On Slot Allocation for Time-Constrained Messages in Dual-Bus Networks

Ching-Chih Han, *Member*, *IEEE*, Chao-Ju Hou, *Member*, *IEEE*, and Kang G. Shin, *Fellow*, *IEEE*

Abstract—Several access schemes have been suggested for dual-bus network topology, e.g., DQDB [32], Fasnet [37], CRMA [41], and *Simple* [36]. It is therefore important to provide various services in this type of networks. This paper addresses the issue of guaranteeing the timely delivery of isochronous (real-time) messages with hard deadlines in slotted dual-bus networks. We propose a slot allocation scheme which can allocate bandwidth for a set of isochronous message streams and provide deterministic deadline guarantees.

The proposed slot 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 of the ratio of maximum message size to message deadline over all streams. We also discuss the implementation details of this scheme, and compare our scheme with another bandwidth allocation scheme proposed in [45].

Index Terms—MAC protocol, dual-bus networks, pinwheel problem, real-time communications, slot allocation.

1 Introduction

There has been an increasing need of timely and dependable communication services either for such embedded real-time applications as air-traffic control, automated factories, and industrial process controls, or for interactive distributed services such as multimedia conferencing and video/audio virtual realities. The former (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. One example is a monitor task that collects remote sensor data and displays the data at a control station. Failure to meet the deadlines of these tasks may lead to catastrophic consequences. The latter (interactive distributed) services need a certain amount of bandwidth to deliver video/audio frames in time consistent with human perception. Performance objectives used in conventional networks—such as maximizing the throughput or minimizing the response time—are not the most important concern to both types of applications. Instead, guaranteed and predictable performance must be ensured, and appropriate network architectures and protocols are required to provide users with a convenient means of guaranteeing message-transmission delay bounds.

The problem of guaranteeing the timely delivery of isochronous messages with hard deadlines has been studied by numerous researchers. The efforts have been directed mainly toward designing *medium access control* (MAC) protocols for local/metropolitan area networks (L/MANs). For example, the IEEE 802.4 token bus [4] (adopted for the Manufacturing Automation Protocol [18]), the IEEE 802.5 token ring [5], and FDDI [6] (developed by ANSI for high bandwidth fiber optic networks) adopt the *timed-token* MAC protocol for providing bounded medium access times. Both Agrawal et al. [1], [2], [3], [14] and Han et al. [26], [27] attempted to solve the *synchronous bandwidth allocation* problem for FDDI networks to meet the protocol constraint while transmitting all synchronous messages before their deadlines. In this paper, we focus on the problem of providing deterministic deadline guarantees to message streams with timing constraints in slotted dual-bus networks.

The dual-bus network considered in this paper consists of two high-speed unidirectional *slotted* buses running in opposite directions (Fig. 1). Every station is connected to both buses by active or passive taps which enable transmission on each bus. The two buses transport messages in opposite directions, so there exists a transmission path from every station to every other station. Data transmission on both buses is slotted. The slot generator at the head of each bus is responsible for generating empty slots and transport them "downstream" and for preassigning sufficient empty slots to isochronous message streams to ensure their timely delivery. (Although Fig. 1 shows slot generators as separate functional units, the slot generation function can be embedded in the stations at the two ends of the network.) Each slot contains an *Access Control Field* (ACF) and a payload. There are three fields in ACF that are of particular interest:

1) the *busy* bit, which indicates whether or not the slot is empty,
2) the *real-time* bit (or the *pre-arbitration* bit), which indicates whether or not the slot has been pre-assigned by the slot generator to some isochronous message stream, and

- *C.-C. Han and C.-J. Hou are with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210-1272, E-mail: {cchan, jhou}@ee.eng.ohio-state.edu.*
- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.*

Manuscript received 6 Feb. 1996; revised 4 Feb. 1997.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number 104119.0.

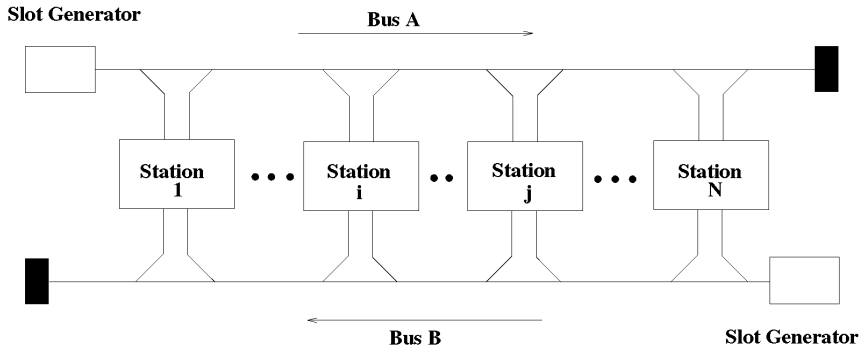


Fig. 1. The dual-bus network configuration.

3) the *virtual circuit identifier* (VCI), which indicates to which isochronous stream the slot is assigned if the real-time bit is set.

Access to the slots is managed by using these three fields. We assume that messages to be transmitted on the buses are divided into one or more fixed-length packets/cells, and packet/cell size matches the payload size of a slot, i.e., each message cell needs one slot time for its transmission.

The slotted dual-bus network configuration described above is general enough to accommodate several MAC schemes suggested for this topology, e.g., DQDB [32], Fasnet [37], CRMA [41], and *Simple* [36], to name a few. Under the slotted dual-bus network configuration, we first characterize each isochronous message stream M_i with two parameters: relative message deadline D_i and maximum (total) message size C_i (measured in packets) that can arrive within any time interval of length D_i. Second, we formally define the *slot allocation problem* in slotted dual-bus networks, and devise a slot allocation scheme as a solution to the problem. Our solution approach is to devise an on-line scheduler that preallocate slots to a set of isochronous message streams $\{M_i = (C_i, D_i) \mid 1 \le i \le n\}$ in such a way that in *any* time interval of length D_i, there are at least C_i slots allocated to M_i for all i. Based on the scheduler, we then propose a slot allocation scheme that can be readily used by the slot generator to generate slot allocation schedules. The proposed slot allocation scheme is guaranteed to find a feasible slot allocation schedule to satisfy the above criterion and ensures that all isochronous messages can be transmitted before their deadlines 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 of the ratio of maximum message size to message deadline over all isochronous streams, i.e., $\sum C_i/D_i$. Finally, we elaborate on how to implement the proposed scheme as a **SlotManager** daemon or a **SlotManager** chip that resides in the slot generator.

Numerous researchers have studied the slotted dual-bus networks in terms of the design of MAC schemes for non-real-time traffic [36], [37], [41], the fairness issues [8], [21], [22], [44], and the queuing performance [9], [10], [34]. By contrast, only a few of them have focused on slot allocation for real-time communication [11], [39], [40], [43], [45], [46], [47]. (Most of them are in the domain of DQDB networks perhaps except for [46], [47].) Potter et al. [43] proposed a request control scheme to guarantee bandwidth for queue-arbitrated access. In combination with a traffic shaping mechanism, Martini et al. [39], [40] proposed a guaranteed bandwidth (GBW) protocol to arbitrate the queue-arbitrated slots for the connection-oriented services. Both schemes proposed in [39], [40], [43] do not provide any performance guarantee for real-time traffic. Sha et al. [46], [47] proposed a global priority scheme to guarantee message deadlines and prevent priority inversion. The major drawback of the scheme is that it requires a large number of priority levels which may not be supported in realistic networks. Chan et al. [11] proposed a reservation-arbitrated access scheme exclusively for isochronous voice transport, and achieved statistical multiplexing in isochronous services by allowing voice packets to be occasionally dropped. As a result, their scheme may not be well-suited for embedded real-time systems with hard deadline constraints.

