On Slot Allocation for Time-Constrained Messages in DQDB Networks

Ching-Chih Han†, Chao-Ju Hou‡, and Kang G. Shin†

†Real-Time Computing Laboratory
Dept. of Elec. Eng. and Comput. Sci.
The University of Michigan
Ann Arbor, MI 48109
{*cchan,kgshin*}*@eecs.umich.edu*

‡Computer Engineering Division
Dept. of Elec. and Comput. Eng.
The University of Wisconsin
Madison, WI 53706
jhou@ece.wisc.edu

Abstract

This paper addresses the issue of guaranteeing the timely delivery of isochronous messages with hard deadlines in a Distributed Queue Dual Bus (DQDB) network. We propose a slot allocation scheme which can allocate bandwidth for a set of isochronous message streams and provide deterministic deadline guarantees. The allocation scheme is guaranteed to find a feasible slot allocation (in the sense that all messages can be transmitted in a timely manner) as long as the total message density is less than or equal to a certain threshold, where the total message density is defined as the summation (over all message streams) of the ratio of maximum message size to message deadline. The implementation of this scheme is simple and does not require any change of the current DQDB standard.

1 Introduction

There has been an increasing need of timely and dependable communication services either for embedded real-time systems in automated factories and industrial process controls, or for interactive distributed services such as multimedia conferencing and video/audio virtual realities. Embedded real-time services are usually realized by executing a number of cooperating/communicating tasks on multiple processors before their deadlines imposed by the corresponding mission/function. Interactive distributed services need a certain amount of bandwidth to deliver video/audio frames in time consistent with human perception. Consequently, network architectures and protocols are required to provide users with a convenient means of guaranteeing message-transmission delay bounds.

The Distributed Queue Dual Bus (DQDB), one service of which is to provide time-constrained communication services, has been adopted by IEEE [1] as its candidate protocol for metropolitan and local area networks. DQDB medium access control (MAC) protocol offers three distinct classes of services: *isochronous* data service, *connection-oriented* data service, and *MAC to logical link control* (LLC) data service. The latter two services are for regular traffic and use the *Queued-Arbitrated* (QA) functions for slot allocation. The isochronous service is for real-time traffic and uses the *Pre-Arbitrated* (PA) functions.

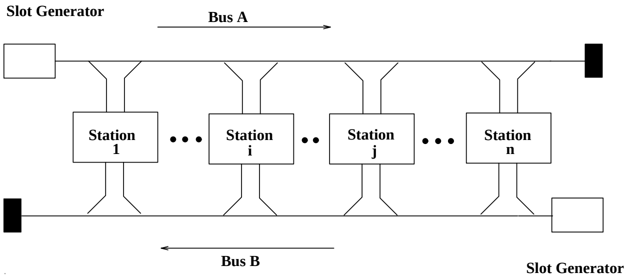

Figure 1: The DQDB network configuration.

The DQDB network consists of two high-speed ($\approx$ 150 Mb/s) unidirectional *slotted* buses running in opposite directions to which every station is connected (Fig. 1). Hence, there exists a transmission path from every station to every other station. Data transmission on both buses is slotted. Each slot can hold one 53-byte cell. Empty slots are generated by the slot generator at the head of each bus and transported "downstream." Access to slots is managed using the *access control field* (ACF) and the *virtual circuit identifier* (VCI). Each message stream of real-time traffic is given a unique VCI. Empty PA slots are reserved/marked by the slot generator at the head of the bus with their VCI fields set to the VCI of the appropriate isochronous message stream. A bit in the ACF is set to indicate whether or not a slot has been pre-assigned. The station with an isochronous message stream then watches for PA slots with the expected VCI and transmits its isochronous messages during those slots. The slot generator must ensure that the PA slots for each VCI are properly assigned so as to guarantee the timely delivery of the messages in each isochronous message stream.

The work reported in this paper was supported in part by the ONR under Grants N00014-92-J-1080 and N00014-94-1-0229, by the NSF under Grant MIP-9203895, and by the Wisconsin Alumni Research Foundation (WARF) under Grant 135-2080.

In this paper, we focus on how to provide deterministic timing guarantees for the isochronous service. To this end, we devise a slot allocation scheme based on the results in the *pinwheel problem* [2, 3]. We first characterize each isochronous message stream M_i with two parameters: relative message deadline D_i and maximum (total) message size C_i within any time interval of length D_i. Second, we formally define the *slot allocation problem* in a DQDB network. Our approach for solving this problem is to find a scheduler that allocates PA slots for isochronous message streams in such a way that in *any* time interval of length D_i, there are at least C_i PA slots allocated to message stream M_i, for all i. We then propose a slot allocation scheme based on the results derived in the pinwheel problem. The proposed scheme is guaranteed to find a feasible allocation schedule to satisfy the above criterion and ensures that all isochronous messages can be transmitted in time as long as the total message density is less than or equal to a certain density threshold, where the total message density is defined as the summation of the ratio of maximum message size to message deadline over all isochronous message streams, i.e., $\sum C_i/D_i$. We elaborate on the design of the on-line slot allocator and the slot allocation scheme, both of which are of polynomial-time complexity and can be readily used by the slot generator to produce a feasible allocation schedule. Finally, we show how to implement the proposed scheme as a daemon **SlotManager** that resides in the slot generator. One notable feature of the implementation is that it does not require any change of the current DQDB standards.

