main limitation of these schemes is that they do not exploit the ability of streams that can tolerate occasional deadline misses. For example, in teleconferencing, occasional misses of audio packets can be tolerated by using interpolation techniques to estimate the information contained in tardy/dropped packets.

On the other hand, there are schemes that try to exploit the ability of streams to tolerate occasional deadline misses by bounding the steady-state fraction of packets that miss their deadlines [4]. The main problem with these approaches is that the deadline misses are not adequately spaced which is often better than encountering spurts of deadline misses. 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 reconstructed audio signal may not be satisfactory. However, if the misses are adequately spaced, then interpolation techniques can be used to satisfactorily reconstruct the signal [5].

To address this problem, the (m, k)-firm guarantee model was proposed in [1]. A real-time stream with an (m, k)-firm guarantee requirement states that m out of any k consecutive packets in the stream must meet their respective deadlines. When a stream fails to meet this (m, k)-firm guarantee, a condition known as *dynamic end-to-end failure* occurs. The probability of dynamic end-to-end failure is then used as a measure of the QoS perceived by a (m, k) firm real-time stream.

Related Work: The message scheduling algorithms, such as Earliest Deadline First (EDF) and its variants [2, 3] that have been proposed for real-time streams are not adequate for (m, k)-firm streams because they do not exploit the m and k parameters of a stream. For scheduling of (m, k)-firm streams, a best-effort scheme has been proposed in [1] for single hop and has been extended to multihop in [6], with the objective of minimizing the dynamic end-to-end failure.

DBP Algorithms: A scheduling algorithm, Distance Based Priority (DBP), has been proposed in [1] in which each stream is associated with a state machine and a DBP value which depends on the current state of the stream. The state of stream captures the meeting and missing of deadlines for a certain number of previous packets of the stream. 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 of a stream, the higher its priority. 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 (k-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 and its next packet meets the deadline, then the stream transits to state MmM. In Figure 1, the failure states are mMm, Mmm, mmM, and mmm.

The Modified DBP (DBP-M) [6] is a multihop version of the original DBP algorithm. In DBP-M, for each stream, the end-to-end deadline is split into link (local) deadlines, along the path from source to destination of the stream, such that the sum of the local deadlines is equal to the end-to-end deadline. DBP-M confronts the problem introduced by multihop networks by having packets transmitted onward until they have missed their respective end-to-end deadlines. Thus, although a packet may miss its local deadline, it is still given a chance to meet its end-to-end deadline.

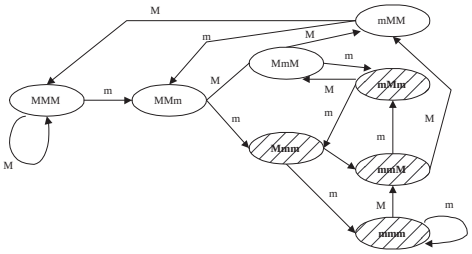

Fig. 1. DBP state diagram of a (2,3) stream

Motivation for Our Work: DBP and DBP-M use a separate queue for each stream at every node along the path of a stream (connection). That is, for each stream that is flowing across the network, a separate queue is created and per-stream state information is maintained at each node along the path of the stream. This solution is not scalable as the number of queues increases with the number of streams which results in high scheduling cost in terms of computational requirements. Similarly, the per-stream state information incurs overhead in terms of computational and memory requirements. The second aspect has been addressed by the Differentiated Services model [9]. In this paper, we address the first aspect by proposing an algorithm that reduces the scheduling cost by maintaining a fixed number of queues.

There exists a tradeoff between dynamic failure performance and the scheduling cost involved in achieving that performance. With the DBP and DBP-M extreme, a significant amount of scheduling cost is required to maintain the one queue per one stream ratio. Given a link that has N streams flowing across it, a DBP-M implementation requires N queues and requires $O(N)$ scheduling cost. However, this queue to stream ratio does deliver the best dynamic end-to-end failure performance for a given set of (m, k) streams.