The scheme proposed by Saha et al. in [45] comes closest to ours. However, their scheme differs from ours in formulating and solving the problem. They adopt the *peak-rate* message model [7], 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 ($D_i \le P_i$). We will show in Section 2 that their message model is a special case of ours (i.e., more restricted than ours). They first devised a bandwidth allocation scheme based on cyclic reservation and derived the schedulability condition under two assumptions:

1) message streams are all periodic with periods $P_i = D_i$ for all i, and
2) 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 then relaxed these assumptions and modified their scheme to handle the general case. The resulting modified scheme may not be able to schedule message-stream sets with some $D_k \approx L$ in the worst case (see Section 5.1 for more details). In contrast, we formulate the problem in a very general setting, i.e., we formulate the slot allocation problem in such a way that any slot allocation scheme that solves this problem should be able to provide deterministic deadline guarantees for *arbitrary* isochronous message streams. By "arbitrary," we mean that the message arrivals in an isochronous stream are not required to be periodic or

separated by a minimum interarrival time, and the (first) message arrivals in different streams are not required to be in phase (i.e., aligned with one another) or aligned with any time instant. Moreover, the proposed scheme has an easy-to-test schedulability condition, i.e., as long as the total message density $\sum \frac{C_i}{D_i}$ is less than or equal to a certain threshold, the proposed scheme can always find a feasible slot allocation schedule.

The rest of the paper is organized as follows. In Section 2, we describe the MAC specification for real-time traffic used in the dual-bus network considered in this paper, discuss the message model used to characterize isochronous streams, and define the total message density of a set of isochronous streams. In Section 3, we formally define the slot allocation problem, and discuss how to solve the problem by generalizing the results of the *pinwheel problem* [12], [13], [28], [29]. In Section 4, we propose an on-line slot allocator which takes a set of message streams as the input and generates the corresponding slot allocation schedule in $O(n)$ time per slot allocated, and a slot allocation scheme of polynomial time complexity. We also discuss the implementation details of the proposed scheme there. In Section 5, we compare our scheme with that proposed in [45] and give a few remarks on possible extension to this work. We conclude the paper with Section 6.

2 MAC Protocol and Message Model

2.1 Access Control for Real-Time Traffic

Many MAC protocols have been developed for non-real-time traffic to ensure lack of starvation and some degree of fairness. For example, *Simple* [36], Fasnet [37], and CRMA [41] control the access to the bus by use of cycles, and by imposing a limit on the number of slots that can be used by each station in a cycle. DQDB [32] implements a distributed global queue by using two counters, *countdown* counter and *request* counter. What is lacking is a MAC protocol for real-time traffic. The intent of this paper is to lay a formal basis of such a protocol by devising a slot allocation scheme for real-time traffic.

Before elaborating on the detailed derivation of the proposed slot allocation scheme, we first give the specification for isochronous services based loosely on the IEEE 802.6 standard [32], [33]. The dual-bus network uses a *prearbitration* (PA) scheme for isochronous (real-time) traffic. Each isochronous message stream is given a unique VCI. The slot generator at the head of the bus is responsible for reserving/marking sufficient empty slots for isochronous message streams by setting

1) the real-time bits in the slots and
2) the VCI fields in the slots to the VCIs of appropriate isochronous message streams.

The stations with an isochronous stream then watch for prearbitrated (PA) slots with the appropriate VCIs and transmit their isochronous messages using those slots. If an empty slot is not pre-assigned for isochronous message streams[1] or if the VCI field in an empty preassigned slot does not match the VCI of any of the isochronous message streams emanating from a station, the station simply transports the PA slot downstream. The slot generator must ensure that the slots for each isochronous message stream are properly preassigned so as to guarantee the timely delivery of the messages in each isochronous stream.

1. Unassigned empty slots are arbitrated among stations using the MAC schemes for non-real-time traffic.

2.2 Message Model

Before delving into the issue of allocating network bandwidth for messages with delivery deadlines, one must specify the traffic characteristics and timing requirements of these messages. Let $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$ be a set of n isochronous message streams (each with a unique VCI) in the dual-bus network. Note that for each station, there may be zero, one, or more isochronous message streams emanating from it. We use a message model similar to the (r, T)-*smooth* traffic model [19], [20] in which each message stream M_i is described by a two-tuple (C_i, D_i):

- C_i is the maximum number of packets/cells in M_i that can arrive in any time interval of length D_i (or, simply, the maximum message size of M_i), and
- D_i is the *relative* transmission deadline (or simply, the deadline) for the messages in 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 two commonly-used real-time traffic models: the *peak-rate* model [7], and the *linear bounded* model [15], [16]. In the peak-rate model, each stream M_i is characterized by a triplet (C_i, D_i, P_i), where

- P_i is the minimum interarrival period for M_i, i.e., if the jth 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 all $j \geq 1$ (if messages in M_i arrive periodically, then P_i is the period),
- C_i is the maximum message size measured in cells in M_i, i.e., C_i is the number of slots 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.

In the (C, D)-smooth message model we used, the interarrival time of two successive messages in 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 not exceed C_i. In the peak-rate model, during any time interval of length P_i, at most one message of size less than or equal to C_i will arrive. $D_i \leq P_i$ implies that the total message size of the messages in M_i that arrive in any time interval of length D_i is bounded by C_i. The peak-rate model is simply a special case of the (C, D)-smooth message model because the peak-rate model is more restricted than the (C, D)-smooth model in the sense that any message stream that satisfies the characteristic (C_i, D_i, P_i) of the peak-rate model also satisfies the characteristic (C_i, D_i) of the (C, D)-smooth model.

In the linear bounded model, each message stream M_i is characterized by a two-tuple (ρ_i, β_i), where ρ_i is the maximum message arrival rate, and β_i is the maximum burst

size. A real-time traffic is said to follow the linear bounded model if the number of cells arrived in any time interval of length t is bounded by the linear function $\rho_i \cdot t + \beta_i$. The linear bounded model can be implemented by the *leaky bucket* [51] or *token bucket* [42] mechanism with a token generation rate ρ_i and a bucket size β_i. It is easy to see that by letting $C_i = \rho_i \cdot D_i + \beta_i$, the linear bounded model becomes a special case of the (C, D)-smooth model because the linear bounded model is more restricted than the (C, D)-smooth model in the sense that any message stream that satisfies the characteristic (ρ_i, β_i) of the linear bounded model also satisfies the characteristic (C_i, D_i) = ($\rho_i \cdot D_i + \beta_i$, D_i) of the (C, D)-smooth model.

Our slot allocation scheme is designed based on the above (C, D)-smooth model, and, hence, can also be used for message streams that conform to the peak-rate model or the linear bounded model. However, it is worth mentioning that the peak-rate model describes the "worst" case scenario of the (C, D)-smooth model in the sense that it is the most "difficult" situation for the slot allocation scheme to meet the deadline constraints of the messages in M_i. Note that if each message in M_i arrives $D_i = P_i$ units of time after the previous message and with a message size C_i, then *all* the C_i cells of each message in M_i must be transmitted within D_i time units after its arrival, and hence, the slot allocation scheme must ensure that there are enough slots (i.e., at least C_i slots) assigned to M_i during any time interval of length D_i.