Numerous studies have been conducted on the QA services in terms of the queueing performance (e.g., [4, 5]), and the fairness issues (e.g., [6, 7]). By contrast, only a few studies have focused on slot allocation for real-time communication in DQDB networks (e.g., [8, 9]). Sha *et al.* [8] proposed a global priority scheme to guarantee message deadlines and prevent priority inversion. This scheme does not completely conform to the DQDB standards. Moreover, this scheme requires a large number of priority levels which are not currently supported in the DQDB standards. The scheme proposed by Saha *et al.* in [9] comes closest to ours. However, their scheme differs from ours in formulating and solving the problem. They adopt the *peak-rate* message model [10], in which each message stream M_i is characterized by three parameters: minimum message inter-arrival time P_i, maximum message size C_i, and relative message deadline D_i ($D_i \leq P_i$). We will show that their message model is just a special case of ours.

The rest of the paper is organized as follows. In Section 2, we discuss the message model used to describe isochronous message characteristics, and define the total message density. In Section 3, we formally define the slot allocation problem, and discuss how the results derived in the pinwheel problem can be used for our problem. In Section 4, we give an on-line slot allocator which takes a set of message streams as the input, and generates the corresponding slot schedule in polynomial time, and propose a slot allocation scheme. In Section 5, we compare our scheme with that proposed in [9]. We conclude the paper with Section 6.

2 Message model and message density

2.1 Message model

Let n be the number of message streams (each with a unique VCI) in the DQDB network. We use a message model similar to the (r, T)-*smooth* traffic model [11]. In our message model, each message stream M_i can be described by two parameters C_i and D_i, where

- C_i is the maximum total number of cells in a message stream M_i that can arrive in any time interval of length D_i (or simply, maximum message size), and
- D_i is the transmission deadline (or simply, the deadline) for messages in a message stream M_i, i.e., if a message of M_i arrives at time t, then it must be transmitted by time $t + D_i$.

This model is, in fact, a generalization of the commonly-used real-time *peak-rate* message model [10], in which each message stream M_i is characterized by three parameters C_i, D_i, and P_i, where

- P_i is the minimum inter-arrival period for message stream M_i, i.e., if the j-th message of M_i arrives at time t, then the $(j+1)$-th message in the stream will arrive at a time no earlier than $t + P_i$, for $j \geq 1$ (if messages in M_i arrive periodically, then P_i denotes the period),
- C_i is the maximum message size in stream M_i, i.e., C_i is the time needed to transmit a maximum-size message in M_i, and
- D_i ($\leq P_i$) is the transmission deadline for the messages in M_i.

Note that in our message model, the inter-arrival time of two successive messages in stream M_i is not required to be larger than or equal to D_i (i.e., more than one message may arrive in a time interval of length $\leq D_i$). However, the total message size measured in cells in M_i that arrive in any time interval of length D_i should be bounded by C_i. In the peak-rate model, during any time interval of length P_i, at most one message with message size at most C_i will arrive. And, $D_i \leq P_i$ implies that the total message size in M_i that arrive in any time interval of length D_i is bounded by C_i. Therefore, the peak-rate message model is simply a special case of our message model.

Since data transmission on the dual buses in a DQDB network is slotted, and since message arrivals may not be aligned with the starts of slots, we need to express D_i in units of slot time, and consider the effect of messages arriving in the middle of a slot. Let L_s denote the length of a slot, and D'_i denote the *effective* deadline expressed in slot times. D'_i can be expressed as

$$D'_i = \lfloor D_i/L_s \rfloor - 1. \tag{2.1}$$

The "-1" term in Eq. (2.1) comes from the fact that message arrivals may not be aligned with the starts of slots. For notational convenience, in the following discussion, we assume that D_i is the effective deadline expressed in units of slot time.

The slot generator must ensure that empty PA slots are properly assigned to each isochronous message stream M_i so that each message in M_i is transmitted within a time period $\leq D_i$ after its arrival as long as the maximum (total) message size in any time interval of length D_i is $\leq C_i$. In other words, the slot generator must assign at least C_i slots to a message stream M_i between the arrival time and the deadline of any message of M_i.

2.2 Message density of isochronous traffic

We define the *message density* of an isochronous message stream M_i as $\rho(M_i) = C_i/D_i$, and the total message density of the set of isochronous message streams $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$ as

$$\rho(\mathbf{M}) = \sum_{i=1}^{n} \rho(M_i) = \sum_{i=1}^{n} \frac{C_i}{D_i}. \tag{2.2}$$

Researchers in the field of real-time computing usually define the *schedulability* criterion for guaranteeing a set of periodic tasks using some priority-driven preemptive scheduling approach by giving the worst-case achievable processor utilization [12]. The message density defined above is similar to the processor utilization defined in real-time scheduling. In this paper, we design a slot allocation scheme for isochronous traffic in a DQDB network. Moreover, we give the worst-case achievable total message density ρ^* for the scheme. That is, the slot allocation scheme is guaranteed to find a feasible allocation (in the sense that all messages in $\mathbf{M}$ can be transmitted in time) for *any* set $\mathbf{M}$ of isochronous message streams as long as $\rho(\mathbf{M}) \leq \rho^*$.