In contrast, classical EDF scheduling and its variations require only one (or a fixed number of) queue(s) per link, i.e. the streams are multiplexed onto the queue(s), hence requiring a scheduling cost of $O(1)$. These methods incur the least scheduling cost but deliver the poorest end-to-end dynamic failure performance for (m, k) streams. Therefore, a better algorithm would require less scheduling cost than DBP-M but would provide better dynamic failure performance than classic EDF scheduling. This is the principal motivation for our work; in it, an integrated heuristic is proposed that allows multiplexing of streams while still providing adequate dynamic failure performance.

2 EDBP Scheduling Algorithm

The proposed EDBP algorithm aims at providing the same dynamic failure performance as that of DBP-M with a minimal scheduling cost by allowing queues to have more than one stream multiplexed. EDBP meets this goal by its integrated heuristic (EDBP value) that incorporates the DBP state of a stream together with the laxity of the packet. The EDBP algorithm has two key parts.

The first part deals with selecting the best (highest priority) packet from a window of packets in each queue (Steps 1-4). The second part selects the best packet from those packets chosen in the first part and transmits it (Steps 5-6). For the EDBP algorithm, the following notations are used:

Q_i: i^{th} queue; P_j: j^th packet in a queue
S_x: stream that produced P_j; w: window size
$EDBP(P_j)$: EDBP value of packet P_j
$EDBPS(S_x)$: EDBP state of stream S_x

The packets in a queue are stored in FIFO order. The cost of algorithm has two parts: queue insertion cost and scheduling cost. The insertion cost is high for EDF because it uses a priority queue and is unit cost for DBP and EDBP. EDF has a unit scheduling cost whereas the scheduling costs of DBP and EDBP are N and $w * Q$, respectively, where N is the number of streams and Q is the number of queues. The EDBP algorithm for transmitting a packet is given in Figure 2 below. Following it, the steps of the algorithm are discussed in detail.

Begin
For each queue Q_i perform Steps 1-4
1) For each P_j from P_0 to P_{w-1}, determine if the packet has missed its end-to-end deadline, such packets are then dropped.
2) Local Deadline $(P_j) = \frac{End-to-EndDeadline(P_j)}{Number\ of\ Hops\ in\ the\ path\ of\ stream\ S_x}$
   Laxity (P_j) = Local Deadline(P_j) - *current time*
   BucketWidth = Max (|(Laxity(P_0)|, |(Laxity(P_1)|,..., |(Laxity(P_{w-1})|) + 1
3) Calculate the EDBP value for each packet P_j.
   $EDBP(P_j)$ = BucketWidth * $EDBPS(S_x)$ + $Laxity(P_j)$
4) Select P_j that has the lowest EDBP value, called best packet.
5) Repeat steps 2-4, treating the best packet from each queue Q_i as a packet in an overall queue and with a window size (w) equal to the number of queues available.
6) Schedule the packet with the lowest EDBP value.
End

Fig. 2. EDBP scheduling algorithm for transmitting a packet

Step 1: The EDBP algorithm examines a window of w packets from each queue starting from P_0 (head packet in queue) up to P_{w-1} to determine if the packet has missed its end-to-end deadline. Thus, if a packet cannot meet its end-to-end deadline, the packet is dropped and the EDBP state of the corresponding stream for the node is adjusted accordingly. As with DBP-M, a packet is not dropped based on its local deadline. The use of the end-to-end deadline as a dropping mechanism is to give the packet a chance to meet its end-to-end deadline by scheduling the packet ahead of time in the downstream nodes across its path.

Step 2: In order to combine the EDBP state of a given packet P_j with the packet's laxity, the EDBP state must be converted to a meaningful value. Therefore, the EDBP algorithm uses the notion of buckets and offsets. The idea of a bucket is to group together the streams that have similar DBP states and the laxity is used as an offset inside the group (bucket). The local deadline cannot be used for the calculation of the bucket width as it is a relative value. However, the laxity of a packet is an absolute value related to the maximum end-to-end

deadline in the network. In this step, for each queue, a window of packets is examined to determine the packet with the largest absolute laxity value.