Since data transmission on the dual buses is slotted, and message arrivals may not be aligned with slot boundaries, we need to express D_i in number of slots, and consider the fact that messages may arrive 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 floor function and the "–1" term in (2.1) come from the fact that D_i may not evenly divide L_s and message arrivals may not be aligned with slot boundaries. For notational convenience, we assume in the following discussion that D_i is the effective deadline expressed in number of slots. Note that as mentioned earlier, message cell size matches the payload size of a slot, so we may think of C_i as measured in slots.

With the (C, D) smooth message model, the notion of timing guarantee can be stated as: the slot generator must ensure that empty PA slots are properly assigned to each isochronous stream M_i so that each message in M_i is transmitted within a time period $\le D_i$ after its arrival as long as the maximum (total) message size in any time interval of length D_i is $\le C_i$. In other words, the slot generator must assign at least C_i slots to M_i between the arrival time and the deadline of any message of M_i, and because each message in M_i may arrive at any time, it also implies that the slot generator must assign at least C_i slots to M_i during *any* time interval of length D_i. We assume the presence of a suitable policing mechanism that marks cells which violate the traffic characteristics declared. No service is guaranteed to cells marked by the policing mechanism.

2.3 Message Density of Isochronous Traffic

We define the *message density* of an isochronous stream M_i as $\rho(M_i) = C_i / D_i$, and the total message density of a set of isochronous 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 [3], [38]. The message density defined above is similar to the processor utilization defined in real-time scheduling. In the following discussion, we propose a slot allocation scheme for isochronous traffic in a dual-bus 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 slot allocation schedule (in the sense that all messages in $\mathbf{M}$ can be transmitted in time) for *any* set $\mathbf{M}$ of isochronous streams as long as $\rho(\mathbf{M}) \le \rho^*$. Note, however, that $\rho(\mathbf{M}) > \rho^*$ does not necessarily imply that $\mathbf{M}$ cannot be feasibly scheduled by the proposed slot allocation scheme (more on this will be discussed later).

3 The Slot Allocation Problem

As discussed in Section 2.2, each isochronous message must be transmitted within a time period $\le D_i$ after its arrival. Hence, we formally define the slot allocation problem as follows.

Problem 1 (Slot Allocation Problem). Given a set of isochronous message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \le i \le n\}$, allocate the slots in such a way that each stream M_i is guaranteed to transmit each of its messages before the message deadline D_i. That is, if a message of M_i arrives at time t, enough slots must be allocated for M_i during time interval $[t, t + D_i]$ for the timely transmission of the message.

According to the message model, the maximum size of a message in M_i is C_i. Therefore, if a message of M_i with size C_i arrives at time t, then the slot allocation scheme must allocate at least C_i slots for M_i during time interval $[t, t + D_i]$. Note that the *exact* time when a message in M_i arrives is not known a priori and message arrivals in different streams do not necessarily align with one another. Hence, one way for a slot allocation scheme to meet the above timeliness criterion is to assign at least C_i slots to M_i for *any* time interval of length D_i.

Consider, for example, Fig. 2, where four possible slot allocation patterns for a stream M_i with $C_i = 2$ and $D_i = 6$ are shown. The slot allocation patterns in Fig. 2a 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, for $1 \le t \le 5$, cannot meet its delivery deadline $t + D_i$. In Fig. 2b, the slots are so allocated that the above criterion is fulfilled, and hence, all messages can meet their delivery deadlines regardless of their arrival times.

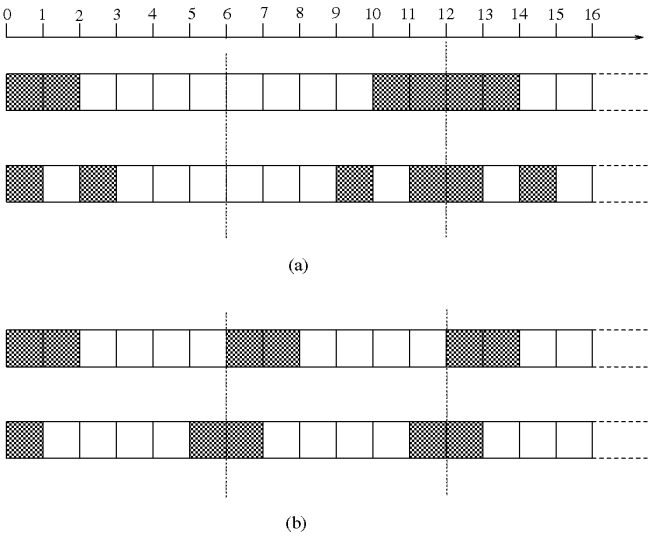


Fig. 2. Four possible slot allocation patterns for message stream $M_i = (C_i, D_i) = (2, 6)$.

A straightforward result on the total message density is stated below.

LEMMA 1. *If the total message density of a set of streams* **M** *is larger than* 1 *(i.e.,* $\rho(\mathbf{M}) > 1$*), then no feasible slot allocation schedule exists for* **M**.

PROOF. The lemma follows simply by observing that each 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 all streams if $\rho(\mathbf{M}) = \sum_{i=1}^{n} \frac{C_i}{D_i} > 1$. □

In Section 4, we propose a scheme to solve the slot allocation problem defined above. The theoretical base of the proposed slot allocation scheme is grounded on some of the results of the pinwheel problem stated below.

PROBLEM 2 (The Pinwheel Problem) [12], [13], [28], [29]. 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, …) 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.

There are two important results of the pinwheel problem upon which we will build our slot allocation algorithm:

1) If a pinwheel instance **A** with total density ≤ 1 consists solely of multiples (i.e., $a_i \mid a_j$ for all $i < j$, where $a_i \mid a_j$ denotes a_i divides a_j, and $\rho(\mathbf{A}) = \sum_{i=1}^{n} 1 / a_i \le 1$), then **A** is schedulable;
2) If a pinwheel instance **A** consists of only two distinct numbers (i.e., instances of the form $\{b, b, \ldots, b, c, c, \ldots, c\}$) with total density ≤ 1, then **A** is schedulable.

THEOREM 1 [29]. *Given a pinwheel instance* $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$, *if* $a_i \mid a_j$ *for all* $i < j$, *and* $\rho(\mathbf{A}) \le 1$, *then* **A** *is schedulable.*

THEOREM 2 [28]. *Given a pinwheel instance* $\mathbf{A} = \{b, b, \ldots, b, c, c, \ldots, c\}$ *with* p *b*s *and* q *c*s, *if* $\rho(\mathbf{A}) = p/b + q/c \le 1$, *then* **A** *is schedulable.*

Based on these two results, Chan and Chin [12], [13] devised several schedulers, e.g., **Sa**, **Sx**, **Sby**, and **Sxy**, to schedule larger classes of pinwheel instances. The basic idea behind Schedulers **Sa** and **Sx** is the *single-integer reduction* technique, which aims to transform an arbitrary instance **A** to another instance $\mathbf{A}' = \{a_1', a_2', \ldots, a_n'\}$ that consists solely of multiples and $a_i' \le a_i$ for all i. From Lemma 1 and Theorem 1, we know that **A'** can be feasibly scheduled if and only if $\rho(\mathbf{A}') \le 1$. Since $a_i' \le a_i$ (i.e., **A'** is more restricted than **A**), if we find a schedule for **A**', then the schedule also satisfies the original constraints for **A**.