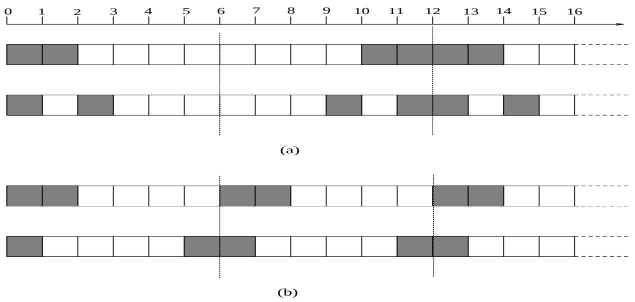

Figure 2: Four possible PA slot allocation patterns for a message stream with $C_i = 2$ and $D_i = 6$.

3 The slot allocation problem

As discussed in Section 2.1, each isochronous message must be transmitted within a time period $\leq D_i$ after its arrival. Note that the *exact* time when an isochronous message arrives is not known *a priori* and message arrivals in different message streams do not necessarily align with one another. Hence, to ensure that the above timeliness criterion will be satisfied, one must solve the following problem:

Problem 1: (**Slot Allocation Problem**) Given a set of isochronous message streams $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$, where $M_i = (C_i, D_i)$ is the i-th message stream, find a slot allocation scheme that allocates the PA slots in such a way that each message stream M_i is guaranteed to transmit each of its messages before the message deadline D_i. $\square$

Note that, according to our message model, the maximum message size in M_i is C_i. Therefore, if a message of M_i arrives at time t, then the slot allocation scheme must guarantee to allocate at least C_i slots for M_i during time interval $[t, t + D_i]$. It is easy to see that if a slot allocation scheme can allocate the PA slots in such a way that for *any* time interval of length D_i, at least C_i slots are allocated to M_i, for all i, then this allocation scheme is a solution to the above slot allocation problem. Consider, for example, Fig. 2, where four possible PA slot allocation patterns for a message stream M_i with $C_i = 2$ and $D_i = 6$ are shown. The PA slot allocation patterns in Fig. 2 (a) do not satisfy the criterion that for any time interval of length D_i, at least C_i slots are allocated to M_i, and a message of M_i that arrives at time t, $1 \leq t \leq 5$, cannot meet its delivery deadline $t + D_i$. In Fig. 2 (b), the PA slots are so allocated that the above criterion is fulfilled, and hence all messages can meet their delivery deadlines regardless of their arrival times.

An easy result regarding the total message density is stated below.

Lemma 1: If the total message density of a set of message streams $\mathbf{M}$ is larger than 1, then Problem 1 does not have solution.

Proof: The lemma follows simply by observing that each message stream M_i $(1 \le i \le n)$ must be granted slots at least C_i/D_i of the time. Clearly, this cannot be done for every i if $\sum_{i=1}^{n} \frac{C_i}{D_i} > 1$. □

In Section 4, we devise a slot allocation scheme to solve the slot allocation problem defined above. The theoretical base of the slot allocation scheme is grounded on some of the results in the *pinwheel problem* [2, 3] stated below.

Problem 2: (**Pinwheel Problem**) Given a multiset of n positive integers $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$, find an infinite sequence (schedule) over the symbols $\{1, 2, \ldots, n\}$ such that there is at least one symbol i within any subsequence of a_i consecutive symbols (slots).

For example, given a multiset $\mathbf{A} = \{2, 4, 5\}$, one solution sequence is $(1, 2, 1, 3,\ 1, 2, 1, 3, \cdots)$ where the subsequence $(1, 2, 1, 3)$ repeats forever. In this solution sequence, we can find one "1" in every $a_1 = 2$ consecutive symbols, one "2" in every $a_2 = 4$ consecutive symbols, and (at least) one "3" in every $a_3 = 5$ consecutive symbols.

If in the slot allocation problem, we require that the slot allocation scheme must be able to generate an infinite schedule such that at least C_i slots are allocated to M_i in any time interval of length D_i, then one can recognize the slot allocation problem as a generalization of the pinwheel problem. Therefore, one approach to solve the slot allocation problem is to view it as the pinwheel problem by considering $M_i = (C_i, D_i)$ as C_i copies of D_i in the corresponding pinwheel instance. For example, the slot allocation problem for $\mathbf{M} = \{(2, 4), (2, 5)\}$ can be transformed into the pinwheel problem with the instance $\mathbf{A} = \{4,\ 4,\ 5,\ 5\}$. However, this approach is not feasible in practice since the method of transforming a message stream M_i into C_i copies of D_i in the corresponding pinwheel instance makes the input size expand from n to $\sum_{i=1}^{n} C_i$ (which is a pseudo-polynomial expansion). In Section 4, we will devise a new on-line slot allocation algorithm which takes a set of message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \le i \le n\}$ as the input, and generates a slot allocation sequence such that for any consecutive D_i slots, at least C_i slots are allocated to M_i, for all i. Moreover, the algorithm takes $O(n)$ time in deciding which message stream the next slot is to be allocated to.