However, the maximum laxity value itself cannot simply be used to determine the bucket width. Consider the case where all of the packets in the window have missed their local deadline and the maximum laxity value is negative. Because the maximum laxity value is negative, priority inversion would occur as a lower EDBP heuristic value means a higher priority. To handle this case, the EDBP heuristic uses the maximum absolute laxity value. Thus, the value is always positive and priority inversion cannot occur.

Consider a second case where all of the packets have a local deadline of zero. Thus, without further modification, the EDBP state of the respective streams would essentially drop out of the EDBP heuristic. To handle this case, the maximum laxity value is further modified by adding one. This ensures that the modified laxity value will always be greater than or equal to one, thus eliminating the possibility of priority inversion or the elimination of the term corresponding to the EDBP state.

Steps 3, 4: Following the bucket width calculation, the best packet for the queue must be selected. The EDBP heuristic itself is divided into two parts, the bucket calculation and the bucket offset calculation. Each packet is placed into its appropriate bucket by multiplying the value of the EDBP state with the bucket width. After the bucket calculation is complete, each packet is appropriately offset into its bucket by adding the laxity value for that packet.

For the EDBP algorithm, a modification of the DBP state calculation is proposed. As with the initial DBP algorithm, the DBP value of a stream in the non-failing state is the number of transitions required to reach a failing state. Consider a (2,3)-firm stream where with a previous history of MMM. The DBP value would be 2, representing the two transitions required to reach a failing state. In the EDBP heuristic, the DBP state is expanded to allow negative values, thus allowing the EDBP state to discern between levels of dynamic failure between different streams. When the stream has reached 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. Under the initial DBP algorithm, a (2,3) stream with a history of Mmm would yield a DBP value of 0. However, when one examines the state diagram for the (2,3) stream, it is discovered that two transitions are required to return to a non-failing state. Under the EDBP algorithm, this stream would receive an EDBP value of -1, thus appropriately placing the packet at a priority level denoting its level of dynamic failure.

Best Packet Selection - Steps 5, 6: Once the best packet has been selected from each queue Q_i, the overall best packet is selected among these packets for transmission. To accomplish this, Steps 2-4 are repeated again with the following modifications. First, the queue being examined is now a queue of the best packets from each queue Q_i. Second, the window size for the EDBP algorithm is equal to the number of queues available. The best overall packet thus obtained will have the lowest EDBP value and is transmitted.

3 Performance Study

A network simulator was developed to evaluate and compare the performance of the EDBP algorithm with that of the DBP-M and EDF algorithms. The simulator uses a single queue for EDF, one queue per stream for DBP-M, and a fixed number of queues (which is an input parameter to the simulator) for EDBP. For our simulation studies, we have selected ARPANET as the representative topology. The algorithms were evaluated using the probability of dynamic failure as the performance metric. In our simulation, one millisecond (ms) is represented by one simulation clock tick.

Source and destination nodes for a stream were chosen uniformly from the node set. The local deadline for each stream was fixed with the end-to-end deadline equal to the fixed local deadline times the number of hops in the stream's path. The m and k values of a stream in the network are exponentially distributed with the condition that $m < k$. The mean inter-arrival time of streams in the network follow a Poisson distribution and stay active for an exponentially distributed duration of time. Packets are assumed to be of fixed size and each link has a transmission delay of one millisecond.

Effect of Number of Queues: In Figure 3, the effect of the number of queues on the probability of dynamic failures is examined in the EDBP algorithm. Thus, in the best case, the number of queues is equal to the number of streams. This is exemplified by the DBP-M algorithm. The EDBP algorithm has been split into two versions, one with $N/2$ queues and the other with $N/4$ queues ($N = 16$). Each increase in the number of queues results in an appropriate increase in the dynamic failure performance of the EDBP algorithm. For this figure, the performance of the EDF and DBP-M algorithms remain unchanged as the queue parameter has no effect on these algorithms. From Figure 3, one can deduce that an increase of the number of queues reduces the multiplexing degree that in turn increases the performance of the EDBP algorithm. The performance of the EDBP algorithm at N/2 queues is extremely close to the performance of the DBP-M algorithm while requiring only half of the scheduling cost of DBP-M.