Without loss of generality, we can assume that $a_1 \le a_2 \le \cdots \le a_n$ and that the smallest number in **A** is a, i.e., $a = a_1$, in the following discussion. Scheduler **Sa** finds an a_i' for each a_i such that

$$a_i' = a \cdot 2^j \le a_i < a \cdot 2^{j+1} = 2a_i'.$$

for some integer $j \ge 0$, i.e., $a_i' = a \cdot 2^{\lfloor \log(a_i/a) \rfloor}$ (note that all logarithms in this paper are to the base 2). This operation is called *specializing* **A** *with respect to* $\{a\}$. Since the instance $\mathbf{A}' = \{a_1', a_2', \ldots, a_n'\}$ consists solely of multiples, as long as $\rho(\mathbf{A}') \le 1$, by Theorem 1, **A**' is schedulable. **Sa** then uses an algorithm, **SpecialSingle**, proposed in [13] to find a feasible schedule for **A**'. Since $a_i' \le a_i$ for all i, the schedule found for **A**' is also a feasible schedule for **A**.

Scheduler **Sx** is based on the same technique as Schedule **Sa** except that **A** is specialized with respect to $\{x\}$, where x is an integer and $a_1/2 < x \le a_1$. Starting from $x = a_1$, **Sx** specializes **A** with respect to $\{x\}$ until $x \ge a_1/2 + 1$ and chooses an x that minimizes $\rho(\mathbf{A}')$, or until it finds an x which makes $\rho(\mathbf{A}') \le 1$ (or until it finds that no such integer exists). Therefore, **Sx** is more powerful than **Sa** in the sense that every pinwheel instance that can be scheduled by **Sa** can also be scheduled by **Sx**. For example, in **Sa**, $\mathbf{A} = \{4, 7, 8, 13, 24, 28\}$ with a total density of $0.672\cdots$ ($\approx 2/3$) is specialized with respect to $\{4\}$ to get $\mathbf{A}' = \{4, 4, 8, 8, 16, 16\}$ with a total density of $7/8$. In comparison, in **Sx**, **A** is specialized with respect to $\{3\}$ to get $\mathbf{A}' = \{3, 6, 6, 12, 24, 24\}$ with a total density of $5/6$ ($< 7/8$).

Schedulers **Sby** and **Sxy** are based on the *double-integer reduction* technique, and make use of Theorem 2 (as well as Theorem 1). In general, the double-integer reduction technique specializes a pinwheel instance **A** with respect to two positive integers $\{b, c\}$, where $b \le c < 2b$ and $b \le a = a_1$. That is, it finds an a_i' for each $a_i \in \mathbf{A}$ such that

$$a_i' = \begin{cases} b \cdot 2^j & \text{if } b \cdot 2^j \le a_i < c \cdot 2^j \\ c \cdot 2^j & \text{if } c \cdot 2^j \le a_i < b \cdot 2^{j+1}, \end{cases}$$

for some integer $j \ge 0$, i.e., $a_i' = \max\{b \cdot 2^{\lfloor \log(a_i/b) \rfloor}, c \cdot 2^{\lfloor \log(a_i/c) \rfloor}\}$. Schedulers **Sby** and **Sxy** differ from each other only in how to select the two integers b and c for the specialization operation.

After the specialization operation is performed, the Schedulers then use an algorithm **SpecialDouble** described in [13] to schedule the specialized pinwheel instance **A**'. A detailed account of all these schedulers/algorithms can be found in [12], [13], [28].

Note that since $a_i' \le a_i$ for all i, we have $\rho(\mathbf{A}') \ge \rho(\mathbf{A})$. The density threshold ρ^* of **A** is then derived in such a way that as long as the total density of **A** is less than or equal to ρ^* then $\rho(\mathbf{A}') \le 1$ (i.e., **A**' is schedulable). In other words, one can schedule all pinwheel instances with densities $\le \rho^*$.It has been shown in [12], [13] that the density thresholds for Schedulers **Sa**, **Sx**, **Sby**, and **Sxy** are 0.5, 0.65, 0.6964, and 0.7, respectively. Note, however, that if a pinwheel instance **A** has a total density larger than ρ^*, it does not necessarily imply that the instance is not schedulable. Instance **A** can be feasibly scheduled as long as the total density of the transformed set **A**' is less than or equal to 1.

In the slot allocation problem, if we require that the slot allocation scheme must be able to generate an infinite slot allocation sequence such that at least C_i slots are allocated to M_i in any time interval of length D_i, it follows that the slot allocation problem is a generalization of the pinwheel problem (note that in the pinwheel problem, $C_i = 1$ for all i). Therefore, one plausible approach to 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, an instance $\mathbf{M} = \{(2, 5), (3, 7)\}$ of the slot allocation problem can be transformed into the pinwheel problem with the instance $\mathbf{A} = \{5, 5, 7, 7, 7\}$. Any occurrence of symbol "1" or "2" in the pinwheel schedule is treated as a slot allocated to M_1 and any occurrence of symbol "3," "4," or "5" in the pinwheel schedule is treated as a slot allocated to M_2 in the slot allocation problem. It is easy to see that if the pinwheel schedule is feasible for the pinwheel instance **A**, then the corresponding slot allocation schedule is also feasible for the instance **M** of the slot allocation problem. However, this approach is not feasible in practice, since the method of transforming a 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 pseudopolynomial expansion). Thus, the schedulers used to solve the pinwheel problem *cannot* be directly applied to the slot allocation problem defined here.

Moreover, although both the scheduling algorithms **SpecialSingle** and **SpecialDouble** designed for the pinwheel problem are effective (i.e., both are parallel fast on-line schedulers and need $O(n)$ hardware [13], [29]), it is not clear whether or not they can be modified to solve the slot allocation problem in which the maximum message size C_i, $1 \le i \le n$, can be any positive integer and arbitrarily large, instead of 1 as in the pinwheel problem. In Section 4, we focus on the single-integer reduction technique and propose a new, simple on-line slot allocation scheme to handle a set of isochronous streams with arbitrary maximum message sizes, C_is, with $O(n)$ time per slot allocated and $O(1)$ hardware. Note, however, that the double-integer reduction technique can also be applied to the proposed slot allocation scheme to further improve the performance.

4 The Slot Allocation Scheme

In this section, we first describe an on-line slot allocator which takes a set of message streams $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \le i \le n\}$ with $D_i \mid D_j$ for all $i < j$, and $\rho(\mathbf{M}) \le 1$ 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 resides in the slot generator and incorporates the on-line slot allocator to allocate slots for a set of general message streams (i.e., $D_i \mid D_j$ for all $i < j$ may not necessarily hold). Finally, we discuss the issues of implementing the slot allocation scheme in a dual-bus network.

4.1 On-Line Slot Allocator

The on-line allocator **SlotAllocator** (Fig. 3) uses the rate-monotonic (RM) [38] (or, precisely, the deadline-monotonic (DM)) rule to assign message priorities so that the 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** always assigns the slot to the stream with the highest priority among the *active* streams, where an active stream is one whose slot requirements are *unfulfilled* at the current period interval. Note that in **SlotAllocator**, the for loop from line 5 to line 13 implements the RM/DM priority assignment rule. The "if" statement (line 6) and the Boolean variable *assigned* are used to locate the highest-priority 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 further discussed later) is the driver to the slot allocator. If there is no message stream with unfulfilled slot requirement (i.e., $c_i = 0$ for all i) with respect to the current d_i, **SlotAllocator** will inform **SlotManager** that a regular traffic (non-real-time) slot should be issued (line 14 of **SlotAllocator**). Note that the index 0 in line 14 denotes that the slot should be marked as a regular traffic slot.

Let f_{ij} denote the $(j \cdot C_i)$th slot assigned to message stream M_i, for $1 \le i \le n$ and $j \ge 1$. In the following theorem, we prove 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.