The question of how to schedule a pinwheel instance has been studied in [2,3]. Holte *et al.* [2] have shown that if a pinwheel instance $\mathbf{A}$ with total density ≤ 1 consists solely of multiples (i.e., $a_i | a_j$ for all $i < j$, where $a_i | a_j$ denotes a_i divides a_j, and $\rho(\mathbf{A}) = \sum_{i=1}^{n} 1/a_i \le 1$), then $\mathbf{A}$ is schedulable. For convenience of reference, we list this result in the following theorem.

Theorem 1: [2] Given a pinwheel instance $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$, if $a_i | a_j$ for all $i < j$, and $\rho(\mathbf{A}) \le 1$, then $\mathbf{A}$ is schedulable. □

Based on this result, Chan and Chin [3] have devised schedulers, **Sa** and **Sx**, to schedule larger classes of pinwheel instances. The basic idea of Schedulers **Sa** and **Sx** is the *single-integer reduction* technique, which aims to transform an arbitrary instance $\mathbf{A}$ to another instance $\mathbf{B} = \{b_1, b_2, \ldots, b_n\}$ that consists solely of multiples and $b_i \le a_i$ for all i. Since $b_i \le a_i$ (i.e., $\mathbf{B}$ is more restricted than $\mathbf{A}$), if we find a schedule for $\mathbf{B}$, then the schedule also satisfies the original constraints for $\mathbf{A}$. However, since $b_i \le a_i$ for all i, $\rho(\mathbf{B}) \ge \rho(\mathbf{A})$. Moreover, from Lemma 1 and Theorem 1, we know that $\mathbf{B}$ can be feasibly scheduled if and only if $\rho(\mathbf{B}) \le 1$. The density threshold ρ^* of $\mathbf{A}$ is then derived in such a way that as long as the density of $\mathbf{A}$ is less than or equal to ρ^* then $\rho(\mathbf{B}) \le 1$ (i.e., $\mathbf{B}$ is schedulable). In other words, with the single-integer reduction technique, one can schedule all pinwheel instances with densities $\le \rho^*$. Note, however, that if a pinwheel instance $\mathbf{A}$ has a total density larger than ρ^*, it does not necessarily mean that the instance is not schedulable by Scheduler **Sa** or **Sx**. $\mathbf{A}$ can be feasibly scheduled as long as the total density of the transformed set $\mathbf{B}$ is less than or equal to 1.

Assume $a_1 \le a_2 \le \cdots \le a_n$. Let a denote the smallest number in $\mathbf{A}$, i.e., $a = a_1$. Scheduler **Sa** finds a b_i for each a_i that satisfies

$$b_i = a \cdot 2^j \le a_i < a \cdot 2^{j+1} = 2b_i,$$

for some integer $j \ge 0$. Since the instance $\mathbf{B}$ consists solely of multiples, $\mathbf{B}$ can be feasibly scheduled as long as $\rho(\mathbf{B}) \le 1$. Chan and Chin call the operation *specializing* $\mathbf{A}$ *with respect to* $\{a\}$. Scheduler **Sx** is based on the same technique except that $\mathbf{A}$ is specialized with respect to $\{x\}$, where x is an integer and $a/2 < x \le a$. Starting from $x = a$, **Sx** specializes $\mathbf{A}$ with respect to $\{x\}$ until $x \ge a/2 + 1$ and chooses an x that minimizes $\rho(\mathbf{B})$, or until it finds an x which makes $\rho(\mathbf{B}) \le 1$ (or until it finds that no such integer exists). (Note that the operation of specializing $\mathbf{A}$ with respect to $\{x\}$ can be done in $O(n)$ time [3].) Therefore, **Sx** is more powerful than **Sa** in the sense that every instance that can be scheduled by **Sa** can also be scheduled by **Sx**. For example, **Sa** specializes $\mathbf{A} = \{4, 7, 8, 13, 24, 28\}$ with a total density of $0.672 \cdots \approx 2/3$ to $\mathbf{B} = \{4, 4, 8, 8, 16, 16\}$ with a total density of $7/8$. In comparison, **Sx** specializes $\mathbf{A}$ to **B'** $= \{3, 6, 6, 12, 24, 24\}$ with a total density of $5/6$ $(< 7/8)$.

It has been shown in [3] that the density thresholds for Schedulers **Sa** and **Sx** are 1/2 and 13/20, respectively. That is, as long as the total density of **A** is less than or equal to 1/2, the total density of the resulting set after specializing **A** with respective to $\{a\}$ will be less than or equal to 1, and hence, the resulting specialized set is schedulable (and so is the original set **A**). Similarly, as long as the total density of **A** is less than or equal to 0.65, the minimum total density of the resulting sets after specializing **A** with respective to $\{x\}$, for $a_1/2 < x \leq a_1$, will be less than or equal to 1.