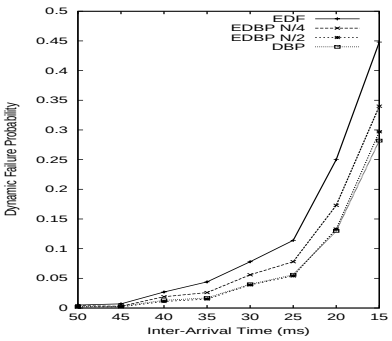

Fig. 3. Effect of No. of Queues

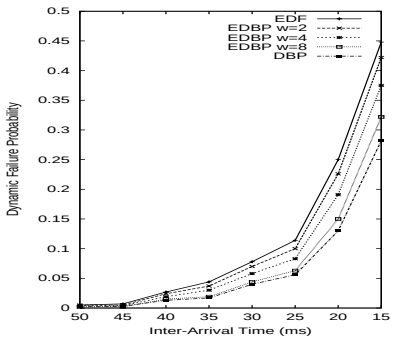

Fig. 4. Effect of Window Size

Effect of Window Size: However, in a given setting, it may not be practical or even possible to increase the number of queues available. Figure 3 repeats the same settings used in Figure 3, except that the window size is varied instead of

the number of queues. Three versions of the EDBP algorithm are examined with $w = 2, 4, 8$. As the window size increases, the dynamic failure performance of the EDBP algorithm increases because the window size offsets the penalty imposed by the multiplexing of streams onto a given queue.

When the effect of window size is compared to the effect of additional queues in the EDBP algorithm, our experiments show that the increase in queues produces a more profound effect than an increase in window size. The underlying cause is due to the multiplexing of streams onto queues. Consider a scenario in which a stream (S_x) with a small period (high rate) and another stream (S_y) with a large period (low rate) are multiplexed onto the same queue. In this case, S_x will have a higher chance of having its packets inside the window than S_y. This results in more dynamic failure for S_y. However, as the number of available queues increases, the chance of these streams being separated into different queues increases as well, thus explaining the difference in performance. Therefore, to obtain the best performance from the EDBP algorithm, the window size must be appropriately tuned to the degree of multiplexing.

4 Conclusions

In this paper, we have addressed the problem of best-effort scheduling of (m, k)-firm real-time streams in multihop networks. The proposed algorithm, EDBP, allows multiplexing of streams onto a fixed number of queues and aims at maximizing the dynamic failure performance with minimal scheduling cost. Our simulation studies have shown that the performance is close to that of the DBP-M algorithm with a significantly lower scheduling cost.

References

1. 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.
2. D. Ferrari and D.C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE JSAC*, vol.8, no.3, pp.368-379, Apr. 1990.
3. H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol.83, no.10, pp. 1374-1396, Oct. 1995.
4. D. Yates, D.T.J. Krouse, and M.G. Hluchyj, "On per-session end-to-end delay distributions and call admission problem for real-time applications with QoS requirements," in *Proc. ACM SIGCOMM*, pp.2-12, 1993.
5. 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.
6. W. Lindsay and P. Ramanathan, "DBP-M: A technique for meeting end-to-end (m,k)-firm guarantee requirements in point-to-point networks," in *Proc. IEEE Conference on Local Computer Networks*, pp. 294-303, Nov. 1997
7. S.S. Panwar, D. Towsley, and J.K. Wolf, "Optimal scheduling policies for a class of queues with customer deadlines to the beginning of service," *Journal of the ACM*, vol.35, no.4, pp.832-844, Oct. 1988.
8. S. Shenker and L. Breslau, "Two issues in reservation establishment," in *Proc. ACM SIGCOMM*, pp.14-26, 1995.
9. W. Weiss, "QoS with Differentiated Services," *Bell Labs Technical Journal*, pp. 44-62, Oct.-Dec 1998.