THEOREM 3. *For a set of isochronous message streams* $\mathbf{M} = \{M_i = (C_i, D_i) \mid 1 \le i \le n\}$, *if* $D_i \mid D_j$ *for all* $i < j$, *and* $\rho(\mathbf{M}) \le 1$, *then* **SlotAllocator** *(which is based on the RM/DM priority assignment) will allocate* C_i *slots to* M_i *in any time interval of length* D_i, *for all* i.

PROOF. It suffices for us to show that

1) the first C_i slots assigned to M_i are those slots in interval $[1, D_i]$, i.e., $f_{i1} \le D_i$, 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 $t + q \cdot D_i \ge 1$, and

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 and wait for its
reception by P. */
/* receive(P, message): wait for and receive a message from
process P. */
/* M = {M_i = (C_i, D_i) | 1 ≤ i ≤ n} is a set of isochronous message
streams with D_i | D_j for all i < j, and ρ(M) ≤ 1. */

1. receive(SlotManager, M);
2. for i ← 1 to n do {c_i ← C_i, d_i ← D_i;}
3. do {/* note that D_1 ≤ D_2 ≤ ... ≤ D_n */
4.    assigned ← false;
5.    for i ← 1 to n do {
6.       if (not assigned and c_i > 0) {
7.          send(SlotManager, i); /* assign the current slot to
            message stream M_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 */
14.   if (not assigned) send(SlotManager, 0); /* the current slot
      will be left as a regular traffic slot */
15.} forever
```

Fig. 3. The **SlotAllocator** process.

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

We prove conditions 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, $f_{11} = C_1 \le D_1$ (otherwise, if $C_1 > D_1$, $\rho(\mathbf{M}) \le 1$ does not hold) and condition 1 is satisfied for $i = 1$. From the **SlotAllocator** process, 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 jth set of C_1 consecutive slots assigned to M_1 are slots $(j-1) \cdot D_1 + 1$ through slot $(j-1) \cdot D_1 + C_1$. (Note that M_1 has the highest priority.) Therefore, condition 2 is satisfied for $i = 1$.

Now, suppose conditions 1 and 2 hold for all $i < k$. Since

i) $D_i \mid D_k$, for all $1 \le i < k$,
ii) $f_{i1} \le D_i$, for all $1 \le i < k$, by the induction hypothesis of condition 1, and
iii) if slot t is assigned to M_i, so is slot $(t + q \cdot D_i)$ for all $1 \le i < k$ and $q \ge 0$ by induction hypothesis of condition 2,

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

Next, since $f_{k1} \le D_k$, from the **SlotAllocator** process, it is easy to see that the next slot to be assigned to M_k is slot $D_k + 1$ or later. Since $D_i \mid D_k$ for $1 \le i < k$, by induction hypothesis of condition 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 looser deadlines (and hence, lower priorities) have no effect on the allocation pattern of the higher-priority streams $M_1, M_2, \ldots, M_k$. Hence, condition 2 is true for $i = k$.

From conditions 1 and 2, in each slot interval $[(j-1) \cdot D_i + 1, j \cdot D_i]$, there are (exactly) C_i slots assigned to M_i, for $1 \le i \le n$ and $j \ge 1$. Now, to prove condition 3, we need to consider a time interval of length D_i with slots t_1 and t_2 as the first and the last slots, respectively, where $1 \le t_1 \le j \cdot D_i \le t_2 \le (j+1) \cdot D_i$, for some $j \ge 1$, 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 conditions 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 $[t_1, t_2]$. □

4.2 Slot Allocation Scheme

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

To allocate slots for a set of general message streams **M**, **SlotManager** performs the following steps.

Step 1. Upon system initialization, gather/maintain the required information regarding the connection requests

SlotManager

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

/* Upon system initialization */
1. collect and update M;
2. specialize the deadline constraint multiset D of M to get D' (M');
3. if (ρ(M') > 1) reject M and exit;
4. else {
5.    send(SlotAllocator, M');
6.    do {
7.       receive(SlotAllocator, i);
8.       if (i ≠ 0)
9.          inquire the VCIServer for stream M_i's VCI and fill
            the VCI field of the current slot with stream M_i's VCI;
10.      else
11.         tag the current slot as a regular traffic slot;
12.      wait for the "time slot" to fully elapse;
13.   }forever
14.}
```

Fig. 4. The **SlotManager** process.

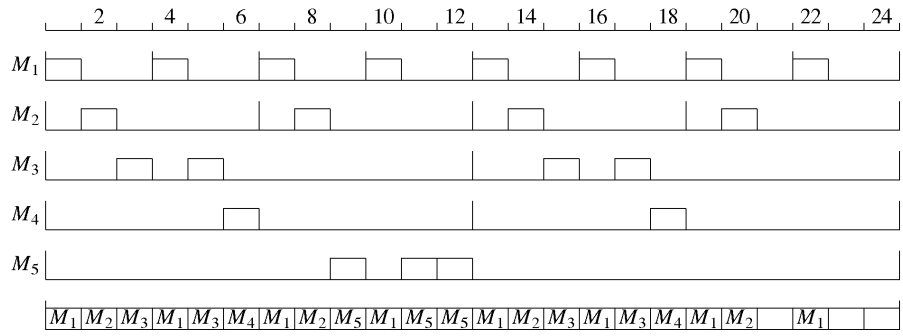


Fig. 5. The slot allocation schedule for the set of message streams **M** = {(1, 4), (1, 7), (2, 13), (1, 23), (3, 28)}.

from the isochronous message streams, especially (C_i, D_i), for each stream M_i and the source and destination station ids of M_i.

Step 2. 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 the same as that used in Scheduler **Sx**, then we find a D'_i for each D_i that satisfies

$$D'_i = x \cdot 2^j \le D_i < x \cdot 2^{j+1} = 2D'_i,$$

for some integer $j \ge 0$, 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 3. Check if the total message density of the specialized stream set $\mathbf{M'} = \{M'_i = (C_i, D'_i) | 1 \le i \le n\}$ is less than or equal to 1, i.e., if $\rho(\mathbf{M'}) = \sum_{i=1}^{n} C_i/D'_i \le 1$. If not, reject **M** and stop. Otherwise, proceed to the next step.

Step 4. Use the on-line allocator, **SlotAllocator**, to obtain the message stream id *i*. If $i > 0$, assign the slot to M_i by filling the slot's VCI field with M_i's VCI. If $i = 0$, mark the slot as a regular traffic slot. Note that **SlotAllocator** will generate an infinite sequence of message stream ids for **M'** such that there are C_i copies of message stream M_i's id within any subsequence of D'_i ($\le D_i$) message stream ids.

Step 5. Wait for the time slot to fully elapse, and repeat Step 4 for the next slot.

EXAMPLE. Consider a set of isochronous message streams, **M** = {(1, 4), (1, 7), (2, 13), (1, 23), (3, 28)}. Specializing the deadline constraint multiset **D** = {4, 7, 13, 23, 28} with respect to {3} yields **D'** = {3, 6, 12, 12, 24}. Since $D'_i | D'_j$, for all $i < j$, and

$$\rho(\mathbf{M'}) = \sum_{i=1}^{5} C_i/D'_i = 1/3 + 1/6 + 2/12 + 1/12 + 3/24 = 21/24 < 1,$$

by Theorem 3, **SlotAllocator** can generate a feasible schedule for **M'**. By applying **SlotAllocator**, we obtain the slot allocation schedule 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* 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 ($\le D_i$) for all *i*.