4 DQDB slot allocation scheme

In this section, we first describe an on-line slot allocator which takes a set of message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \leq i \leq n\}$ with $D_i | D_j$ for all $i < j$, and $\rho(\mathbf{M}) \leq 1$ as the input, and generates the corresponding slot allocation schedule in $O(n)$ time per slot allocated. We also formally prove the correctness of the on-line slot allocator. Then, we present the slot allocation scheme which incorporates the on-line slot allocator to allocate slots to the message streams. We also discuss the issues of implementing the allocation scheme in a DQDB network.

4.1 On-line slot allocator

The on-line allocator **SlotAllocator** (Fig. 3) uses the rate-monotonic (RM) [12] (or, precisely, the deadline-monotonic (DM)) scheduling algorithm to assign message priorities so that the message streams with tighter deadline constraints get higher priorities. Specifically, **SlotAllocator** treats each message stream $M_i = (C_i, D_i)$ as a periodic task with execution time C_i and period (= deadline) D_i. At any time slot, **SlotAllocator** assigns the slot to the message stream with the highest priority among the *active* message streams, where an active message stream is a message stream whose slot requirements are *unfulfilled* at the current period interval. This technique is actually a discrete version of the technique used in [13] for scheduling real-time *distance-constrained task system* (DCTS). Note that in **SlotAllocator**, the for loop from line 5 to line 13 implements the RM/DM scheduling algorithm. The "if" statement (line 6) and the boolean variable *assigned* are used to locate the highest-priority message stream (i.e., the smallest index i) that has unfulfilled slot requirement (i.e., $c_i > 0$) with respect to the current d_i. The **SlotManager** (line 1) (to be discussed later) is the driver to the slot allocator.

```
SlotAllocator
/* d_i: slack time of the current message of M_i. */
/* c_i: remaining slot requirement w.r.t. current d_i. */
/* send(P,message): send a message to process P. */
/* receive(P,message): wait for and receive a message
   from process P. */

1. receive(SlotManager, M);
/* M is a set of message streams whose deadline constraint
   multiset consists solely of multiples and whose total
   message density is ≤ 1. */
2. for i ← 1 to n do { c_i ← C_i, d_i ← D_i; }
3. do {
4.     assigned ← false;
5.     for i ← 1 to n do {
/* note that D_1 ≤ D_2 ≤ ··· ≤ D_n */
6.         if (not assigned and c_i > 0) {
/* assign the current slot to message stream M_i */
7.             send(SlotManager, i);
8.             assigned ← true;
9.             c_i ← c_i − 1;
10.        } /* if */
11.        d_i ← d_i − 1;
12.        if (d_i == 0) { c_i ← C_i; d_i ← D_i; }
13.    } /* for */
/* the current slot will be left as a QA slot */
14.    if (not assigned) send(SlotManager, 0);
15. } forever
```

Figure 3: **SlotAllocator** process.

Let f_{ij} denote the $(j \cdot C_i)$-th slot assigned to message stream M_i, for $1 \leq i \leq n$ and $j \geq 1$. To facilitate the proof that **SlotAllocator** indeed produces a feasible slot allocation for a set of message steams whose deadline constraint multiset consists solely of multiples and whose total message density is less than or equal to one, we need the following lemma:

Lemma 2: Using **SlotAllocator** to allocate slots, $f_{1j} = (j-1) \cdot D_i + C_i$.

Proof: From process **SlotAllocator** (Fig. 3), it is easy to see that as long as $c_1 > 0$ the next slot will be assigned to message stream M_1 (i.e., M_1 has the highest priority). Since c_1 will be set to C_i every D_i slot times, the lemma follows directly from the definition of f_{ij}. □

Theorem 2: For a set of isochronous message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \leq i \leq n\}$, if $D_i | D_j$ for all $i < j$ and $\rho(\mathbf{M}) \leq 1$, then **SlotAllocator** will produce a feasible slot allocation schedule for **M**.

Proof: Similar to the proof in [13], it suffices for us to show that

(1) the first C_i slots assigned to message stream M_i are slots in slot interval $[1, D_i]$, i.e., $f_{i1} \leq D_i$, for all i,

(2) if slot t is assigned to M_i, then slot $(t + q \cdot D_i)$ is also assigned to M_i for any integer q such that $t + q \cdot D_i \geq 1$, and

(3) in *any* time interval of length D_i, there are C_i slots assigned to M_i, for all i.

We prove (1) and (2) by induction on message stream id i. Since M_1 has the tightest deadline D_1 (and hence, the highest priority), the C_1 consecutive slots from slot 1 to slot C_1 will be assigned to M_1 (i.e., the C_1 slots do not interleave with slots assigned to other message streams). Hence, $f_{11} = C_1 \leq D_1$ (otherwise, if $C_1 > D_1$, $\rho(\mathbf{M}) \leq 1$ does not hold) and (1) is satisfied for $i = 1$. From Lemma 2 and process **SlotAllocator**, it is easy to see that the next C_1 consecutive slots assigned to M_1 are slot $D_1 + 1$ through slot $D_1 + C_1$. Moreover, the j-th set of C_1 consecutive slots assigned to M_1 are slots $(j-1) \cdot D_1 + 1$ through slot $(j-1) \cdot D_1 + C_1$. Therefore, (2) is satisfied for $i = 1$.