4.3 Implementation of the Slot Allocation Scheme

The slot allocation scheme can be implemented either as a **SlotManager** daemon (Fig. 4) or as a **SlotManager** chip (Fig. 6) that resides in the slot generator. (Note that the hardware chip performs exactly the same functions as the **SlotManager** daemon.) The advantage of hardware implementation is speed. It needs only $O(1)$ time, instead of $O(n)$, to find the index of the highest-priority active message stream and allocate a slot. However, the maximum number of connections (message streams), *n*, that can coexist in the dual-bus network is limited by the number of modules[2] in **SlotAllocator** used to find the highest-priority active stream.

In order to set up, maintain, and disconnect connections for message streams, **SlotManager**, **SlotAllocator**, and **VCIServer** co-reside in the slot generator station of each bus. **SlotManager** considers each time-constrained connection between a source-destination pair as an isochronous message stream, and gathers/maintains all required information regarding the active connections, especially (C_i, D_i) and the source and destination stations' ids[3] 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 slots (by setting the PA bits and 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 if the total message density after specialization is less than or equal to one. 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 number for the connection. **SlotManager** notifies **SlotAllocator** and the source and destination stations of the acceptance of

2. That is, the building block zoomed out in Fig. 6.

3. The source and destination station ids are collected for the purpose of slot reuse. We will briefly discuss slot reuse in Section 5.2.

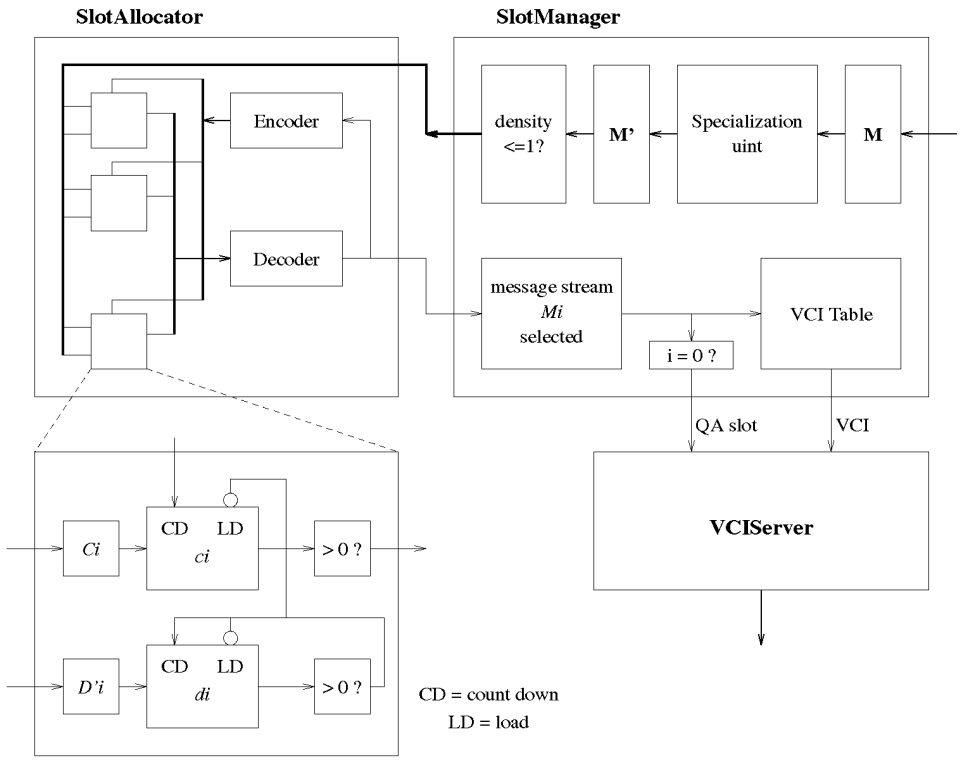


Fig. 6. The proposed slot allocation scheme block diagram.

the new connection. **SlotAllocator** then reestablishes the corresponding new schedule. An existing connection can be cleared (disconnected) 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 regular traffic slots. The interested reader is referred to [30] for a detailed account of how to dynamically establish or terminate a real-time message connection.

5 Comparison with Related Work and Remarks

5.1 Comparison with Related Work

As mentioned in Section 1, the slot allocation scheme proposed by Saha et al. in [45] comes closest to ours but differs in formulating and solving the problem. This difference has yielded a significant consequence explained below.

They adopted the commonly-used peak-rate message model which is, as discussed in Section 2.2, a special case of the message model we used in this paper.

They first devised a slot allocation scheme based on cyclic reservation and derived the schedulability condition under two assumptions:

1) message streams are all periodic with periods $P_i = D_i$, for all i, and
2) message arrivals are aligned with the beginnings 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

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

then the set of message streams is schedulable, where $R_i \stackrel{\Delta}{=} \frac{C_i}{P_i/L}$ is the average slot requirement of 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 handle 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 subsequently modified R_i' as $R_i'' = C_i/((D_i - L)/L) = LC_i/(D_i - L)$. The schedulability condition then becomes

$$\sum_{i=1}^{n} R_i'' = \sum_{i=1}^{n} \frac{LC_i}{(D_i - L)} \le L.$$

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 slot allocation schedule for such a message stream set. Moreover, it is also difficult to choose an appropriate L value if $GCD(D_1, D_2, \ldots, D_n)$ is small. 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 the same as that used in Scheduler **Sx**). Actually, our scheme can find a feasible allocation schedule for a message stream set as long as the total density of the message stream set after specialization is less than or equal to one.

Another shortcoming of their slot allocation scheme is that their scheme requires that the message arrival times are known to the slot manager a priori, which is only true for strictly periodic streams. Therefore, their scheme cannot be used to guarantee message deadlines for streams that conform to the (*C, D*)-smooth message model, but are not periodic. (Note, however, that they also proposed a more complicated implementation which can handle both periodic and sporadic streams and uses the queue arbitration function defined in the DQDB standard. The use of queue arbitration function is out of the scope of this paper.)

5.2 Remarks

In this section, we compare the slot allocation problem with the *distance-constrained* scheduling problem described in [23], [25]. We claim that the solution schedule to the slot allocation problem generated by **SlotManager** can also be used as a schedule for the *discrete-time* version of the *distance-constrained task system* (DCTS) [23], [25]. In the DCTS model, two consecutive executions of the same task must be well-spaced and "close" to each other. Specifically, given a distance-constrained task set $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$, where each task T_i has an execution time C_i and a (temporal) distance constraint D_i, if f_{ij} denotes the finish time of the jth execution/invocation of task T_i, then the distance constraint D_i for T_i requires that $f_{i1} \le D_i$ and $f_{i,j+1} - f_{ij} \le D_i$, for all $j \ge 1$. (In contrast, in the traditional real-time task model [38], every task must be executed once during a certain fixed period. The execution of a task in one period is independent of the execution of the same task in any other period.)

Now consider a distance-constrained task set **T** in which both C_i and D_i are integers, for all i. From the proof of Theorem 3, it is easy to see that a feasible schedule produced by the proposed slot allocation scheme, **SlotManager**, is also a feasible schedule for the task set **T**. For example, the slot allocation schedule in Fig. 5 is a feasible schedule for a distance-constrained task set $\mathbf{T} = \{(C_i, D_i) \mid 1 \le i \le 5\} = \{(1, 4), (1, 7), (2, 13), (1, 23), (3, 28)\}$ (it is easy to check that $f_{i1} \le D_i$ and $f_{i,j+1} - f_{ij} \le D_i$, for $1 \le i \le 5$ and $j \ge 1$).

6 Conclusion

We have proposed a slot allocation scheme for allocating bandwidth and guaranteeing the timely delivery of the messages in isochronous (real-time) streams in a slotted dual-bus network. The dual-bus network configuration is general enough to accommodate several MAC protocols suggested for this topology, e.g., DQDB [32], Fasnet [37], CRMA [41], and *Simple* [36]. The (*C, D*)-smooth message model used in this paper is more general, and includes the peak-rate model and the linear-bounded model as special cases. The proposed slot allocation scheme uses an on-line slot allocation algorithm of polynomial-time complexity (whose correctness is rigorously proved). The resulting scheme is guaranteed to find a feasible slot allocation schedule for a set of message streams as long as the total message density of the set is less than or equal to a certain density threshold. For example, currently, the best density threshold is 0.7 for the double-integer reduction specialization operation.

The message density threshold actually serves as a metric for evaluating the predictability and stability of an allocation scheme. By predictability and stability, we mean that given *any* set of isochronous message streams, as long as its total message density is less than or equal to the derived threshold, a feasible slot allocation schedule is guaranteed to be found by the proposed slot allocation scheme. Moreover, a message stream can be freely added to, or deleted from, the given message set as long as the total message density is held below the threshold. This complements the work done by Liu and Layland [38] in deriving the processor utilization bounds for scheduling computation tasks in a single CPU environment.

We have also addressed the implementation issues of the proposed slot allocation scheme. We configure all the functionalities into three modules: **SlotManager**, **SlotAllocator**, and **VCIServer**, all of which can be implemented as software daemons, or on a VLSI chip. Both implementations are simple and practical.

The performance (in terms of the number of message streams that can be scheduled) of the proposed slot allocation scheme can be improved by using the concept of slot reuse [17], [31], [35], [48], [49], [50]. That is, the cell that has passed on to its destination can be taken out of the slot in order to release the slot for reuse by downstream stations. For example, given that the sources and the destinations of two message streams (connections) M_i and M_j are stations N_i^s and N_j^s, and N_i^d and N_j^d, respectively, if $N_i^s < N_i^d \le N_j^s < N_j^d$, these two connections are actually spatially nonintersecting, and, hence, can use the same virtual connection, i.e., use all the prearbitrated slots assigned to the same VCI. We have studied the problem of grouping spatially nonintersecting message streams into stream subsets. All the message streams in a stream subset use all the prearbitrated slots with the same VCIs for message transmission. The interested reader is referred to [24] for a detailed account.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments on an early draft of this paper. The work reported in this paper was supported in part by the U.S. Office of Naval Research under Grants N00014-92-J-1080 and N00014-94-1-0229, and by the U.S. National Science Foundation under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] G. Agrawal, B. Chen, and W. Zhao, "Local Synchronous Capacity Allocation Schemes for Guaranteeing Message Deadlines with the Timed-Token Protocol," *Proc. INFOCOM*, pp. 186-193, Los Alamitos, Calif., Apr. 1993.

[2] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol," *Proc. IEEE Int'l Conf. Distributed Computing Systems*, pp. 468-475, June 1992.

[3] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing Synchronous Messages Deadlines with the Timed Token Medium Access Control Protocol," *IEEE Trans. Computers*, vol. 43, no. 3, pp. 327-350, Mar. 1994.

[4] "Token Passing Bus Access Method and Physical Layer Specifications," ANSI/IEEE Standard 802.4-1985, 1985.

[5] "Token Ring Access Method and Physical Layer Specifications," ANSI/IEEE Standard 802.5-1985, 1985.

[6] "Fiber Distributed Data Interface (FDDI)—Token Ring Media Access Control (MAC)," Am. Nat'l Standard, ANSI X3.139-1987, 1987.

[7] 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.

[8] H.R. van As, J.W. Wong, and P. Zafiropulo, "Fairness, Priority, and Predictability of the DQDB MAC Protocol Under Heavy Load," *Proc. Int'l Zurich Seminar*, pp. 410-417, Mar. 1990.

[9] C.C. Biskikian, "Waiting Time Analysis in a Single Buffer DQDB (802.6) Network," *IEEE J. Selected Areas in Comm.*, vol. 8, no. 8, pp. 1,565-1,573, Oct. 1990.

[10] W.E. Burr, S. Wakid, X. Qian, and D. Vaman, "A Comparison of FDDI Asynchronous Mode and DQDB Queue Arbitrated Mode Data Transmission for Metropolitan Area Network Application," *IEEE Trans. Comm.*, vol. 42, nos. 2/3/4, pp. 1,758-1,768, Feb./Mar./Apr. 1994.

[11] C. Chan and V.C.M. Leung, "Reservation-Arbitrated Access for Isochronous Voice Transport Over Dual-Bus Metropolitan Area Network," *Proc. ICC '95*, Seattle, June 1995.

[12] M.Y. Chan and F. Chin, "General Schedulers for the Pinwheel Problem Based on Double-Integer Reduction," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 755-768, June 1992.

[13] M.Y. Chan and F. Chin, "Schedulers for Larger Classes of Pinwheel Instances," *Algorithmica*, vol. 9, pp. 425-462, 1993.

[14] B. Chen, G. Agrawal, and W. Zhao, "Optimal Synchronous Capacity Allocation for Hard Real-Time Communications with the Timed Token Protocol," *Proc. 13th Real-Time Systems Symp.*, pp. 198-207, Phoenix, Ariz., Dec. 1992.

[15] R.L. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114-131, Jan. 1991.

[16] R.L. Cruz, "A Calculus for Network Delay, Part II: Network Analysis," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 132-141, Jan. 1991.

[17] M.W. Garrett and S.-Q. Li, "A Study of Slot Reuse in Dual Bus Multiple Access Networks," *Proc. INFOCOM*, pp. 617-629, 1990.

[18] "Manufacturing Automation Protocol," General Motors Corp., version 3.0, implementation release, May 1987.

[19] S.J. Golestani, "Congestion-Free Communication in High-Speed Packet Networks," *IEEE Trans. Comm.*, vol. 39, no. 12, pp. 1,802-1,812, Dec. 1991.

[20] S.J. Golestani, "A Framing Strategy for Congestion Management," *IEEE J. Selected Areas in Comm.*, vol. 9, no. 7, pp. 1,065-1,077, Sept. 1991.

[21] E.L. Hahne, A.K. Choudhury, and N.F. Maxemchuk, "Improving the Fairness of Distributed-Queue-Dual-Bus Networks," *Proc. IEEE INFOCOM '90*, pp. 175-184, June 1990.

[22] E.L. Hahne and N.F. Maxemchuk, "Fair Access of Multi-Priority Traffic to Distributed-Queu-Dual-Bus Networks," *Proc. IEEE INFOCOM '91*, pp. 889-900, Apr. 1991.

[23] C.-C. Han, "Scheduling Real-Time Computations with Temporal Distance and Separation Constraints and with Extended Deadlines," PhD thesis, Technical Report UIUCDCS-R-92-1748, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1992.

[24] C.-C. Han, C.-J. Hu, and K.G. Shin, "On Slot Reuse for Isochronous Services in DQDB Networks," *Proc. IEEE 16th Real-Time Systems Symp.*, pp. 222-231, Dec. 1995.

[25] C.-C. Han and K.-J. Lin, "Scheduling Distance-Constrained Real-Time Tasks," *Proc. IEEE Real-Time Systems Symp.*, pp. 300-308, Dec. 1992.

[26] C.-C. Han and K.G. Shin, "A Polynomial-Time Optimal Synchronous Bandwidth Allocation Scheme for the Timed-Token MAC Protocol," *Proc. IEEE INFOCOM '95*, Boston, Apr. 1995.