Suppose (1) and (2) hold for all $i < k$. Now, since

(i) $D_i | D_k$, for all $1 \leq i < k$,

(ii) $f_{i1} \leq D_i$, for all $1 \leq i < k$, by the induction hypothesis of (1), and

(ii) if slot t is assigned to M_i, so is slot $(t + q \cdot D_i)$ for $1 \leq i < k$ and $q \geq 0$ by induction hypothesis of (2),

we know that the number of slots assigned to M_i, for $1 \leq i \leq k$, from slot 1 to slot D_k is exactly $C_i \cdot \frac{D_k}{D_i}$. If (1) is not true for M_k, we have $\sum_{i=1}^{k} C_i \cdot \frac{D_k}{D_i} > D_k$, or $\sum_{i=1}^{k} \frac{C_i}{D_i} > 1$, a contradiction to the assumption that $\rho(\mathbf{M}) \leq 1$.

Next, since $f_{k1} \leq D_k$, from process **SlotAllocator**, it is easy to see that the next slot that will be assigned to M_k is slot D_k+1 or later. Since $D_i|D_k$ for $1 \leq i < k$, by induction hypothesis of (2), the allocation pattern for the slots assigned to $M_1, M_2, \ldots, M_{k-1}$ in slot interval $[1, D_k]$ repeats in slot interval $[(j-1) \cdot D_k + 1, j \cdot D_k]$, for all $j > 1$. Moreover, the allocation pattern for the slots assigned to M_k in slot interval $[1, D_k]$ satisfies the deadline constraint of M_k, and will thus repeat in slot interval $[(j-1) \cdot D_k + 1, j \cdot D_k]$, for all $j > 1$. Note that the message streams, $M_{k+1}, M_{k+2}, \ldots, M_n$, with the looser deadlines (and hence, the lower priorities) have no effect on the allocation pattern of the higher-priority message streams $M_1, M_2, \ldots, M_k$. Hence, (2) is true for $i = k$.

From (1) and (2), we have that in each slot interval $[(j-1) \cdot D_i + 1, j \cdot D_i]$, there are (exactly) C_i slots assigned to message stream M_i, for $1 \leq i \leq n$ and $j \geq 1$. Now, to prove (3), we need to consider a time interval of length D_i with slot t_1 and slot t_2 as the first and the last slots, respectively, where $1 \leq t_1 \leq j \cdot D_i \leq t_2 \leq (j+1) \cdot D_i$, for some $j \geq 1$, and $t_2 - (t_1 - 1) = D_i$. Since the slot allocation pattern in slot interval $[t_1, j \cdot D_i]$ repeats in slot interval $[t_1 + D_i, (j+1) \cdot D_i] = [t_2 + 1, (j+1) \cdot D_i]$, and since by (1) and (2) the slot allocation pattern in slot interval $[j \cdot D_i + 1, (j+1) \cdot D_i]$ satisfies the deadline constraint (i.e., the number of slots in $[j \cdot D_i + 1, (j+1) \cdot D_i]$ allocated to M_i is C_i), it is obvious to see that the slot allocation pattern in time interval $[t_1, t_2]$ also satisfies the deadline constraint of M_i, i.e., there are C_i slots assigned to M_i in time interval $[t_1, t_2]$. □

```
SlotManager
/* Assume M = {M_i = (C_i, D_i) | 1 ≤ i ≤ n},
   where D_1 ≤ D_2 ≤ ··· ≤ D_n. */

/* Upon system initialization */
collect and update M;
specialize the deadline constraint set D of M to get M';
if (ρ(M') > 1) reject M and exit;
else {
    send(SlotAllocator, M);
    do {
        receive(SlotAllocator,i);
        if (i ≠ 0)
          inquire the VCIServer for message stream
          M_i's VCI and fill the VCI field of the current
          slot with message stream M_i's VCI;
        else
          tag the current slot as a QA slot;
        wait for the "time slot" to fully elapse;
    } forever
}
```

Figure 4: **SlotManager** process.

4.2 DQDB slot allocation scheme

We now describe the slot allocation scheme which incorporates **SlotAllocator** to allocate slots to the message streams. The process **SlotManager** (Fig. 4) is a driver to the slot allocator process **SlotAllocator**. **VCIServer** is responsible for mapping a message stream id to its corresponding VCI number.

To allocate slots for a set of general message streams $\mathbf{M}$ (i.e., $D_i | D_j$ for all $i < j$ may not necessarily hold), the slot manager **SlotManager** does the following steps.

Step 1. Specialize $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$ using the chosen specialization operation to get the *specialized* message stream set **M'** with the specialized deadline constraint multiset $\mathbf{D'} = \{D'_1, D'_2, \ldots, D'_n\}$. For example, if the specialization operation used is Scheduler **Sx**, then we find a D'_i for each D_i that satisfies $D'_i = x \cdot 2^j \leq D_i < x \cdot 2^{j+1} = 2D'_i$, for some integer $j \geq 0$, where x is an integer $\in (D_1/2, D_1]$ that results in the minimum $\rho(\mathbf{M'})$. (Note that $D'_i | D'_j$, for all $i < j$.)