[27] C.-C. Han, K.G. Shin, and C.-J. Hou, "Synchronous Bandwidth Allocation for Real-Time Communications with the Timed-Token MAC Protocol," submitted to *J. ACM*.

[28] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel, "Pinwheel Scheduling with Two Distinct Numbers," *Theoretical Computer Science*, vol. 100, no. 1, pp. 105-135, June 1992.

[29] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem," *Proc. IEEE 22nd Hawaii Int'l Conf. Systems Science*, pp. 693-702, Jan. 1989.

[30] C.-J. Hou and K.S. Tsoi, "Dynamic Real-Time Channel Setup and Tear-Down in DQDB Networks," *Proc. IEEE 16th Real-Time Systems Symp*, pp. 232-241, Dec. 1995.

[31] N.-F. Huang, H.-I. Liu, and G.-K. Ma, "On the Reuse of Isochronous Channels in DQDB Metropolitan Area Networks," *Proc. Int'l Conf. Comm.*, pp. 956-959, 1994.

[32] "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.

[33] "IEEE Standard for LANs and MANs: Supplements to DQDB Access Method and Physical Layer Convergence Procedure (PCLP) for DS-1 Based Systems and Isochronous Service on a DQDB Subnetwork of a MAN," 1994.

[34] W. Jing and M. Paterakis, "Message Delay Analysis of the DQDB Subnetwork Based on an Approximate Node Model," *IEEE Trans. Comm.*, vol. 42, nos. 2/3/4, pp. 1,120-1,130, Feb./Mar./Apr. 1994.

[35] A.E. Kamal, "Efficient Multi-Segment Message Transmission with Slot Reuse on DQDB," *Proc. INFOCOM '91*, pp. 869-878, Apr. 1991.

[36] J. Limb, "A Simple Multiple Access Protocol for Metropolitan Area Networks," *Proc. SIGCOMM*, pp. 67-79, Philadelphia, Sept. 1990.

[37] J.O. Limb and C. Flores, "Description of Fasnet—A Unidirectional Local Area Communications Network," *Bell Systems Technical J.*, vol. 61, pp. 1,413-1,440, Sept. 1982.

[38] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogamming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[39] P. Martini and G. Wershmann, "Real-Time Communication in DQDB: A Comparison of Different Strategies," *Proc. 17th Conf. Local Computer Networks*, 1992.

[40] P. Martini, "Connection Oriented Data Service in DQDB," *Computer Networks ISDN Systems*, vol. 26, pp. 679-694, 1994.

[41] M.M. Nassehi, "CRMA: An Access Scheme for High-Speed LAN's and MAN's," *Proc. SUPERCOMM/ICC*, pp. 1,697-1,702, Atlanta, Apr. 1990.

[42] C. Partridge, *Gigabit Networking*. Reading, Mass.: Addison-Wesley, 1994.

[43] P. Potter and M. Zukerman, "Cyclic Request Control for Provision of Guaranteed Bandwidth within DQDB Framework," *Proc. Int'l Switching Symp.*, 1990.

[44] N.S.V. Rao, K. Maly, and S. Dharanikota, "Average Waiting Time Profiles of Uniform DQDB Model," *Proc. IEEE INFOCOM '94*, vol. 1, pp. 1,326-1,333, June 1994.

[45] D. Saha, M.C. Saksena, S. Mukherjee, and S.K. Tripathi, "On Guaranteed Delivery of Time-Critical Messages in DQDB," *Proc. IEEE INFOCOM '94*, vol. 1, pp. 272-279, June 1994.

[46] L. Sha, S. Sathaye, and J.K. Strosnider, "Scheduling Real-Time Communication on Dual-Link Networks," *Proc. IEEE Real-Time Systems Symp.*, pp. 188-197, Dec. 1992.

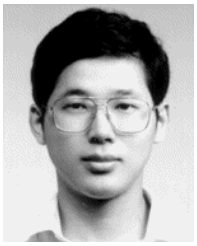
[47] L. Sha, S.S. Sathaye, and J.K. Strosnider, "Analysis of Dual-Link Networks for Real-Time Applications," *IEEE Trans. Computers*, vol. 46, no. 1, pp. 1-13, Jan. 1997.

[48] O. Sharon and A. Segall, "A Simple Scheme for Slot Reuse without Latency for a Dual Bus Configuration," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, pp. 96-104, Feb. 1993.

[49] O. Sharon and A. Segall, "On the Efficiency of Slot Reuse in the Dual Bus Configuration," *IEEE/ACM Trans. Networking*, vol. 2, no. 1, pp. 89-100, Feb. 1994.

[50] O. Sharon and A. Segall, "Schemes for Slot Reuse in CRMA," *IEEE/ACM Trans. Networking*, vol. 2, no. 3, pp. 270-278, June 1994.

[51] J.S. Turner, "New Directions in Communications (Or Which Way to the Information Age)," *IEEE Comm. Magazine*, vol. 24, no. 10, pp. 8-15, Oct. 1986.



Ching-Chih (Jason) Han received the BS degree in electrical engineering from National Taiwan University, Taiwan, Republic of China, in 1984, the MS degree in computer science from Purdue University, West Lafayette, Indiana, in 1988, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1992.

From August 1992 to January 1994, he was an associate professor in the Department of Applied Mathematics at National Sun Yat-sen University, Kaohsiung, Taiwan. From February 1994 to July 1996, he was a visiting associate research scientist in the Real-Time Computing Laboratory at the University of Michigan, Ann Arbor. Since August 1996, he has been with the Department of Electrical Engineering at The Ohio State University, where he is currently an assistant professor. His current research interests include computer/communication networks, high-speed networking, real-time computing/scheduling, wireless communications, multimedia applications, and parallel and distributed systems.

Chao–Ju Hou received the BSE degree in electrical engineering in 1987 from National Taiwan University, the MSE degree in electrical engineering and computer science (EECS), the MSE degree in industrial and operations engineering, and the PhD degree in EECS, all from The University of Michigan, Ann Arbor, in 1989, 1991, and 1993, respectively. From August 1993 to July 1996, she was an assistant professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. Since August 1996, she has been with the Department of Electrical Engineering at The Ohio State University-Columbus, where she is currently an assistant professor.

She is a recipient of the U.S. National Science Foundation CAREER award, Wisconsin/Hilldale Undergraduate/Faculty Research Fellowships, and Women in Science Initiative Awards in Wisconsin. Her research interests are in the areas of distributed and fault-tolerant computing, design and implementation of middleware services that provide QoS control and monitoring in high-speed networks, and performance modeling/evaluation. She has served on the program committees of several IEEE conferences, and is a member of the IEEE, IEEE Computer Society, ACM Sigmetrics, and Society of Woman Engineers.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor, Michigan.

He has authored/coauthored more than 360 technical papers (about 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He has written (jointly with C. M. Krishna) a textbook, *Real-Time Systems*, published by McGraw-Hill in 1996. In 1987, he received the Outstanding *IEEE Transactions on Automatic Control* Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from the University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. (The latter is being pursued as a key thrust area of the newly established U.S. National Science Foundation Engineering Research Center on Reconfigurable Machining Systems.)

From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California at Berkeley, and International Computer Science Institute, Berkeley, California, IBM T.J. Watson Research Center, and the Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division, EECS Department, the University of Michigan, for three years beginning in January 1991.

He is an IEEE fellow, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, a program cochair for the 1992 International Conference on Parallel Processing, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the Computer Society of the IEEE, an editor of *IEEE Trans. on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.