Step 2. Check whether or not the total message den-

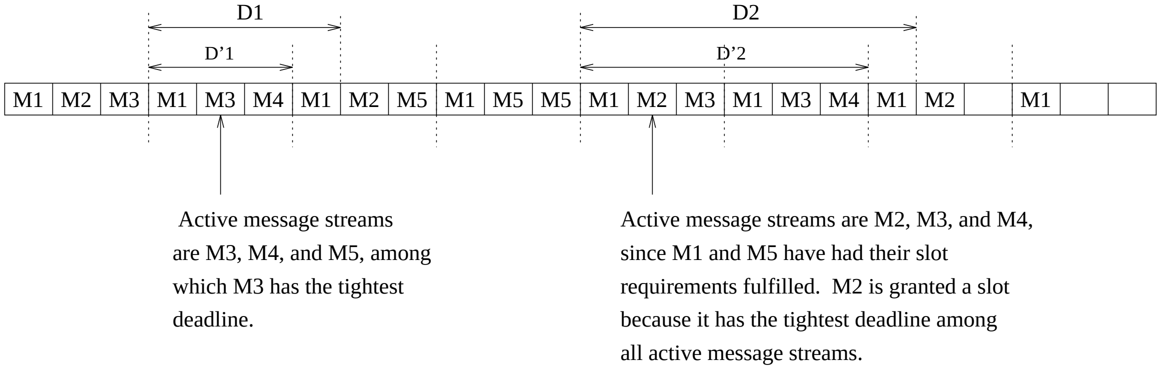

Figure 5: The slot allocation sequence for the set of message streams $\mathbf{M} = \{(1,4), (1,7), (2,13), (1,23), (3,28)\}$.

sity of the specialized message stream set **M'** $= \{M'_i = (C_i, D'_i) \mid 1 \le i \le n\}$ is less than or equal to 1, i.e., whether or not $\sum_{i=1}^{n} C_i/D'_i \le 1$. If not, reject **M** and stop. Otherwise, proceed to **Step 3**.

Step 3. Use the on-line allocator **SlotAllocator** to get the message stream id i and assign the next slot to message stream M_i. Note that **SlotAllocator** will generate an infinite sequence of message stream ids for **M'** such that there are C_i message stream M_i's ids within any subsequence of D'_i $(\le D_i)$ message stream ids generated.

Example: Consider a set of isochronous message streams, $\mathbf{M} = \{(1,4), (1,7), (2,13), (1,23), (3,28)\}$. Specializing the deadline constraint multiset $\mathbf{D} = \{4, 7, 13, 23, 28\}$ with respect to $\{x = 3\}$ yields **D'** $= \{3, 6, 12, 12, 24\}$. Since $D'_i | D'_j$, for all $i < j$, and $\rho($**M'**$) = \sum_{i=1}^{5} C_i/D'_i = 21/24 < 1$, by Theorem 2, we conclude that **SlotAllocator** can generate a feasible schedule for **M'**. By applying **SlotAllocator**, we obtain the slot allocation sequence shown in Fig. 5 in which the sequence repeats every $D'_5 = 24$ slots. Note that because of the RM/DM scheduling property, a slot is always assigned to a message stream with the tightest deadline among all *active* message streams (which have unfulfilled slot requirements with respect to their current deadlines). As one can readily see, there are at least C_i slots assigned to M_i in any time interval of length D'_i $(\le D_i)$, and hence, of length D_i. □

Implementation of the slot allocation scheme: In order to set up, maintain, and disconnect connections for message streams, **SlotManager**, **SlotAllocator**, and **VCIServer** co-reside in the slot generator of the bus. **SlotManager** considers each time-constrained connection between a source and destination pair as an isochronous message stream, and gathers/maintains all required information regarding the active connections, in particular, (C_i, D_i) for each message stream M_i.

At system initialization, **SlotManager** gathers the connection information, invokes **SlotAllocator** to generate the (virtually) infinite sequence of message stream ids with which **SlotManager** allocates PA slots accordingly (by filling the slots with appropriate VCIs). In order to set up a new connection, the source station sends a call setup request to **SlotManager**, specifying $M_{new} = (C_{new}, D_{new})$. Upon receiving the request, **SlotManager** specializes the augmented deadline constraint multiset $\mathbf{D} \cup \{D_{new}\}$ and checks whether or not the total message density after specialization is less than or equal to 1. If not, the request for establishing the connection is rejected. Otherwise, **SlotManager** notifies **VCIServer** of the new connection, which in turn assigns a new VCI for the connection. **SlotManager** notifies **SlotAllocator** and the source and destination stations of the approval of the new connection. **SlotAllocator** then re-calculates the corresponding new schedule, and initiates the new schedule at an appropriate time. An existing connection can be cleared either by the source or by the destination station. The call clear request is sent to **SlotManager** which responds simply by notifying **SlotAllocator** to delete the corresponding message stream and tag the slots originally assigned to the message stream as QA slots. Note that the implementation is simple, and does not require any changes in the current DQDB standards.

5 Comparison with related work

As mentioned in Section 1, the scheme proposed by Saha *et al.* in [9] comes closest to ours but differs in the approaches to formulating and solving the problem. The consequence is significant as explained below. As discussed in Section 2.1, they adopted the commonly-used peak-rate message model, which is a special case of our message model. They first devised a bandwidth allocation scheme based on cyclic reservation and derived the schedulability condition under two assumptions: (i) message streams are all periodic with periods $P_i = D_i$, for all i, and (ii) message arrivals are aligned with the starts of allocation cycles, and the length L of the allocation cycle evenly divides P_i for all i. They derived the schedulability condition as "if

$$\sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \frac{C_i}{(P_i/L)} \leq L,$$

then the set of message streams is schedulable, where $R_i \triangleq \frac{C_i}{P_i/L}$ is the average slot requirement of the message stream M_i per allocation cycle." (Note that L is the length of the allocation cycle and P_i/L is the number of allocation cycles in a time period P_i.) Then, in order to accommodate the fact that messages may arrive in the middle of an allocation cycle and thus may not utilize the allocation cycle, they modified R_i as $R'_i = C_i/((P_i - L)/L) = LC_i/(P_i - L)$. In order to relax the assumption that $P_i = D_i$, they consecutively modified R'_i as $R''_i = C_i/((D_i - L)/L) = LC_i/(D_i - L)$. The schedulability condition then becomes

$$\sum_{i=1}^{n} R''_i = \sum_{i=1}^{n} \frac{LC_i}{(D_i - L)} \leq L.$$

Now, if there exists some D_k such that $D_k \approx L$, then in most cases the schedulability condition does not hold, which implies that their scheme cannot guarantee to find a feasible allocation schedule for such a message stream set. And, it is also difficult to choose an appropriate L value if $GCD(D_1, D_2, \ldots, D_n) = 1$. Actually, the schedulability condition could lead to an erroneous result if L is larger than D_i, for some i (in which case $\frac{LC_i}{(D_i - L)} < 0$). One such example is $\mathbf{M} = \{(C_i, D_i, P_i) \mid i = 1, 2\} = \{(3, 4, 5), (3, 5, 5)\}$ and $L = 6$. Their schedulability condition yields $\frac{6 \cdot 3}{(4-6)} + \frac{6 \cdot 3}{(5-6)} = -27 < 6$; however, the message stream set is not schedulable (see Lemma 1 in Section 3). In contrast, our scheme can definitely find a feasible allocation schedule as long as the total message density, $\sum_{i=1}^{n} C_i/D_i$, is less than or equal to a certain density threshold (e.g., 0.65 if the specialization operation used is as that used in Scheduler **Sx**). Actually, our scheme can find a feasible allocation schedule for a message stream set as long as its total message density after specialization is less than or equal to 1.

6 Conclusion

We have proposed a slot allocation scheme for allocating bandwidth and guaranteeing the timely delivery of messages for isochronous message streams in DQDB networks. The proposed slot allocation scheme is based on the single-integer reduction technique **Sx**, and an on-line slot allocator of polynomial-time complexity. The resulting scheme is guaranteed to find a feasible allocation schedule for a message stream set as long as its total message density is ≤ 0.65.

References

[1] "IEEE Standards for Local and Metropolitan Area Networks: Distributed Queue Dual Bus (DQDB) Subnetwork of a Metropolitan Area Network (MAN)." IEEE 802.6, July 1991.

[2] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: a real-time scheduling problem," in *Proc. IEEE 22nd Hawaii Int'l Conf. on Systems Science*, pp. 693–702, January 1989.

[3] M. Y. Chan and F. Chin, "Schedulers for larger classes of pinwheel instances," *Algorithmica*, vol. 9, pp. 425–462, 1993.

[4] C. C. Biskikian, "Waiting time analysis in a single buffer DQDB (802.6) network," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 1565–1573, October 1990.

[5] W. Jing and M. Paterakis, "Message delay analysis of the DQDB subnetwork based on an approximate node model," *IEEE Trans. on Communications*, vol. 42, pp. 1120–1130, February/March/April 1994.

[6] E. L. Hahne and N. F. Maxemchuk, "Fair access of multi-priority traffic to distributed-queue-dual-bus networks," in *Proc. IEEE INFOCOM'91*, pp. 889–900, April 1991.

[7] N. S. V. Rao, K. Maly, and S. Dharanikota, "Average waiting time profiles of uniform DQDB model," in *Proc. IEEE INFOCOM'94*, vol. 1, pp. 1326–1333, June 1994.

[8] L. Sha, S. Sathaye, and J. K. Strosnider, "Scheduling real-time communication on dual link networks," in *Proc. IEEE Real-Time Systems Symposium*, pp. 188–197, December 1992.

[9] D. Saha, M. C. Saksena, S. Mukherjee, and S. K. Tripathi, "On guaranteed delivery of time-critical messages in DQDB," in *Proc. IEEE INFOCOM'94*, vol. 1, pp. 272–279, June 1994.

[10] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, pp. 122–139, January 1994.

[11] S. J. Golestani, "A framing strategy for congestion management," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1065–1077, Sept. 1991.

[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. of ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[13] C.-C. Han and K.-J. Lin, "Scheduling distance-contrained real-time tasks," in *Proc. IEEE Real-Time Systems Symposium*, pp. 300–308, December 1992.